OpenSolaris Hot Topic Seminar 2010.03

opensolaris™

「IPS」のパッケージ作成入門

～だれでもcontributer計画～

OpenSolaris Users Group Leader

ジャストプレイヤー株式会社

瀧 康史 /TAKI, Yasushi

Agenda

概要
IPSのパッケージをなぜ作る必要があるのか?

OSD CBE(Common Build Environment)の作成
共通のビルド環境を自分のところにつくろう!

Countirutorになろう
SourceJuicerにアップロードしよう

Appendix:より実践的なポーティング
実際にビルド時につまずくこととはなにか?

公開用IPSサーバを設定する
独自IPSサーバをたてよう!

概要

Configure && make install よりも…

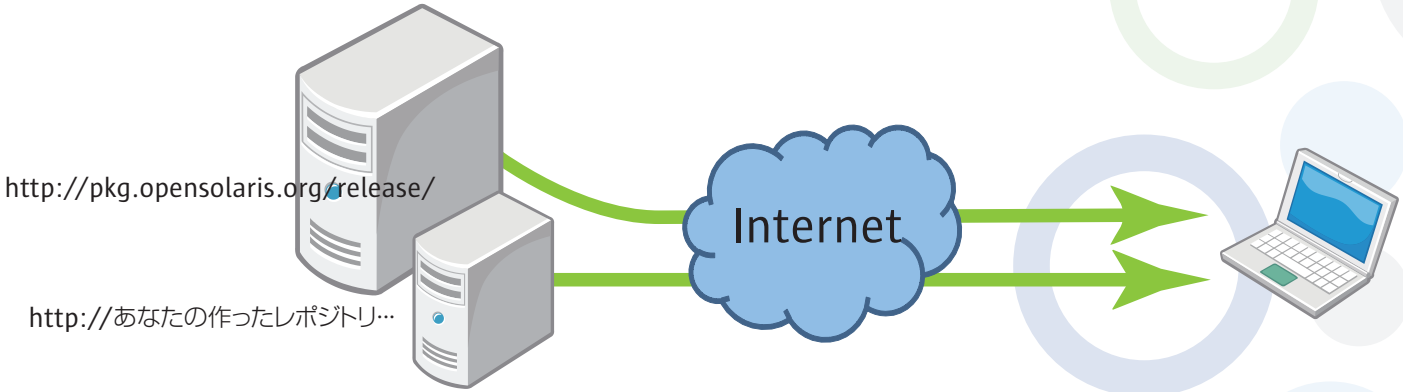
IPS パッケージを作ろう!

なぜパッケージ化するのでしょうか?

- 自分のため
  - インストールしたパッケージを再びインストールするときに楽だから。
  - 同じ環境を作り込むのが楽だから→依存関係の自動解決等。
  - アンインストールが簡単だから。
    - beadmを使えば、configure && makeでも、直後にならアンインストールできますが
  - アップデートが簡単だから。
- 人のため
  - 同じソフトをインストールしたい人が楽だから
  - そのソフトをアップデートするのも楽だから
- 会社利用
  - 自社ソフト、自社ハードのドライバの配布のため
  - 管理サーバの環境をそろえる等

SVr4 のパッケージとの違い

- IPSには、SVr4パッケージのような.pkg等、ローカルに保存できる形式がありません。
  - pkgファイル単品の配布ができません。
  - 配布のためには必ず「IPSサーバ」をたてる必要があります。
- パッケージはアップデートのしやすさと、**依存関係の整理を意識**して作られています。
- SVr4のパッケージとIPSのパッケージには、パッケージ名の互換がありません。
- SVr4のpkgにできて、IPSにできないこともあります。
  - 考え方と方針が違うためです。
- ユーザは、複数のレポジトリを登録することができます。



なぜ、IPS が必要なのか？

それは、OpenSolarisがIPSで管理されているから!

他にもこんな理由があります。

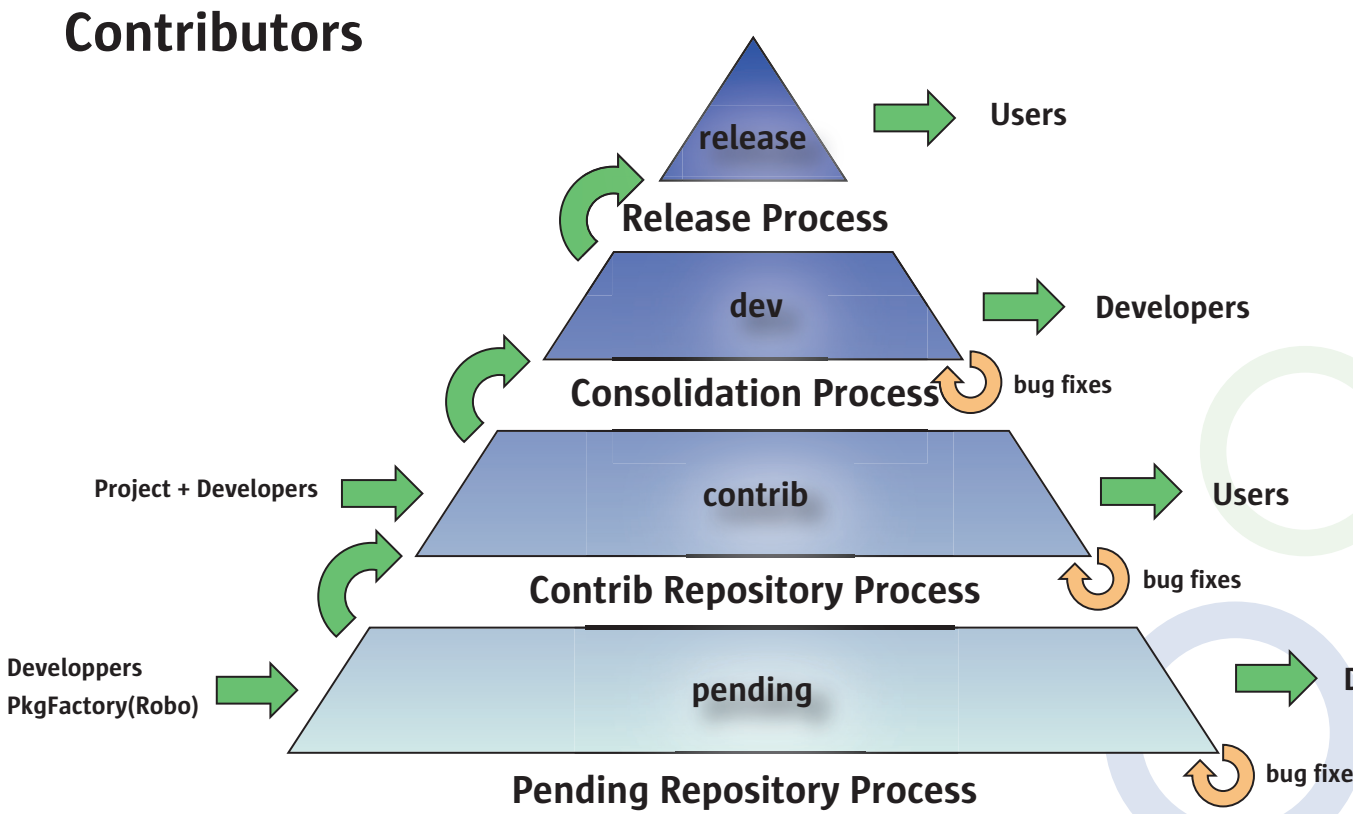
- インストール方法を教えるのが簡単になる。ドキュメントもシンプルになる。
- アップデートが簡単なので、配布元が「最新環境」を配布しやすい。
- ローカルOS内のファイル管理を一元化しやすい。
  - ローカル環境内のデータに矛盾が起きやすくなる。
  - パッケージシステムが増えると、管理が大変になる。
  - 基本、レポジトリは混ぜたら危険！！
- 依存関係の問題
  - IPSで管理されたシステムに、SVr4のレポジトリが入ることは、異なった管理システムで、異なるレポジトリを管理する事と等価。
  - SVr4の.pkg形式でインストールされたドライバに依存するアプリを簡単にインストールさせることは難しい！！

依存される可能性があるものは、IPSで配布すべき!!

opensolaris.org のレポジトリ

opensolaris.orgで提供されているレポジトリには、次のものがあります。

それぞれのレポジトリのポリシーを、考えて使う必要があります。



release & dev

OSのcoreとミドルウェア

contrib & pending

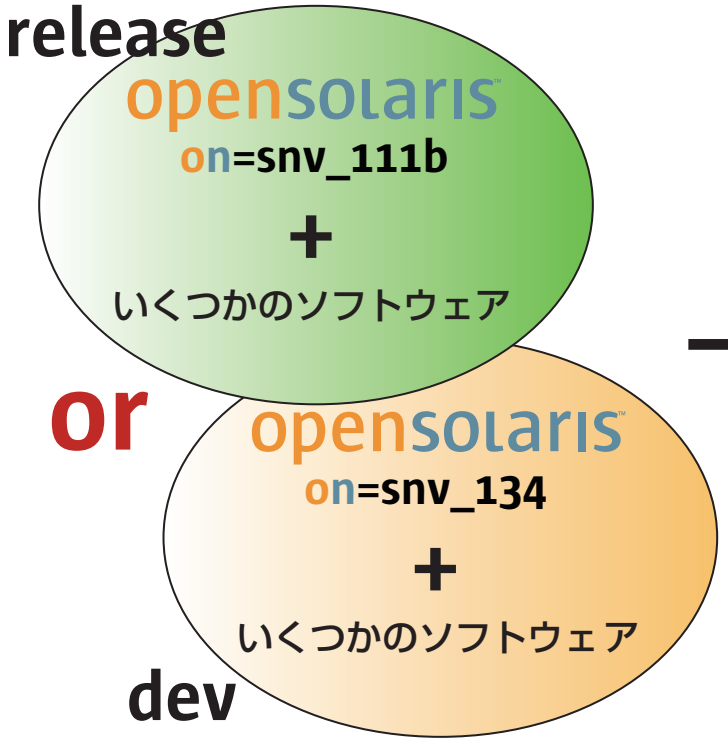
コントリビュータが提供するパッケージ。

「contrib」に入れば、多くのOpenSolarisユーザが簡単にソフトを利用できる。

OSSデベロッパーとしての目標!!

+ contrib , +pending

contribとpendingは、release or devに追加するパッケージです。



+

contrib

- 安定したソフトが投入されている。
- 多くのユーザがまずは追加する。

```
pkg set-publisher -O http://pkg.opensolaris.
  org/contrib contrib
```

pending

- パッケージ開発中のもの。
- 必要なときだけenableにすると良い。

```
pkg set-publisher --disable -O http://jucr.
  opensolaris.org/pending jucr.opensolaris.org
```

利用前に

```
pkg set-publisher --enable jucr.opensolaris.org
```

ソフトウェアの配布のために

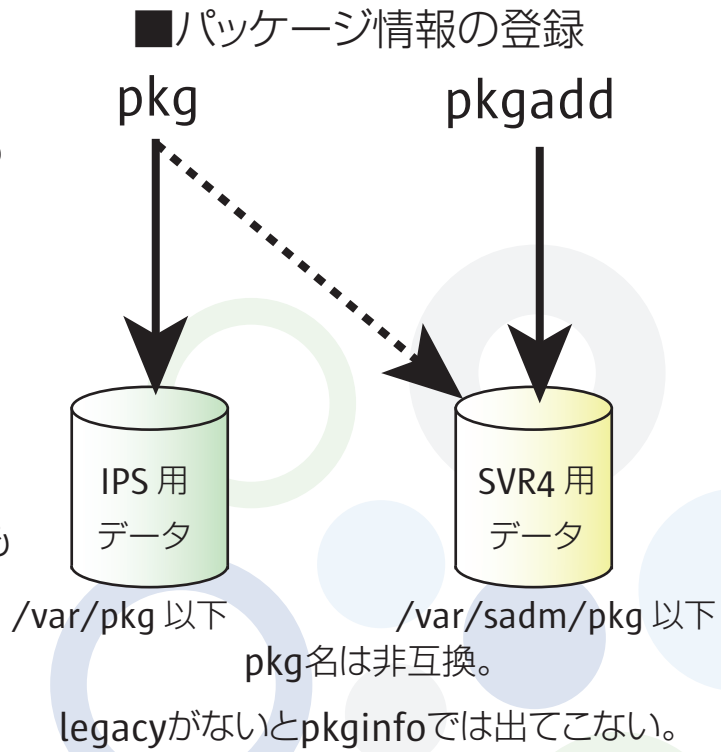
ソフトウェアの動作環境を作るためには、様々なお膳立てが必要

- OSのバージョンが合っているか?
- 利用しているミドルウェアはすべて入っているか?
- ミドルウェアの設定はきちんと行われているのか?

これらをインストールスクリプトで行うのは難しい。

次の考慮も重要

- 自分達(自社)が配布したソフトへ依存するソフトもありうるか?
- その先はどうか?



様々なソフトが依存し合う環境では、OSのパッケージ管理に合わせておく方がよい。

OpenSolaris Desktop Common Build Environment

～Source Juicerへの道のり～

OSD CBE

ビルド環境の統一化(Common Build Environment)

- xxは簡単にビルドできるよ。ただし、僕の環境ではね…では、困る。
  - コンパイラ、環境変数、PATHの指定、パッチなどをみんな同じ環境で!
- すべてのポーター(Softwareをポーティングする人)が統一した環境下でビルドができる必要がある。

ポーター向けビルド方法のノウハウを1つのファイル(spec)に!

従来は、次のものをノウハウとして共有していました。

- ダウンロードサイト(tar ballの存在場所)
- パッチプログラム
- ビルド方法の記載(BUILD.shとかつくりませんでしたか?)
  - 特殊なコマンド入力
  - configure、 makeのオプション

統一の環境(Build Grid)でビルドしたい!

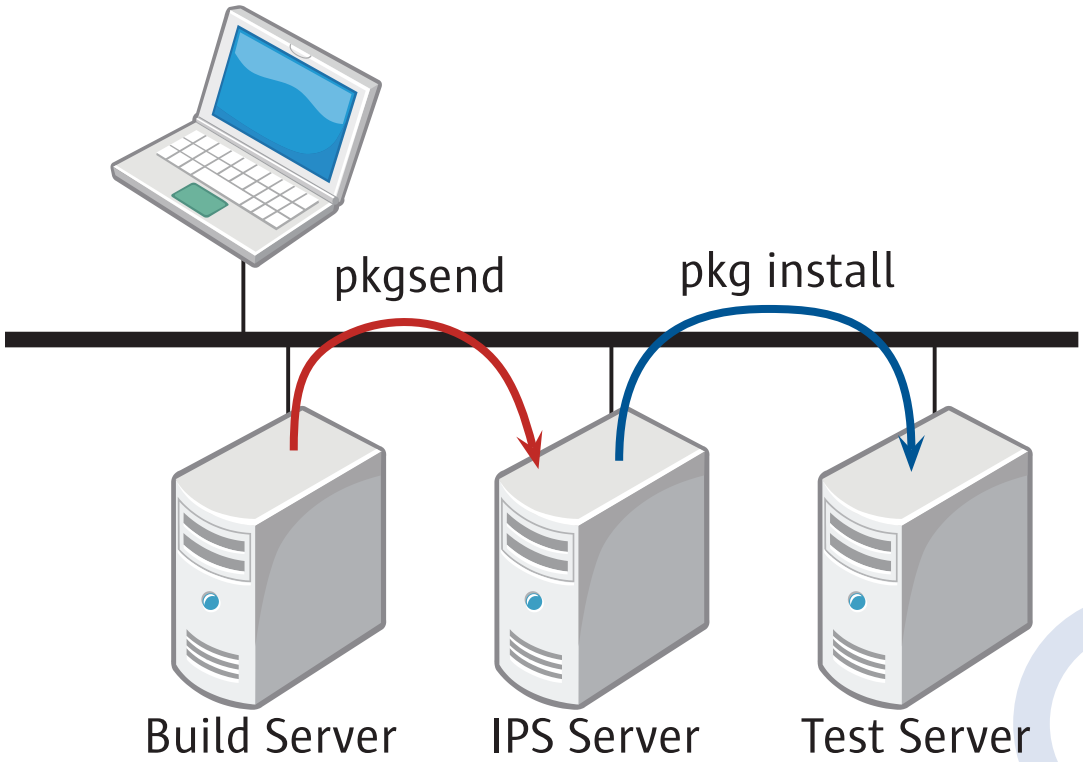
つまり、それがSource Juicer!

つまり・・・こんな流れです

- ローカル環境の準備（最初に1度だけ）
  - OSD CBEの作成
  - ローカルIPSサーバの作成
  - テストサーバの作成
- ローカルでIPSパッケージの作成
  - specファイルを書く。
  - ビルドに成功する。
  - 自分のマシンでインストールテストする。
- SourceJuicer
  - Submit: specファイル、Patch、コピーライトなどをsource juicerにアップロードする
  - Validate: aproverから、コピーライト承認がおりる
  - Build: ビルドグリッドが動いて自分のプログラムがコンパイルされるのを待つ
  - PASSしたら、めでたくpendingに入る。
  - インストールテストをする。

OSD CBE の作成の準備

ビルド用、環境テスト用、レポジトリ用と3つの役割をもつサーバが必要になる。



ビルド用

ビルドに必要なパッケージがインストールされた、コンパイルマシン。

ローカル用IPSサーバ

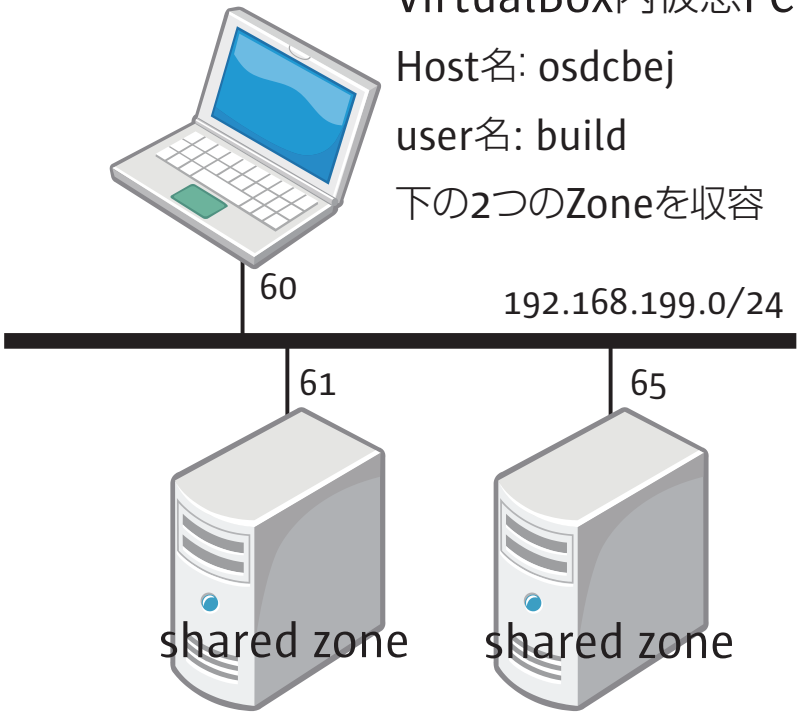
svc:/application/pkg/server (pkg.depod)を起動

テストマシン

動作テスト用。

プレーンな環境に、何度でも戻せる必要あり。

VirtualBox 環境



Zone Host 名 : work-spec
user名: osduser
ビルドサーバ兼IPSサーバ

Zone Host 名 : testzone
ユーザなし
テスト用zone

今回用意した環境

- SourceJuicerと同じバージョンのOSの必要があるため、VirtualBoxに隔離環境を用意。
- そのVirtualBoxの上に、2つのZone。
  - work-spec
    - ビルド兼IPSサーバ用
  - testzone
    - プレーンなテスト環境

※これら2つのzoneが入っているVirtualBoxのovfイメージを配布

```
http://dist.justplayer.com/vmimages/
  SourceJuicer/2009.06/SourceJuicer.7z
```

OSD CBE の作成 #1

※これらの処理は、前ページのwork-spec上で行っています。
※細かなログは、http://kohju.justplayer.com/Tips_Solaris_compile_JDSenv.html 参照。

オペレータユーザの作成

ZONE名	work-spec
オペレータユーザ	osduser
プロファイル	Primary Administrator
ZFS	rpool/zones/work-spec/ROOT/export/home/osduser
HOME	/export/home/osduser
SHELL	/bin/zsh
初期パスワード	osduser

```
pfexec zfs create -o mountpoint=/export rpool/zones/work-spec/ROOT/export
pfexec zfs create -p rpool/zones/work-spec/ROOT/export/home/osduser
pfexec pkg install -v SUNWzsh
pfexec useradd -m -d /export/home/osduser -s /bin/zsh -P 'Primary Administrator,Software
  Installation' osduser
pfexec chown osduser:other /export/home/osduser
pfexec passwd osduser
```

以後、特に記載がない場合、''osduser''でログインしているものとします。

OSD CBE の作成 #2

事前に入れておいたほうが良いもののインストール

Contribレポジトリの追加

```
pfexec pkg set-publisher -O http://pkg.opensolaris.org/contrib contrib
```

開発環境のインストール

```
pfexec pkg install -v \
 SUNWgtar \
 SUNWxcu4 \
 SUNWcar \
 SUNWkvm \
 SUNWwget \
 SUNWpkgcmds \
 SUNWrsync \
 ss-dev sunstudio12u1\
 gcc-dev \
 SUNWgnome-common-devel \
 SUNWperl-xml-parser \
 SUNWgnome-xml-root SUNWgnome-xml SUNWgnome-xml-share \
 SUNWgpch SUNWsprot SUNWhea SUNWsfwhea SUNWxwinc SUNWxorg-headers SUNWgnu-coreutils
  SUNWgnu-diffutils
```

OSD CBE の作成 #3

コンパイラについて

いくつかのソフトは自分がコンパイルされた環境を覚えており、拡張モジュールのコンパイルに自分と同じコンパイラを要求します。

OpenSolarisはON（OS+NET）がStudio12でコンパイルされているため、これを拡張するモジュール群が、ONと同じコンパイラを要求することがあります。トラブルを避けるためには、デフォルトをStudio12にしておくとよいのですが、12のインストールはちょっと面倒なので、ここでは簡単に利用出来る12u1を利用します。

従来、SUN　Studioは、/opt/SUNWsproにインストールされておりました。そこで、下記の様にシンボリックリンクを張っておきます。

```
pfexec ln -s /opt/sunstudio12.1 /opt/SUNWspro
```

次ページにあるcbe-install中においても、コンパイラ（開発環境）は、gcc-dev、ss-dev（StudioExpress）、Studio12、Studio12u1が選べます。

SourceJuicerは、デフォルトで、「/opt/SunStudioExpress/bin/cc」を利用しますが、specファイル内で明記しておけば、gccなどの利用も可能です。

OSD CBE の作成 #4

OSD CBE 1.7.0RC1をインストール

このプログラムは、SVr4のパッケージでインストールされます。

```
wget \
http://dlc.sun.com/osol/jds/downloads/cbe/test/desktop-cbe-1.7.0-rc1-x86.tar.bz2
gtar jxvf desktop-cbe-1.7.0-rc1-x86.tar.bz2
cd desktop-cbe-1.7.0-rc1
./cbe-install
```

質問がいっぱい出てきますが、よく読めば、ほとんどデフォルトのままで動作します。

例が欲しい場合は、私のブログを参照してください。

```
http://kohju.justplayer.com/Tips_Solaris_compile_JDSenv.html
```

OSD CBE の作成 #5

pkgbuildのアップデート

ods-cbeに含まれているpkgbuildは古いため、IPSに対応していません。そこで最新版に入れ替えます。

1.3.1をアンインストール

```
pfexec pkgrm SFpkgbuild
```

Source Juicer上のバージョンに入れ替えるため、juicerを追加します。

```
pfexec pkg set-publisher -O http://jucr.opensolaris.org/pending jucr.opensolaris.org
```

インストールします。

```
pfexec pkg install -v pkgbuild
```

pfexec pkg install -v pkgbuildjuicerのレポジトリは開発用なので、「毎日」何かがカレントで書き換わる。次のようにして、レポジトリのメタ情報のリフレッシュは頻繁にすると良いでしょう。

```
pfexec pkg refresh --full juicer
```

OSD CBE の作成 #6

ACLOCALの修正

dirlistへの追加

/opt/dtbld/share/aclocal以下に、CBE m4のファイルが見つからないと、コンパイル時にACLOCALでエラーが出ます。この場合、/usr/share/aclocal/dirlistに、下記のものがあるか確認する必要があります。

```
/usr/sfw/share/aclocal
/opt/dtbld/share/aclocal
```

なければ追加します。

```
pfexec sh -c 'echo /opt/dtbld/share/aclocal >> /usr/share/aclocal/dirlist'
```

aclocalの名前解決

OpenSolarisは、aclocalという名で実行できないのでシンボリックリンクで回避します。

```
ln -s /usr/bin/aclocal-1.10 /usr/bin/aclocal
```

ビルド用ユーザ環境の設定 #1

cbe-setupによって、~osduser/packages/以下に、ワークディレクトリが作られているはずです。

```
packages/
|-- BUILD                                  ← tar ballから展開。ビルドディレクトリ
|   `-- CBEenv-1.7.0-rc1
|-- PKGMAPS
|   |-- copyright                          ← copyright宇が保存される
|   |-- depend                             ← SVr4用の依存関係ファイルの保存
|   |   `-- CBEenv.depend
|   |-- manifest                           ← IPS、pkgsend用 manifest
|   |-- pkginfo                            ← SVr4用のpkginfo（メタ情報）ファイル
|   |   |-- CBEenv-src.pkginfo
|   |   `-- CBEenv.pkginfo
|   |-- proto                              ← SVr4用のproto（ファイル一覧）ファイル
|   |   |-- CBEenv-src.proto
|   |   `-- CBEenv.proto
|   `-- scripts                            ← SVr4 PKG用、IPS登録用のスクリプト
|       `-- CBEenv.preremove
|-- PKGS                                   ← パッケージの仮想root
|   `-- CBEenv
|       |-- install
|       |   |-- depend
```

ビルド用ユーザ環境の設定 #2

```
|       |   `-- preremove
|       |-- pkginfo
|       |-- pkgmap
|       `-- reloc
|           `-- bin
|               |-- env.csh
|               |-- env.sh
|               |-- env_include.sh
|               |-- gendiff
|               `-- ld-wrapper
|-- SOURCES                                   ← tar ball、パッチ、追加ファイルなど
|   |-- env.csh
|   |-- env.sh
|   |-- env_include.sh
|   |-- gendiff
|   `-- ld-wrapper
|-- SPECS                                     ← include用のspecファイル
|   |-- CBE.inc
|   `-- default-depend.inc
`-- SPKGS                                     ← ソースパッケージ（rpmのsrpmに相当）
    `-- CBEenv-src
        |-- pkginfo
        |-- pkgmap
```

ビルド用ユーザ環境の設定 #3

```
`-- reloc
    `-- share
        `-- src
            `-- CBEenv-1.7.0-rc1
                |-- SOURCES
                |   |-- env.csh
                |   |-- env.sh
                |   |-- env_include.sh
                |   |-- gendiff
                |   `-- ld-wrapper
                `-- SPECS
                    |-- CBE.inc
                    |-- CBEenv.spec
                    `-- default-depend.inc
```

ビルド用ユーザ環境の設定 #4

環境変数の設定

cbe-setup実行後の最後に書かれているメッセージの通り、下記の通り設定すると、環境が読み込まれます。

sh系

```
. /opt/dtbld/bin/env.sh
```

csh系

```
source /opt/dtbld/bin/env.csh
```

ログイン後にいちいち実行するのも面倒なので、ビルド専用ユーザであるosduserに、/opt/dtbld/bin/env_include.shを投入しておきます。

```
echo . /opt/dtbld/bin/env_include.sh >> ~/.zshrc
echo init_dt_cbe  >> ~/.zshrc
```

※シェルは適当に選んでください。

workディレクトリの作成

適当に掘っておきます。

```
mkdir ~/work/
```

ビルド用ユーザ環境の設定 #5

サンプルのspecの取得

サンプルのspecファイルを取得します。spec-filesのincludeファイルと、いろいろなincludeファイルを取得します。下記の例ではtrunkを引っ張ってきます。

ダウンロード(レポジトリから取得なので多少時間はかかります)

```
cd ~/work/
svn co svn+ssh://anon@svn.opensolaris.org/svn/jds/spec-files/trunk spec-files
```

gnomeなどのbuild環境が必要であるならば、下記を参照してください。

* http://hub.opensolaris.org/bin/view/Project+jds/building
    - 4. OSD Build Sources

特別gnome系のソースがいらない場合は、この中にある基本的なspecファイル用のincludeファイルがあります。これをコピーします。

```
cp ~/work/spec-files/include/*.inc ~/packages/SPECS/
```

以上で、OSD-CBEの環境構築は終わりです。

開発用 IPS サーバの設定

pkg/serverは、SUNWipkgに含まれているので、pkgコマンドとともにインストールされています。

右は、起動の様子とログの状態です。

```
pfexec svcadm enable pkg/server
```

デフォルトでは、

- ポート80でListen
- 読み書き両用（pkgsend可能）

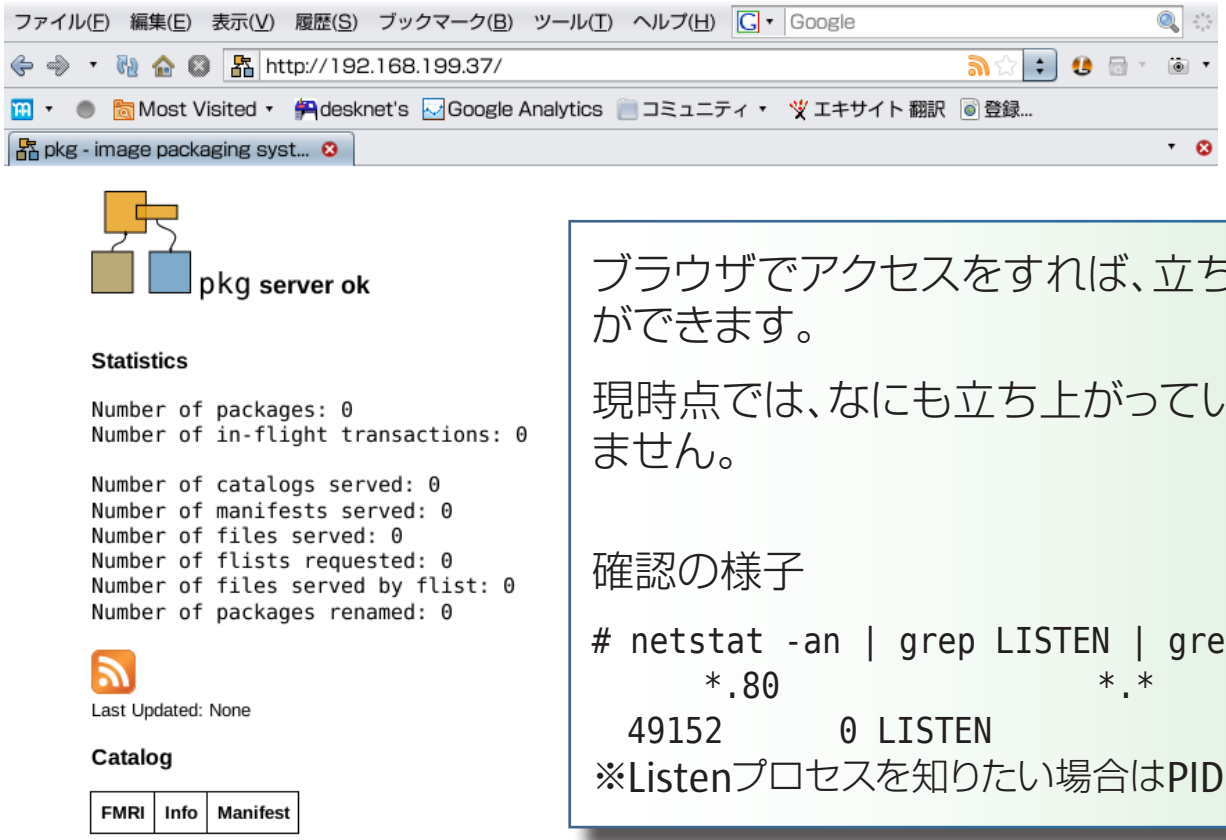
で、レポジトリが立ち上がっています。

ビルドサーバにも、登録をします。

```
pfexec pkg set-publisher -O http://
  localhost/ mypkgs
```

```
root@test-ips:~# svcs -xv pkg/server
svc:/application/pkg/server:default (image packaging repository)
 State: disabled since Thu Apr 02 16:04:43 2009
Reason: Disabled by an administrator.
   See: http://sun.com/msg/SMF-8000-05
Impact: This service is not running.
root@test-ips:~# svcadm enable pkg/server
root@test-ips:~# svcs -xv pkg/server
svc:/application/pkg/server:default (image packaging repository)
 State: online since Thu Apr 02 16:05:32 2009
   See: /var/svc/log/application-pkg-server:default.log
Impact: None.
root@test-ips:~#       cat       /var/svc/log/application-pkg-
  server:default.log
[ Apr  2 16:05:32 Enabled. ]
[ Apr  2 16:05:32 Executing start method ("/lib/svc/method/svc-
  pkg-depot start"). ]
ppriv  -s  A=basic,-file_link_any,-proc_info,-proc_session,net_
  privaddr -e /usr/lib/pkg.depotd -d /var/pkg/repo -p 80 -s
  10 -t 60 --content-root=/usr/share/lib/pkg --log-access=none
  --log-errors=stderr
[02/Apr/2009:16:05:32] INDEX Search Available
[02/Apr/2009:16:05:32] ENGINE Listening for SIGHUP.
[02/Apr/2009:16:05:32] ENGINE Listening for SIGTERM.
[02/Apr/2009:16:05:32] ENGINE Listening for SIGUSR1.
[02/Apr/2009:16:05:32] ENGINE Bus STARTING
[02/Apr/2009:16:05:32]  ENGINE  Started  monitor  thread  '_
  TimeoutMonitor'.
[02/Apr/2009:16:05:33] ENGINE Serving on 0.0.0.0:80
[02/Apr/2009:16:05:33] ENGINE Bus STARTED
```

立ち上がったIPSの確認



ブラウザでアクセスをすれば、立ち上がった、状態の確認ができます。

現時点では、なにも立ち上がっていないので、なにもありません。

確認の様子

```
# netstat -an | grep LISTEN | grep *.80
      *.80                 *.*                0      0
  49152      0 LISTEN
```

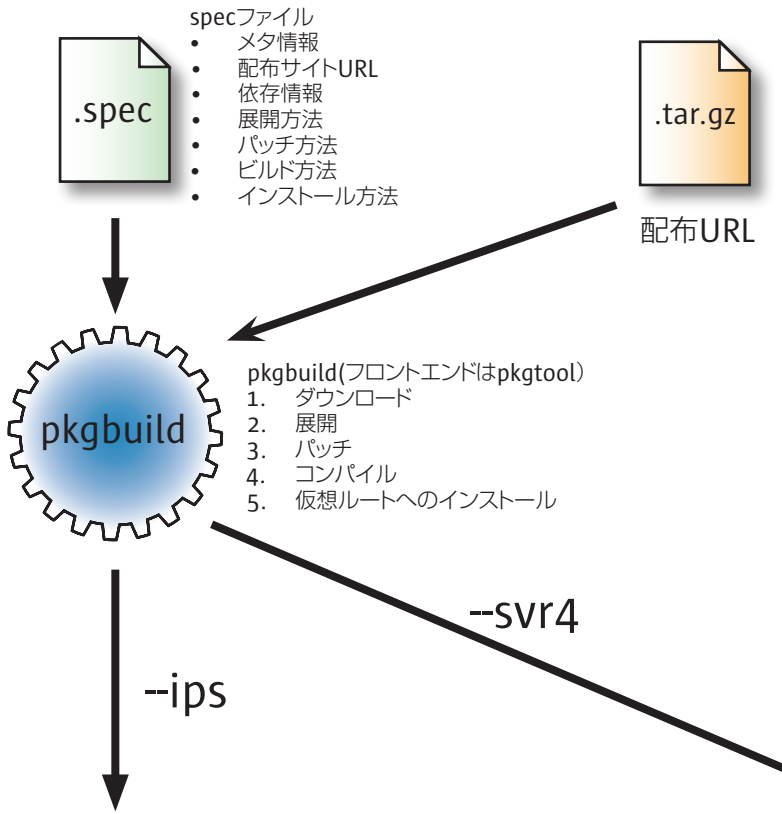
※Listenプロセスを知りたい場合はPIDをしらべてpfilesする

※配布するVirtualBoxの環境は、ここまでできています。

specファイルを作成する

specファイルを作成して
IPSに登録してみよう!

pkgbuild を取り巻く流れ



pkgbuild(フロントエンドはpkgtool)は、specファイルを用いて次のことを行います。

1.ソース配布URLから自動的にファイルをダウンロード

2.ビルド

3.IPSに登録(SVr4のpkg作成)

用意するファイルは、

- spec
- copyright
- パッチ(あれば)

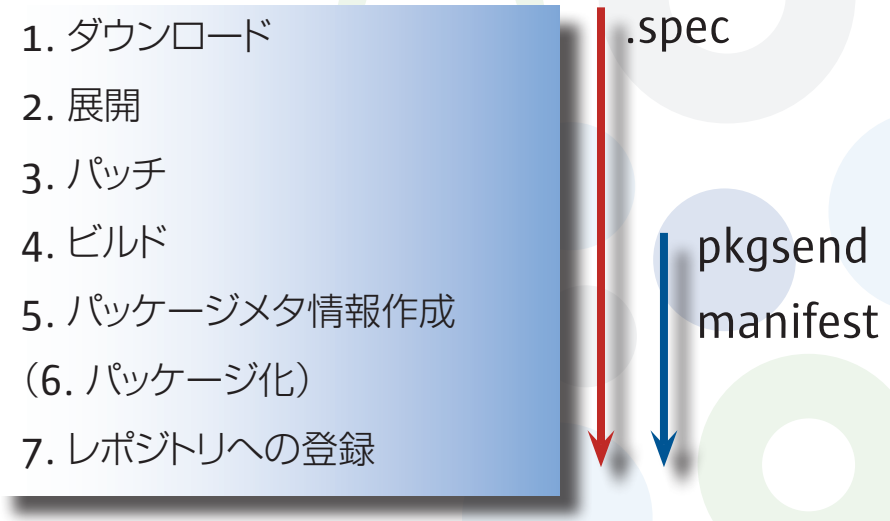
程度です。

1. IPS用manifest作成
2. IPSへ登録

1. SVR4用ファイル作成(proto/pkginfo/depend等)
2. pkg化

specとpkgsend manifestの関係

- specファイルはRedHat Linux系からきた考え方
  - 移植品ではなく、perlでかかれた異なる実装（全般的にexperimental）
  - rpmのspecとは、似て非なるもの。
- Spec File Extra（SFE）という、プロジェクトの派生
  - もともとのSFEはspecを配り、「SVr4のパッケージを作ってインストールする」ためのものでした。
- IPSのpkgsend manifestよりも、さらにバックヤードの考え方を持ちます。
  - 配布サイトの場所
  - パッチ
  - ソースのビルド方法
  - パッケージの構成情報



開発環境の work ディレクトリ構造

参考として、私が使ってるサンプルのディレクトリ構造です。

```
work/pkglabo
|-- bin              ← 便利スクリプト群
|   |-- mk-Requires.pl
|   |-- pkgsend_manifest.sh
|   `-- specbuild.sh
|-- manifests        ← specファイルでは作れないmanifest群
|   |-- GNUmakefile
|   |-- JPCenvcmds.lst
|   |-- JPCenvcmds.manifest
|       ～割愛～
`-- specs            ← specファイル群
    |-- eb.copyright
    |-- eb.spec
    |-- ebview.copyright
    |-- ebview.spec
    |-- lv.copyright
    |-- lv.spec
    |-- only-depend.spec.sample
    |-- patches
    |   `-- lv-01-kohju.diff
    |-- xz.copyright
    `-- xz.spec
```

specbuild.sh

```
#!/bin/sh

PKGTOOL=/opt/dtbld/bin/pkgtool
SOURCES=~/packages/SOURCES/

SPEC=$1

if [ \! -f ${SPEC} ]; then
    echo specbuild.sh file.spec
fi
NODE=${1%.spec}

${PKGTOOL} build-only --patchdirs=`pwd`/patches
  --sourcedirs=`pwd` --ips --download ${SPEC}
```

http://dist.justplayer.com/src/pkglabo に公開

specのサンプル

specファイルのセクションについて

参考）http://jucr.opensolaris.org/help/spec_file

基本構造は大体次の通りです。

1. メタ情報セクション	メタ情報を記述する
2. %prep/%setupセクション	アーカイブを展開し、ディレクトリを作成する。
3. %buildセクション	パッチや、ビルドのためのスクリプトを書きます。
4. %installセクション	仮のroot'/'にインストールします。
5. %files セクション	インストール構造を示します。
6. %changelogセクション	更新履歴を記載する

メタ情報セクション

```
%include Solaris.inc

Name:           nano
Summary:        GNU nano text editor
Version:        2.0.9
License:        GPLv2
Url:            http://www.nano-editor.org
Source:         http://www.nano-editor.org/dist/v2.0/%{name}-%{version}.tar.gz
Group:          Editor
Distribution:   OpenSolaris
Vendor:         OpenSolaris Community
BuildRoot:      %{_tmppath}/%{name}-%{version}-build
SUNW_Basedir:   %{_basedir}
SUNW_Copyright: %{name}.copyright
%include default-depend.inc

# OpenSolaris IPS Manifest Fields
Meta(info.upstream):            Chris Allegretta
Meta(info.maintainer):          Peter Jones
Meta(info.repository_url):      svn://svn.sv.gnu.org/nano/trunk/nano/
Meta(info.classification):      Editor

%description
GNU nano is an effort to provide a Pico-like editor, but also includes some features that
were missing in the original, such as 'search and replace', 'goto line' or internationalization
support.
```

パッケージの名前
サマリー
バージョン名
ライセンス名（意味コードではない）
WEBサイト
ファイル配布URL
グループ
参照）
http://opensolaris.org/os/community/sw-porters/contributing/ipsclass/

sourcejuicerでは、まずライセンスチェックされます。英語サイトがない場合は、日本語のURLを書き、翻訳してあげると良いです。

ソースコード・メンテナー
パッケージ・メンテナー
レポジトリのURL
IPS Class（Groupと一緒）

説明文

%prep/%setup セクション

```
%prep
rm -rf %name-%version
%setup -q -n nano-%version
```

このセクションは、tar ballからディレクトリへ展開するセクションです。

%setupは、疑似命令のようなもので、ワークディレクトリ作成、tarで展開、cdまでをします。

%build セクション

```
%build
export CFLAGS="%optflags"
export LDFLAGS="%{_ldflags}"
./configure --prefix=%{_prefix}                  \
            --bindir=%{_bindir}                     \
            --mandir=%{_mandir}                     \
            --infodir=%{_infodir}                   \
            --sysconfdir=%{_sysconfdir}             \
            --enable-all
make
```

configure && makeを行うセクションです。

ビルドのための変数を設定したり、configure前後の処理、make処理などを行います。

実際の例では、この直前にはpatchが入ることもあります。

%install セクション

```
%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT
rm -f $RPM_BUILD_ROOT%{_infodir}/dir
```

インストールコマンドを記載します。make installを行います。

最近のconfigure && makeの仕組みでは「DESTDIR=$RPM_BUILD_ROOT」を設定して置くことで、指定したWORKディレクトリにインストールを行います。

パッケージャはそのディレクトリを起点に、パッケージを作成します。

%files セクション

```
%files
%defattr (-, root, bin)
%dir %attr (0755, root, bin) %{_bindir}
%{_bindir}/*
%{_infodir}/*
%dir %attr(0755, root, sys) %{_datadir}
%dir %attr(0755, root, bin) %{_mandir}
%dir %attr(0755, root, bin) %{_mandir}/*
%{_mandir}/*/*
%dir %attr(0755, root, bin) %{_basedir}/share/nano
%{_basedir}/share/nano/*
%dir %attr(0755, root, bin) %{_basedir}/share/locale
%{_basedir}/share/locale/*
```

ここも重要なセクションで、実際のディレクトリのインストール先の配置を決めます。

meta情報エリアにあるbasedirよりも上にはインストールはできません。

ここに、書かれていないファイルが実際にあった場合、取りこぼしとしてエラーを出力します。

一方、ここに書かれているのにファイルがなかった場合、やはりエラーを出します。

%changelog

```
%changelog
* Thu Oct 23 2008 - andras _dot_ barna _at_ gmail _dot_ com
- new version, add --enable-all, add SFEncursesw for utf-8
* Wed Jul  5 2006 - laca _at_ sun _dot_ com
- delete -share subpkg
- update file attributes
* Fri Feb  3 2006 - mike kiedrowski (lakeside-AT-cybrzn-DOT-com)
- Initial version
```

変更履歴です。

ファイルの準備

specファイルの準備

specファイルをダウンロードします。

```
wget http://jucr.opensolaris.org/help/nano.spec
```

そのまま使えるのではなく、ファイルは<pre>～</pre>の間にあるので、切り出します。%filesセクションの追加項目も追加します。

copyrightファイルの削除

specファイルから自動的に、tar.gzを取得しますが、tar.gzからcopyrightを取り出す必要があります。そこで、先に取得しておきます。

```
http://www.nano-editor.org/dist/v2.0/nano-2.0.9.tar.gz
gtar zxf nano-2.0.9.tar.gz
cp nano-2.0.9/COPYING nano.copyright
```

※今回は不要ですが、初めてビルドするプログラムの場合は、ここで事前にビルド試験をしたりします。

patch、その他のファイル

必要なら用意しますが、今回は不要です。

ビルド作業

ビルドは一般ユーザで行います。

```
% pkgtool build-only --sourcedirs=`pwd` --ips --download nano.spec

INFO: IPS packages will be installed by default from http://localhost:80/
INFO: Copying %use'd or %include'd spec files to SPECS directory
INFO: Processing spec files
INFO: Finding sources
INFO: Running pkgbuild -ba [...] nano.spec (nano)
```

このあたりでしばらく止まります。

```
INFO: nano PASSED

Summary:

                          package |      status | details
--------------------------------+-------------+--------------------------------
                             nano |      PASSED |
```

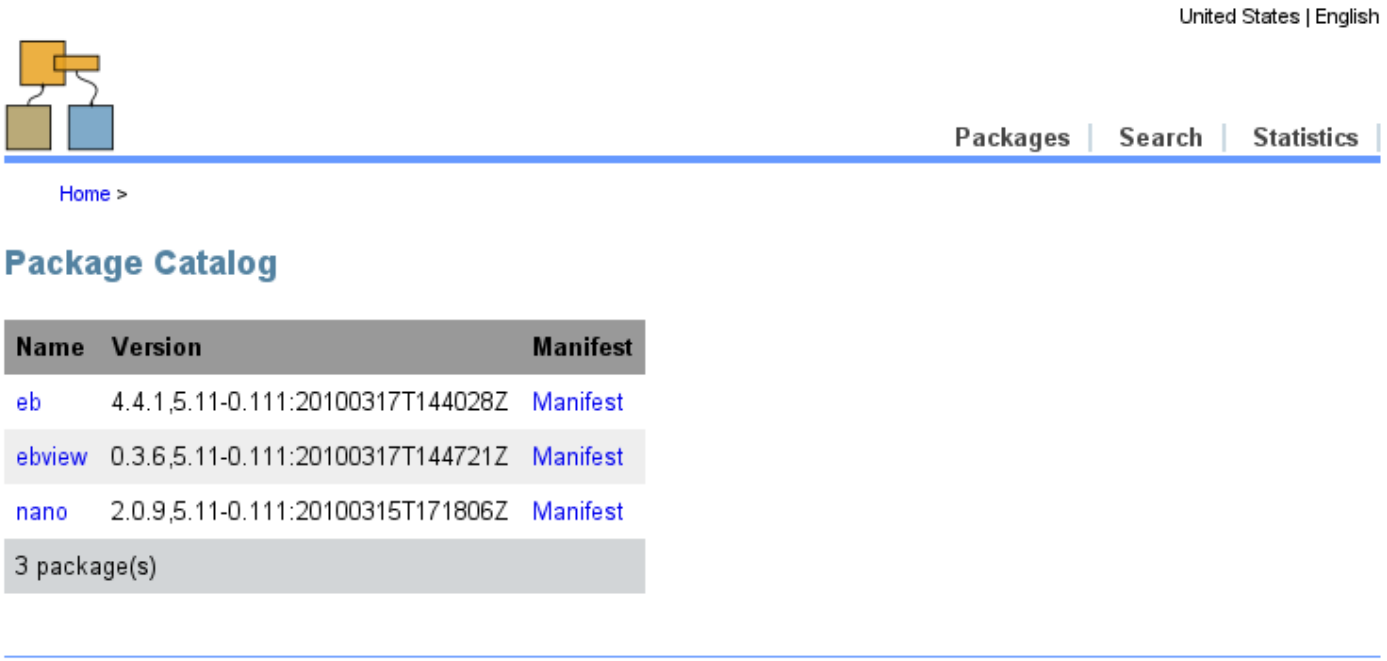
エラーがでたら、/tmp/nano.logを参考にします。

これで、IPSへの登録成功です!

確認作業（ブラウザから）

前ページで「PASS」であれば、すでに、work-specへの登録はされています。

ブラウザでは、一覧を見ることはできます。



確認作業（pkg コマンド）

テストサーバから、pkgコマンドで一覧がとれるように確認します。

念のため、publisher登録されているか確認します。

```
% pkg publisher -a
発行元                                  タイプ  状態    URI
opensolaris.org          (優先)      起点    online   http://pkg.opensolaris.org/dev/
contrib                              起点    online   http://pkg.opensolaris.org/contrib/
mypkgs                   (無効)      起点    online   http://192.168.199.61/
```

上記のように無効になっている場合は、enableにします。

```
% pkg set-publisher --enable mypkgs
```

pkgコマンドは、ローカルにキャッシュがあるため、それを更新します。

```
% pkg refresh --full mypkgs
```

一覧の取得

```
% pkg list -a 'pkg://mypkgs/*'
NAME (PUBLISHER)                                VERSION         STATE      UFIX
nano (mypkgs)                                   2.0.9-0.111     known      ----
```

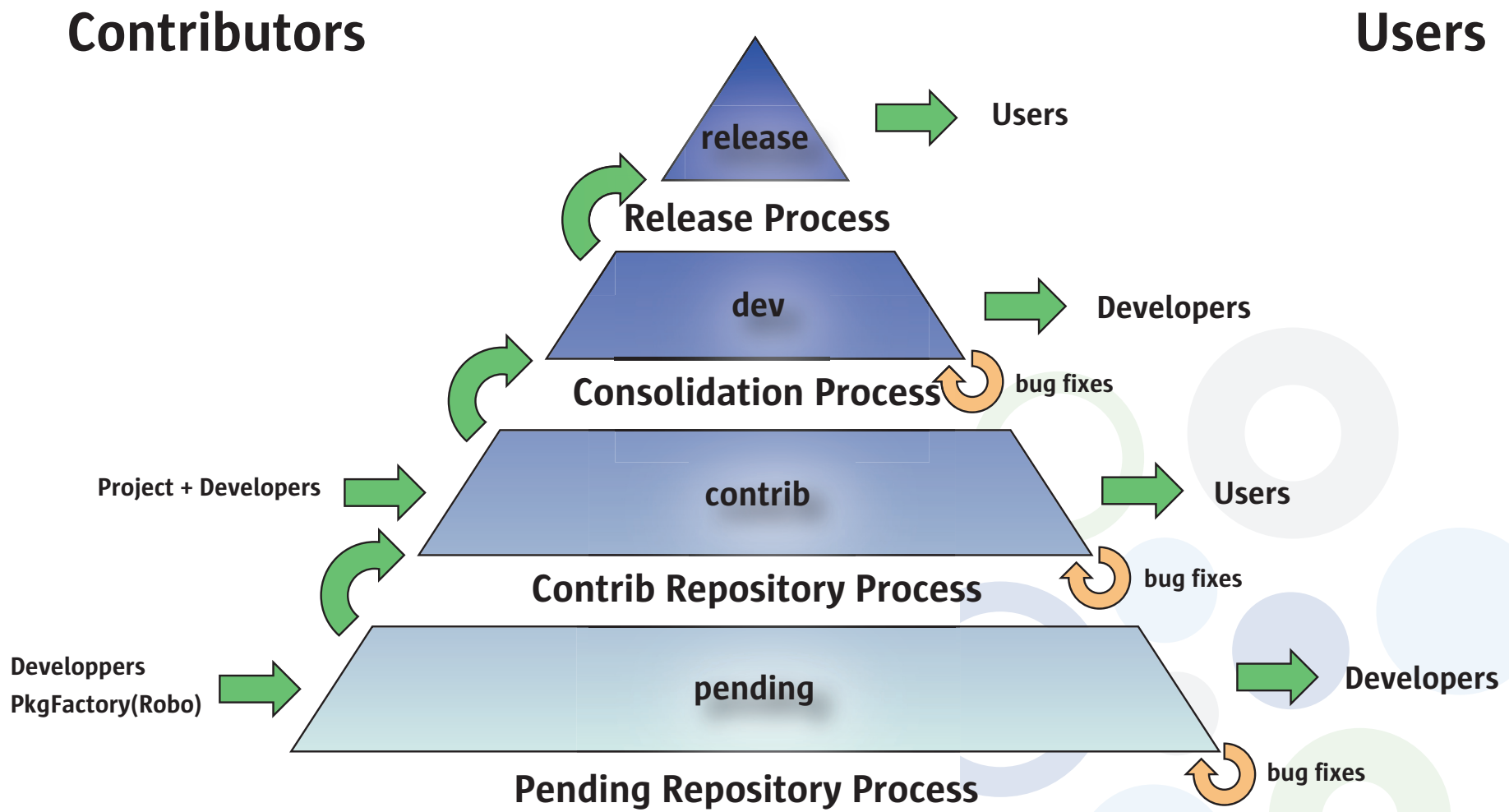
後は普通にインストールしてみたり、contentsコマンドで内容を確認してみてください。

これで、自分でパッケージを作れるようになりました。

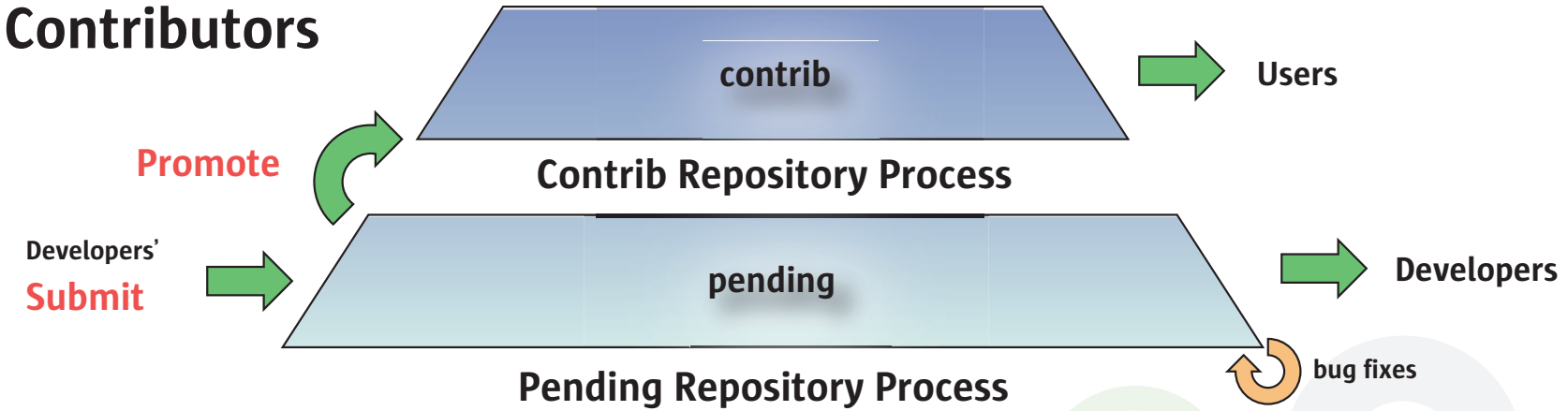
Contributor になろう！

Souce JuicerへのContribute!

opensolaris.org レポジトリの確認



pending → contrib の関係



opensolaris.orgのレポジトリのヒエラルキー中で、我々ができる部分は上記の箇所です。

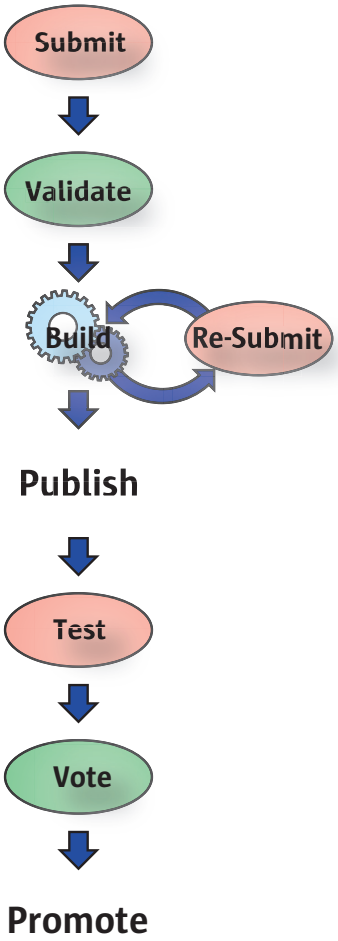
- pendingへ、specファイルなどの提供(Submit)
- contribへ、採用される(Promote)

■ 旧pendingについて

過去にあった、http://pkg.opensolaris.org/pendingは、2010.03に廃止されました。旧pendingのspecファイルは下記にあります。有用だと思うものがあれば、あなたがpkgメンテナーになって、是非、contributeしてください!!!

```
http://src.opensolaris.org/source/xref/pkgfactory/spec-archive-2008-11/
```

contrib レポジトリまでの道



次のような手順を経て、contribに公開されます。

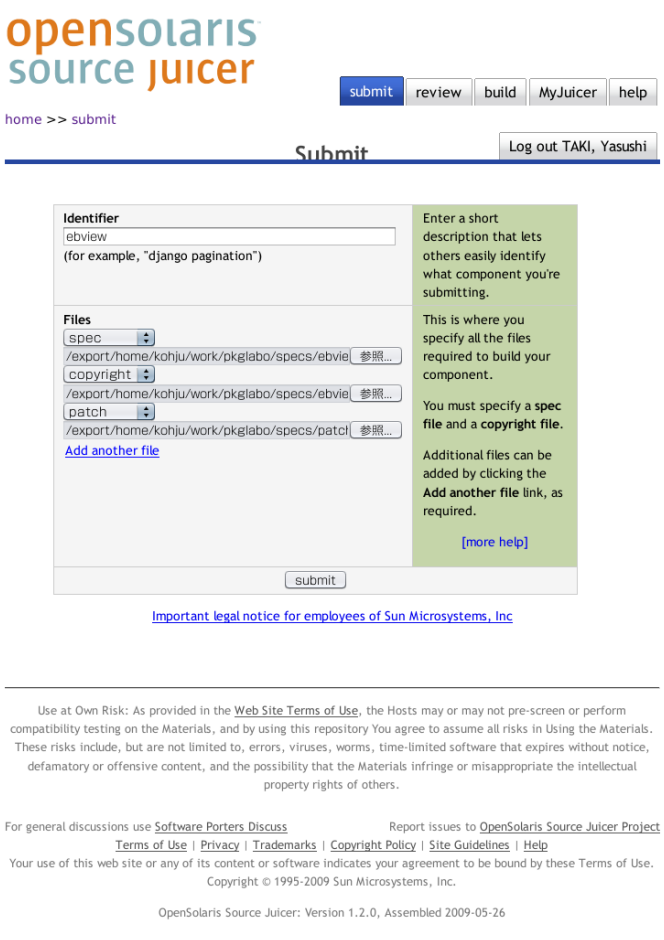
- Submit(Contributor):spec/copyright/patch等をsourcejuicerにアップロードします。
- Validate(Reviewer):copyrightファイル、specのcopyright行、配布元サイトのライセンス記載、アーカイブのlicenseなどを確認します。
- Build: Build Grid内でビルドされます。Zoneがその都度作られているようです。
- Publish：pendingレポジトリに公開されること。
- Test(Contributor):pendingレポジトリから、実際にインストールしてテストします。
- Vote(Approvers):そのソフトをPromoteするべきかの投票を行う。
- promote: contribレポジトリに公開されること。

※括弧内は役割を持った人のこと。

http://hub.opensolaris.org/bin/view/Community+Group+sw-porters/reporoles

Source Juicer への Submit

ログイン画面



spec、copyrightのアップロード

My Juicer : My Submissions

opensolaris source juicer

submit | review | build | MyJuicer | help

home >> myjucr

Log out TAKI, Yasushi

My Submissions

Identifier	Submitter	Date	Files	Validated?	Comments	Votes	
ebview	TAKI, Yasushi	2009-05-10	ebview.copyright ebview-01-kohju.diff ebview-02-kohju.diff ebview-03-kohju.diff ebview-04-kohju.diff ebview.spec	✓	7	+0	-0
xz	TAKI, Yasushi	2009-05-06	xz.spec xz.copyright	✓	4	+0	-0
eb	TAKI, Yasushi	2009-05-06	eb.copyright eb.spec	✓	11	+0	-0
lv	TAKI, Yasushi	2009-05-05	lv.spec lv.copyright lv-01-kohju.diff	✓	1	+0	-0
tree	TAKI, Yasushi	2009-05-02	tree.spec tree.copyright tree-01-kohju.diff	✗	5	+0	-0

自分でsubmitしたものです。

Validateが通るとチェックが入ります。

My Juicer : My Comments

掲示板機能です。

My Comments (Submissions I Have Commented On)

Identifier	Submitter	Date	Files	Validated?	Comments	Votes	
ebview	TAKI, Yasushi	2009-05-10	ebview.copyright ebview-01-kohju.diff ebview-02-kohju.diff ebview-03-kohju.diff ebview-04-kohju.diff ebview.spec	✓	7	+0	-0
eb	TAKI, Yasushi	2009-05-06	eb.copyright eb.spec	✓	11	+0	-0
xz	TAKI, Yasushi	2009-05-06	xz.spec xz.copyright	✓	4	+0	-0
lv	TAKI, Yasushi	2009-05-05	lv.spec lv.copyright lv-01-kohju.diff	✓	1	+0	-0
tree	TAKI, Yasushi	2009-05-02	tree.spec tree.copyright tree-01-kohju.diff	✗	5	+0	-0

My Juicer：My Build

My Builds

JobID	Identifier	Submitter	Build Start	Build Finish	Install	Build Log	Status
1323	lv	TAKI, Yasushi	May 05 22:05	May 05 22:05	Install	Log	PASSED
1390	xz	TAKI, Yasushi	May 09 13:05	May 09 13:05	Install	Log	PASSED
1391	eb	TAKI, Yasushi	May 09 13:05	May 09 13:05		Log	FAILED
1403	ebview	TAKI, Yasushi	May 11 04:05	May 11 04:05		Log	FAILED

Statusがバグっているのか、表示方法にコツがいるのか、最初のSubmitのものしか出ていないように見えます。

詳しくはBuildのタブから見た方がよいです。

現在の Source Jucer の環境 #1

現在の環境変数です。

```
RPM_OPT_FLAGS=-i -xO4 -xspace -xstrconst
  -xpentium -mr -xregs=no%frameptr
SHELL=/bin/bash
RPM_PACKAGE_RELEASE=0
RPM_PACKAGE_NAME=eb
RPM_SOURCE_DIR=/packages/SOURCES
CXX32=/opt/SunStudioExpress/bin/CC
A__z=” *SHLVL
CC64=/opt/SunStudioExpress/bin/cc
MAIL=/var/mail/bld
PATH=/opt/jdsbld/bin:/usr/ccs/bin:/usr/
  gnu/bin:/usr/bin:/usr/sbin:/bin:/usr/
  sfw/bin:/opt/SunStudioExpress/bin
LD=/opt/jdsbld/bin/ld-wrapper
PWD=/packages/BUILD/eb-4.4.1
RPM_BUILD_ROOT=/var/tmp/pkgbuild-bld/
  eb-4.4.1-build
RPM_PACKAGE_VERSION=4.4.1
RPM_OS=solaris
CXX=/opt/SunStudioExpress/bin/CC
RPM_DOC_DIR=/usr/share/doc/packages/eb
HOME=/export/home/bld
SHLVL=2
CXX64=/opt/SunStudioExpress/bin/CC
LOGNAME=bld
CC32=/opt/SunStudioExpress/bin/cc
RPM_BUILD_DIR=/packages/BUILD
RPM_ARCH=i386
CC=/opt/SunStudioExpress/bin/cc
_=/usr/bin/env
OLDPWD=/packages/BUILD
```

現在の Source Jucer の環境 #2

CPU情報とOSのバージョンです。

```
% isainfo -v

64-bit amd64 applications
  ssse3 cx16 mon sse3 sse2 sse fxsr mmx cmov amd_sysc cx8 tsc fpu
32-bit i386 applications
  ssse3 ahf cx16 mon sse3 sse2 sse fxsr mmx cmov sep cx8 tsc fpu

% uname -a
SunOS zone090601 5.11 snv_111b i86pc i386 i86pc
```

現在の Source Jucer の環境 #3

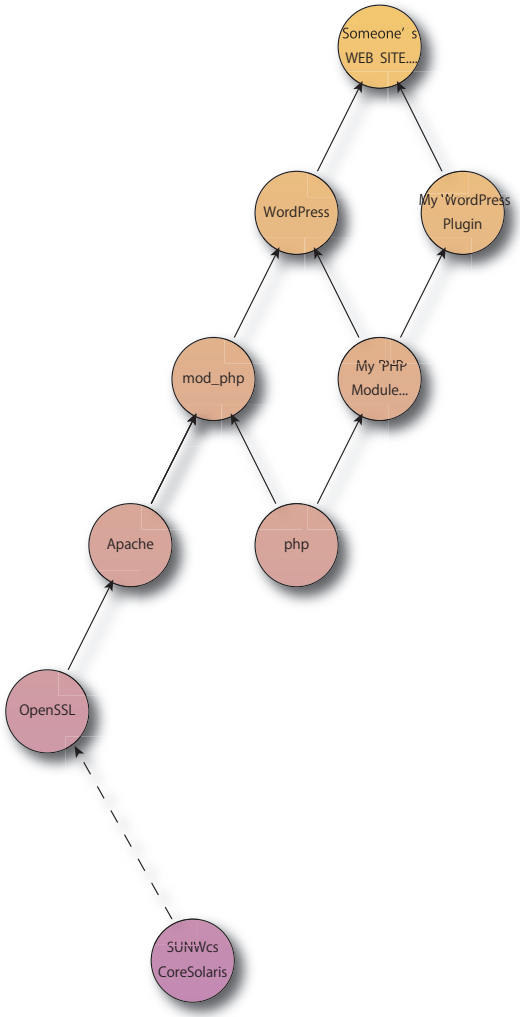
下記は、デフォルトで入っているパッケージです。

SUNWPython
SUNWTcl
SUNWTk
SUNWadmap
SUNWadmlib-sysid
SUNWadmr
SUNWarc
SUNWatfs
SUNWbash
SUNWbip
SUNWbzip
SUNWckr
SUNWcnetr
SUNWcpcu
SUNWcpp
SUNWcs
SUNWcsd
SUNWcsl
SUNWdoc
SUNWdtrc
SUNWesu
SUNWgpch
SUNWgss
SUNWgtar
SUNWgzip
SUNWhea
SUNWinstall-libs
SUNWipkg
SUNWj6cfg
SUNWj6dev
SUNWj6dmo
SUNWj6dmx
SUNWj6dvx
SUNWj6man
SUNWj6rt
SUNWj6rtx
SUNWkrb
SUNWlang-common
SUNWlang-en
SUNWlang-enUS
SUNWless
SUNWlexpt
SUNWlibC
SUNWlibm
SUNWlibms
SUNWlibsasl
SUNWlldap
SUNWloc
SUNWlxml
SUNWmd
SUNWnfsc
SUNWnis
SUNWopenssl
SUNWperl584core
SUNWpicl
SUNWpkgcmds
SUNWpool
SUNWpr
SUNWpython-cherrypy
SUNWpython-mako
SUNWpython-ply
SUNWpython-pyopenssl
SUNWpython24-simplejson
SUNWrcmdc
SUNWsfwhea
SUNWsmapi
SUNWsprot
SUNWssh
SUNWsshcu
SUNWsshd
SUNWtecla
SUNWtls
SUNWtoo
SUNWvim
SUNWwbsup
SUNWwget
SUNWwsr2
SUNWxwrtl
SUNWzfs
SUNWzone
entire
sunstudioexpress

Appendix:
より実践的なポーティング

実際にソフトウェアのポーティングでつまずくこと

パッケージの依存関係を意識する



プログラムをポーティングするには様々な依存があります。

先ほどのサンプルのように、ほとんど依存が無いのはむしろ希です。

大抵のプログラムは何かのプログラムに依存し、そしてそのプログラムが何かから依存関係にあるのはよくある話です。

この依存関係をきちんと書かなくては、IPSで一発インストールができません。

依存には、実行依存と、ビルド依存があります。

ビルド依存はビルドの時だけ利用するもので、実行依存は実行のために必要な依存となります。

依存のキーはパッケージ名となります。

そのためパッケージ名は非常に大きな意味を持ちます。

選択の自由があるということ

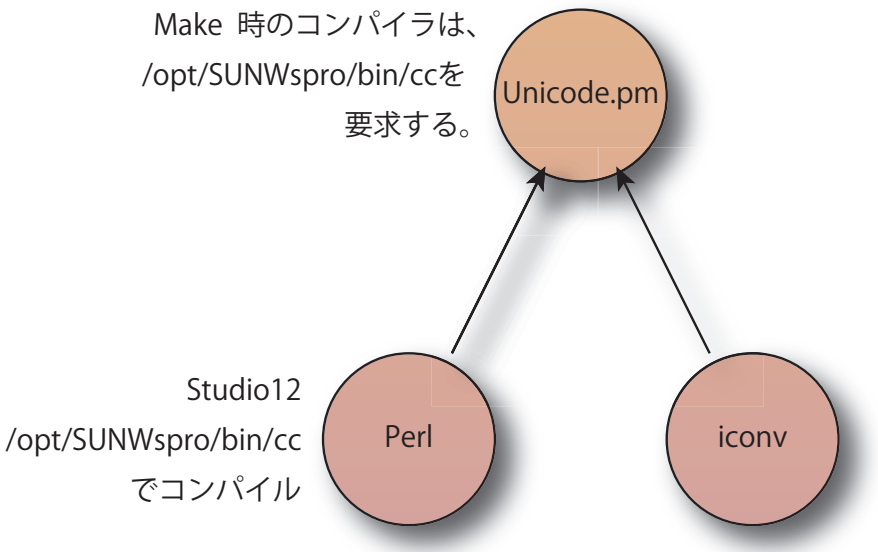
OpenSolarisには、「**選択の自由**」があります。

- コンパイラが複数あります。
  - Studio12、gccなど
- 同じ目的のプログラムが複数インストールできます。
  - SUNの実装と、GNUの実装・・・
- 同じプログラムの違うバージョンが複数インストールできます。
  - PostgreSQL 8.1/8.2/8.3・・・・
  - バイナリコンパチのための古いsoライブラリなど

これらを、どのように選ぶか?を、ポーターは考えないとなりません。

- 選択の方法
  - 環境変数の変更、PATHの変更（ほとんどはこれで可能）
  - 簡単なパッチの作成、ラッピングシェルスクリプトの作成等々。

自由に対する選択



次のように選択をします

- 環境変数を工夫する
- PATH系、その他の環境変数
- configureやMAKEの引数
- コンパイラに依存するもの
  - コンパイラ、コンパイラオプションを指定するものがあります。
  - gccでないとコンパイルできないものもあります。
- ライブラリ、ヘッダに依存するもの
  - 無いものは先にpendingへ登録。
  - contribにあるものとは依存関係が作れます。
- CPUに依存するもの

実践的な spec ファイル #1

```
#
# spec file for package eb
#
# This file and all modifications and additions to the pristine
# package are under the same license as the package itself.
#
#
%include Solaris.inc
```

~/package/SPECS/Solaris.incです。

ここで、%{_basedir}などのディレクトリや、コンパイルオプションなど、様々な変数が定義されます。Solaris.inc、prod.inc、options.incに目を通しておくと良いでしょう。

```
%define _prefix /usr
%define tarball_version  4.4.1

Name:                    eb
Summary:                 the library for accessing to the EPWING format Dictionaries
Version:                 4.4.1
License:                 Modified BSDL
Url:                     http://www.sra.co.jp/people/m-kasahr/eb/
Source:                     ftp://ftp.sra.co.jp/pub/misc/%{name}/%{name}-%{tarball_
  version}.tar.bz2
```

実践的な spec ファイル #2

```
Distribution:           OpenSolaris
Vendor:                 OpenSolaris Community
#SUNW_Basedir:          %{_basedir}
SUNW_Basedir:           /
```

Solaris10の頃は、1つのパッケージが、/usr（たとえばSUNWapch22u）と、/（たとえばSUNWapch22r）に割れていました。これは、/usrが共有されていたり、SPARSE　ZONE（/usrがglobal zoneからinheritanceしている）を作る時などに必要でした。

specから、SVr4のパッケージを作るときには、サブパッケージとして割る方がベターですが、OpenSolaris用に作るときには割りません。

```
SUNW_Copyright:         %{name}.copyright
BuildRoot:              %{_tmppath}/%{name}-%{version}-build

BuildRequires: SUNWbtool
BuildRequires: SUNWbinutils
BuildRequires: SUNWxcu4
BuildRequires: SUNWgmake
```

これらは、configureからのビルドで、大抵必要になるツール達です。

なるべく、GNUものではなく、Solarisの純正を使うように設定しています。

また、ここではこれだけしか使いませんでしたが、他にも必要な事があります。

実践的な spec ファイル #3

```
BuildRequires: SUNWzlib
BuildRequires: SUNWgnu-gettext
```

この2つは、このライブラリが内部的に使っているライブラリです。

```
Requires: SUNWzlib
Requires: SUNWgnu-gettext
```

「Requires:」に記載したパッケージも、その環境に「インストールされている」必要があります。自分のCBEでは、インストールされていなければ、依存の記述が「無視」されます。

その結果、自分のCBEは使い込むと「てんこもりのインストール環境」になります。

SourceJuicerはビルドの度にプレーンなビルド環境をZoneとして作るので、**自分のCBEではビルドできても、Juicerだとビルドできない**ことがおきるのです! 大注意!

```
#%include default-depend.inc
```

default-depend.incを入れると、CoreSolarisパッケージ群に依存します。

なるべく入れておくべきなのかもしれませんが、CoreSolarisに依存させると、zone内にパッケージを入れたいときにGlobalZoneのバージョンに依存することになります。

議論があるところかもしれません。

```
# OpenSolaris IPS Package Manifest Fields
Meta(info.maintainer):      pkglabo.justplayer.com <pkgadmin@justplayer.com>
Meta(info.upstream):        Motoyuki Kasahara <m-kasahr@sra.co.jp>
```

実践的な spec ファイル #4

```
# Meta(info.repository_url):     [open source code repository]
Meta(info.classification):   System Libraries

%description
EB library is for accessing to the EPWING format Dictionaries

%prep
%setup -c -n %name-%version
%ifarch amd64 sparcv9
rm -rf %{name}-%{tarball_version}-64
cp -rp %{name}-%{tarball_version} %{name}-%{tarball_version}-64
%endif
```

ここでは、64bit版も展開しています。

1. %setup疑似命令で、eb-4.4.1というディレクトリに展開。
2. eb-4.4.1をeb-4.4.1-64にコピー

この結果、32bit用（eb-4.4.1）と64bit用（eb-4.4.1-64）のワークディレクトリができます。

このようなオペレーションで、カレントディレクトリがわからなくなったら''pwd''を入れておくと良いでしょう。

実践的な spec ファイル #5

```
%build

CPUS=`/usr/sbin/psrinfo | grep on-line | wc -l | tr -d ' '`
if test "x$CPUS" = "x" -o $CPUS = 0; then
    CPUS=1
fi
```

CPUの個数を調べています。「gmake -j 同時実行数」の数字を求めています。

参考までに、私が出会ったSource　Juicerの環境は、ログを見る限りではCPUが8個あるようです。SourceJuicerはBuild　Gridの名前の通り、毎度違うマシンに、そのパッケージ専用のZoneが作られていますが、高速なマシンのようなので、こうしておくと速くなって嬉しいということです。

ただし、gmakeのMakefile(GNUmakefile)を、きちんと書いておかないと、並列処理時にエラーが出るアプリもあるため注意。自分のマシンがCPU 1つだと、自分のCBEではビルドが成功するのに、Source Juicerでは失敗する可能性がでてきます。

```
cd %{name}-%{tarball_version}
%ifarch sparc
%define target sparc-sun-solaris
%else
%define target i386-sun-solaris
%endif
```

実践的な spec ファイル #6

```
./configure \
 --prefix=%{_prefix}\
 --sysconfdir=%{_sysconfdir} \
 --libdir=%{_libdir} \
 --bindir=%{_bindir} \
 --includedir=%{_includedir} \
 --mandir=%{_mandir}
gmake -j$CPUS
```

ディレクトリは上記のようなものが大体デフォルトです。

configureは、自動的にいろいろ探すという仕様ですが、必要なものは、enableもdisableもwithもwithoutも、なるべく「**全部書く**」ほうがよいでしょう。

書いておけば、書いたものと違えば、ログからわかります。

「自分のCBEはてんこもり」なのに、「Source　Juicerはなにもない」ため、その結果、CBEでは問題なく動いても、Source　Juicerでは、コンパイルエラーがでたり、テストインストールの段階で、ある機能が「**動かない**」ことになります。

気がつかないとこの類のエラーはわかりにくいので注意。

また、CPU依存の最適化は、基本的には書かないようにしましょう。

実践的な spec ファイル #7

```
%ifarch amd64 sparcv9
cd ../%{name}-%{tarball_version}-64
export CFLAGS="%optflags64"
./configure \
 --prefix=%{_prefix}\
 --sysconfdir=%{_sysconfdir} \
 --libdir=%{_libdir}/%{_arch64} \
 --bindir=%{_bindir}/%{_arch64} \
 --includedir=%{_includedir} \
 --mandir=%{_mandir}
gmake -j$CPUS
%endif
```

64ビット版のものは、バイナリのインストールディレクトリが異なるだけのものが大半です。

見ての通り、ここでは変数を詰め直し、再びビルドしています。従って、64ビットにも対応させるためには、コンパイル時間だけでも2倍かかるのです。

実践的な spec ファイル #8

```
%install
cd %{name}-%{tarball_version}
gmake install DESTDIR=$RPM_BUILD_ROOT
```

インストール先は、直接root直下ではなく、DESTDIR=で、仮ROOTを指定する必要があります。たまにINSTALL_DIRのものもあります。

じつは、このテストをしていないソフトが、とてもたくさんあります。大抵、Makefile.inあたりのパッチを作ってお茶を濁すことが多いのですが、正しくは、aclocalからやり直すのが本来かもしれません。

じつは、このためのパッチを作ることが一番多いかもしれません。

```
%ifarch amd64 sparcv9
cd ../%{name}-%{tarball_version}-64
gmake install DESTDIR=$RPM_BUILD_ROOT
cd ..
%endif
```

64ビットのインストールは、いったん重ね書きをしています。

インストール対象の違いが、バイナリだけなので、ほとんどのものは、これで動作します。

もし、このパッケージ固有のファイル（SMFのMANIFESTやら、起動スクリプト、サンプルconfigファイルなど）がある場合は、ここで仮ROOTにコピーしておきます。

実践的な spec ファイル #9

```
%{?pkgbuild_postprocess:    %pkgbuild_postprocess    -v    -c    "%{version}:%{jds_
  version}:%{name}:$RPM_ARCH:%(date +%%Y-%%m-%%d):%{support_level}" $RPM_BUILD_ROOT}

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr (-, root, bin)
%dir %attr (0755, root, bin) %{_bindir}
%{_bindir}/*
%dir %attr(0755, root, bin) %{_libdir}
%{_libdir}/*
%dir %attr(0755, root, bin) %{_prefix}/include/eb
%{_prefix}/include/eb/*
%dir %attr(0755, root, bin) %{_prefix}/share
%{_prefix}/share/*
%{_sysconfdir}/eb.conf

%changelog
* Wed May  6 2009 TAKI, Yasushi <taki@justplayer.com>
- Initial Revision
* Mon Mar  8 2010 TAKI, Yasushi <taki@justplayer.com>
- delete sub packages
- support source juicer
```

パッチの書き方

パッチの作成

ディレクトリ構造

~/work/tmp/php-5.1.6	パッチ当て済みのディレクトリ
~/work/tmp/php-5.1.6.orig	パッチ当て前のディレクトリ

次のように作ります。

```
gdiff -urN php-5.1.6.orig php-5.1.6 > patches/php51-01-kohju.diff
```

※パッチは、目的毎に、1つづつ作りましょう。

specファイル内の指定

ソース指定箇所

```
Source:                    http://museum.php.net/%{tarball_name}5/%{tarball_name}-
  %{tarball_version}.tar.bz2
Source1:                   php51-php5.1.conf
Patch1:                    %{name}-01-kohju.diff
```

実際に当てる箇所

```
%setup -c -n %name-%version
%patch1 -p0
%build
```

gccでしかコンパイルできない！

ものによってはgccでしかビルドできないソフトもあります。

依存関係解決の場所で

```
BuildRequires: SUNWgnu-automake-19
BuildRequires: SUNWlibtool
BuildRequires: SUNWgcc
BuildRequires: SUNWgnu-automake-110
BuildRequires: SUNWgmake
BuildRequires: SUNWbison
BuildRequires: SUNWflexlex
BuildRequires: SUNWaconf
```

%buildでこのようなものを書きます。

```
export CC=gcc
export CFLAGS="%gcc_optflags"
export CXX=g++
export CXXFLAGS="%gcc_cxx_optflags"
export LDFLAGS="%_ldflags"
```

Pythonに絡む場合は、次のようなものをつけるときもあります。

```
export PYCC_CC=gcc
export PYCC_CXX=g++
```

configureで、次のようなオプションがあるものもあります

```
--without-gcc
```

postinstall script はどうするの？

IPSには、Post Install Scriptという概念がありません。

私感と想像ですが、次のようなことが考えられます。

- インストールでスクリプトが入ると、正直、何が起きるかわからない。
  - インストール、アンインストール、アップデートなど、様々な事に対応するのは、大変だし、パッケージ毎の対応のばらつきが起きる。
- パッケージャーで行うことは、ファイルのインストールにとどめた方がよい。
- その結果、ファイルのベリファイ、ファイルの出身パッケージなどが追いやすくなる!
- 定型的に行うこと(ドライバの追加、SMF manifestのインポート等)は、機能の棚卸しと正規化を行い、pkgコマンドのclassを追加するべきである。

どうしても必要な時はどうするの?

正しい方法かわかりませんが….

- zfs autosnapを見る限り、SMFの起動スクリプトに書いている。roleadd等。
- 特定のパッケージをenable/disableすることで、インストール後のパッケージの初期化スクリプトを走らせることができる。

ビルド失敗例 #1

BuildRequiredの依存ものが足りないとき、その1。

```
WARNING: skipping package eb: required package SUNWgnu-gettext not installed
WARNING: and no spec file specified on the command line provides it
INFO: Hint: use the --autodeps to locate spec files for dependencies automatically
Summary:

                          package |      status | details
--------------------------------+-------------+------------------------------
                               eb |  DEP_FAILED | Dependency check failed
```

/tmp/eb.logを見るとわかります。

```
INFO: Checking dependencies of eb
WARNING: skipping package eb: required package SUNWgnu-gettext not installed
WARNING: and no spec file specified on the command line provides it
INFO: Hint: use the --autodeps to locate spec files for dependencies automatically
WARNING: eb won't be built as it requires SUNWgnu-gettext
```

このエラーは、BuildRequiredにあるのに、ビルド環境で無いときに出てきます。従って、juicerではでてきません。

対象方法は、SUNWgnu-gettextをインストールしましょう。ただし、SUNWgnu-gettextは、SVr4のパッケージ名です。

ビルド失敗例 #2

自分のCBEでうまくいって、SourceJuicerでうまくいかないとき。Juicerのログに次のものがありました。

```
pkgbuild: checking host system type... i386-pc-solaris2.11
pkgbuild: checking for a sed that does not truncate output... /usr/bin/sed
pkgbuild: checking for grep that handles long lines and -e... configure: error: no
  acceptable grep could be found in /opt/jdsbld/bin:/usr/ccs/bin:/usr/gnu/bin:/usr/
  bin:/usr/sbin:/bin:/usr/sfw/bin:/opt/SunStudioExpress/bin:/usr/xpg4/bin
pkgbuild: Bad exit status from /var/tmp/pkgbuild-bld/pkgbuild-tmp-2.9172 (%build)
```

configure中に、grepのオプションが足りないようです。ローカルCBEでコンパイルに成功した時の例を見ると…

```
pkgbuild: checking for a sed that does not truncate output... /opt/dtbld/bin/sed
pkgbuild: checking for grep that handles long lines and -e... /usr/xpg4/bin/grep
pkgbuild: checking for egrep... /usr/xpg4/bin/grep -E
pkgbuild: checking for fgrep... /usr/xpg4/bin/grep -F
pkgbuild: checking for non-GNU ld... /usr/ccs/bin/ld
```

grepが、/usr/xpg4/bin/grepを使っているようです。

```
pkg search /usr/xpg4/bin/grep
```

で検索すると、SUNWxcu4のようす。この名前はIPS名なので、

```
pkg contents -m SUNWxcu4 | grep legacy
```

と行って、legacyパッケージ名を調べ、BuildRequireに登録します。

ビルド失敗例 #3

Juicerのログを見ると、何事もなくコンパイル終了していましたが、途中でコンパイルエラーがでていました。

エラー箇所は、linkerのエラーでしたが、configureのログを見ると次のような箇所がありました。

```
pkgbuild: checking for objdump... no
pkgbuild: checking how to recognize dependent libraries... pass_all
pkgbuild: checking for ar... no
pkgbuild: checking for strip... no
pkgbuild: checking for ranlib... no
pkgbuild: checking command to parse link -dump -symbols output from
/opt/SunStudioExpress/bin/cc object... failed
```

これらのファイルを、すべて前ページと同じように調べ、BuildRequireに登録する必要があります。

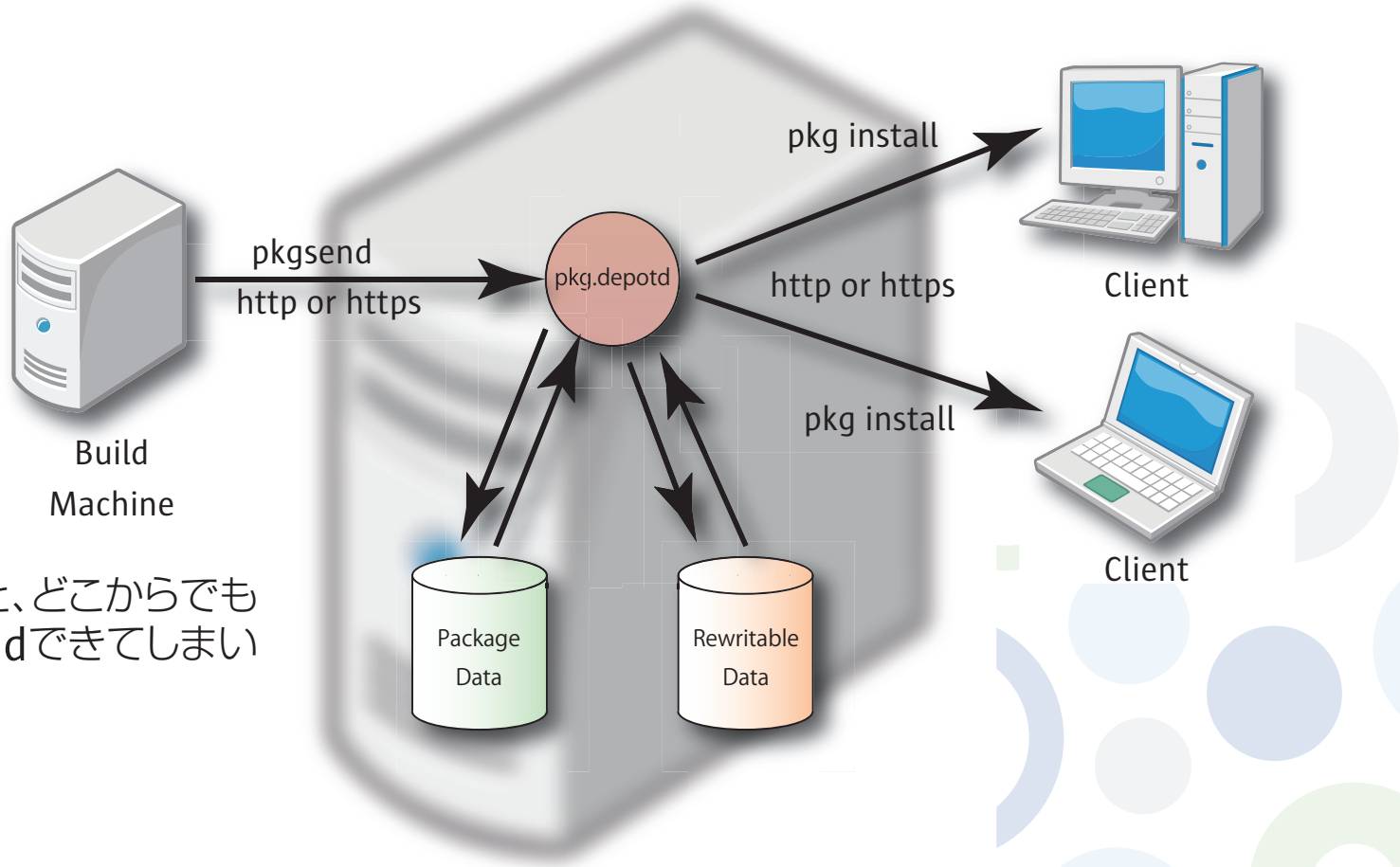
このように、てんこ盛りCBEと、スリムなSourceJuicerの違いで出てくるエラーは様々な見え方でエラーがでてきます。

公開用 IPS サーバを設定する

～独自のIPSサーバを作って公開する～

IPS サーバのモード設定 #1

デフォルトのpkg/serverの起動状態



これだと、どこからでもpkgsendできてしまいます。

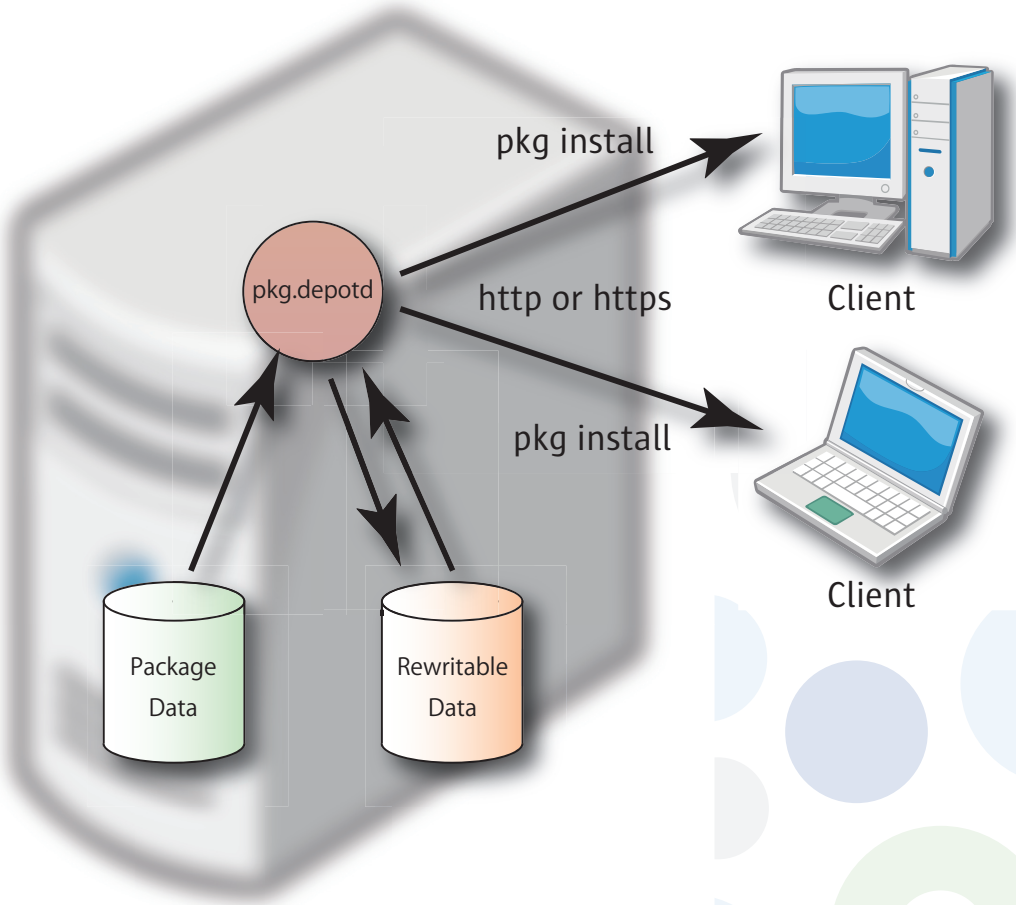
IPS サーバのモード設定 #2

ReadOnlyモードにすることで、pkgsendができないモードで立ち上がることができます。

公開しているサーバはこのように設定を行う必要があります。

また、

- Package Dataはリードオンリーとして読み込みをするため、リードオンリーファイルシステムとして用意することが可能。プロパティ名は、pkg/inst_root。
- Rewritable Dataは、メタ情報など逐次作成する一次データと考えられます。プロパティ名は、pkg/writable_root。



公開用 IPS サーバ

IPSサーバのオプションは、svccfgを利用して設定します。

次のオプションを設定するとよいでしょう（参考:man pkg.depotd）

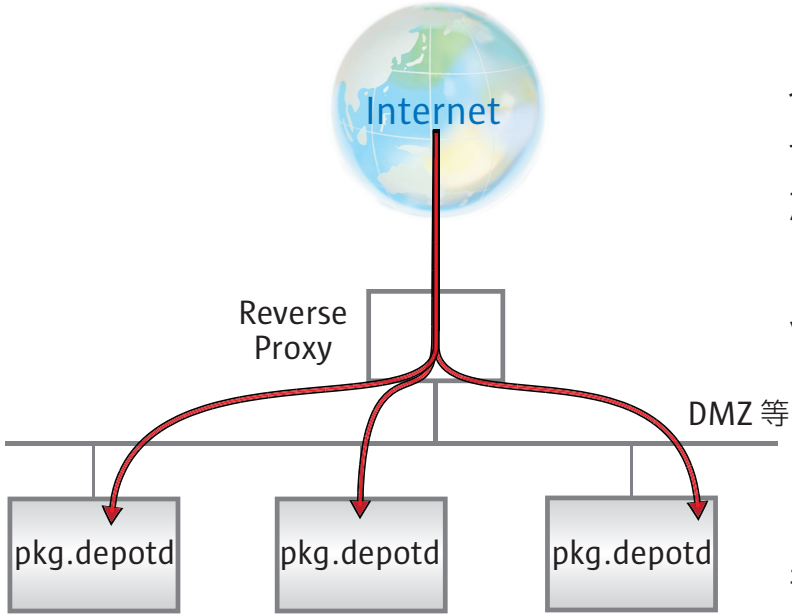
- `pkg/readonly` リードオンリー。trueにすると、pkgsendできなくなります。
- `pkg/content_root` ドキュメントルート。デザインを変更するときのコンポーネントはここに保存（IPSは、httpdとして起動している）。デフォルトは、/usr/share/lib/pkg
- `pkg/inst_root` レポジトリの実体が入ります。レポジトリデータはこの場所にあるので、ここをコピーすることにより、レポジトリの複製が可能です。デフォルトは/var/pkg/repo。
- `pkg/writable_root` リライタブルな一次データが入ります。デフォルトはinst_rootと同じ場所を利用します。カラのディレクトリを指定すると勝手に作り、inst_rootをread only filesystemに設置することが可能になります。
- `pkg/log_access` アクセスログ。
- `pkg/log_errors` エラーログ。
- `pkg/port` LISTENポート番号。
- `pkg/proxy_base` Proxyサーバを経由したとき、実際に参照されるURL。
- `pkg/threads` 受付スレッドの本数。デフォルトは10なので最大同時10人しかアクセスできません。

設定変更例

```
pfexec svccfg -s pkg/server "setprop pkg/port=10000"
pfexec svcadm refresh pkg/server
pfexec svcadm restart pkg/server
```

具体的な設計例

- pkg.depotdはpythonで書かれているサーバですが、これは「アプリケーションサーバ」として考えた方が良いものです。様々な機能は徐々についてきましたが、Internetに向けてサービスする場合は、Apache等をフロントエンドにおいた方が無難です。
- Apacheは、workerモードで動作させます。preforkよりも少ないリソースでセッションを裁くことが可能になるためです。
- Apacheで、第一階層ディレクトリを振り分けます(/devや/releaseなど)。
- 振り分け後はmod_proxyを利用し、下にあるpkg.depotdに接続することになります。



- pkg.depotdが1つのサーバで足りない場合、mod_proxy_balancerを利用します。複数台に分散させることで、落ちにくいサービス環境が可能となります。
- apacheのスレッド数と、pkg.depotdのスレッド数は適宜調整します。
- 一部のリクエストは、mod_mem_cache/mod_disk_cacheを利用し、高速化ができます。
- pkg/writable_rootをramdiskに置くこともできるでしょう。

設定例（zfs,pkg.depotd）

シンプルにApache 1台：pkg.depotd 1台の構成をつくりましょう。

レポジトリのエリアを別の委譲されたZFSにします。

```
zfs create rpool/export/pkglabo/dev
zfs set mountpoint=/var/pkg/repo rpool/export/pkglabo/dev
```

この中に何らかの方法でデータを転送します。

```
zfs set readonly=on rpool/export/pkglabo/dev
```

readonlyとwritableなエリアを分割します。

```
touch /var/pkg/repo.cfg
svccfg -s application/pkg/server setprop pkg/proxy_base = “http://pkglabo.justplayer.
  com/dev”
svccfg -s application/pkg/server setprop pkg/readonly = true
svccfg -s application/pkg/server setprop pkg/cfg_file = “/var/pkg/repo.cfg”
svccfg -s application/pkg/server setprop pkg/writable_root = “/var/pkg/wrepo”
svcadm refresh application/pkg/server
svcadm enable application/pkg/server
```

設定例（Apache）

リバースプロキシを使う方法

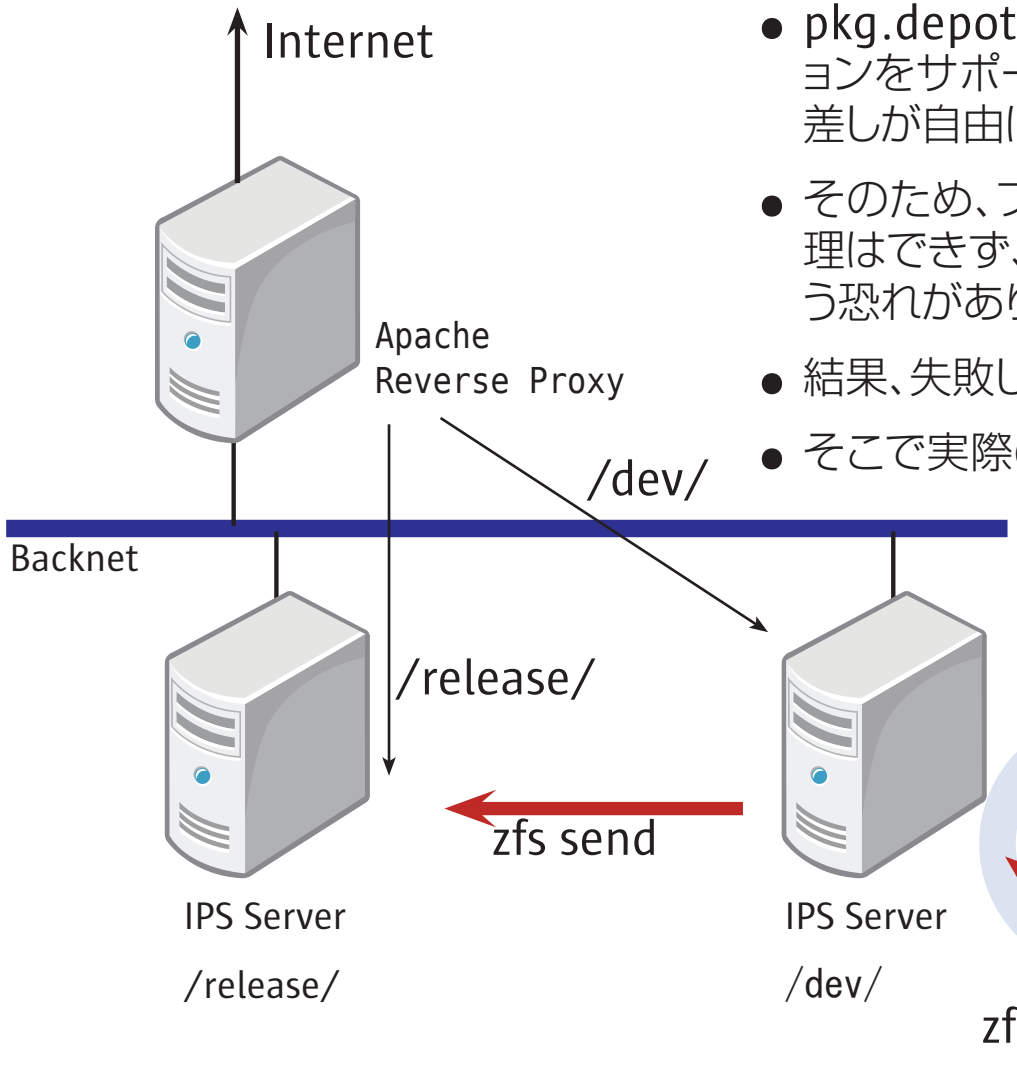
```
ProxyRequests Off
<Proxy *>
Order deny,allow
Allow from all
</Proxy>

ProxyPass /dev        http://inner-server/
ProxyPassReverse /dev http://inner-server/
```

Apacheのmod_rewrite使う方法です。

```
RewriteCond     %{REQUEST_URI}  ^/dev$ [NC]
RewriteRule     ^/dev /dev/ [R=301]
RewriteCond     %{REQUEST_URI}  ^/dev/ [NC]
RewriteRule     ^/dev/(.*) http://inner-server/$1 [P]
```

※複数サーバの場合は割愛。

パッケージデータの流れ



- pkg.depotdは、pkgsendによるトランザクションをサポートしていますが、ファイルの抜き差しが自由にできるわけではありません。
- そのため、ファイル・ディレクトリ構造による管理はできず、未完成でパッケージが増えてしまう恐れがあります。
- 結果、失敗しないpkgsendが求められます。
- そこで実際の開発に使ったIPSとは別に、プレーンなIPSを立てます。
- そのIPSサーバには間違いのないpkgsendを行う必要があります。
- つまり、IPS Serverのβサーバが必要なるのです。

パッケージ名の大変革

パッケージ名の大変革が起きました。

http://hub.opensolaris.org/bin/view/Project+indiana/Renamed+Packages+in+Build+133

下記はphp周りの例です。

OpenSolaris 2009.06		OpenSolaris 2010.03(verはsnv_134)	
SUNWphp52	5.2.9-0.111	web/php-52	5.2.12-0.134
SUNWphp52-mysql	5.2.9-0.111	web/php-52/extension/php-apc	3.0.19-0.134
SUNWphp52-pear	5.2.9-0.111	web/php-52/extension/php-idn	0.2.0-0.134
SUNWphp52-pgsql	5.2.9-0.111	web/php-52/extension/php-memcache	2.2.5-0.134
		web/php-52/extension/php-mysql	5.2.12-0.134
		web/php-52/extension/php-pear	5.2.12-0.134
		web/php-52/extension/php-pgsql	5.2.12-0.134
		web/php-52/extension/php-suhosin	0.9.29-0.134
		web/php-52/extension/php-tcpwrap	1.1.3-0.134
		web/php-52/extension/php-xdebug	2.0.5-0.134

- 名前が変わった
  - 他のOSから来た人が一目でわかりやすくなった。
- パッケージが細分化された
  - 細分化されると依存関係が作りやすい
  - サブパッケージ（1つのtar ballからできる複数のパッケージ）を作る必要がある。

これからも、pkg周りで追求することがありそうです。

ご静聴、ありがとうございました

続きはぜひ、OpenSolaris 勉強会で追求してください。

Japan OpenSolaris Users Group Leader
ジャストプレイヤー株式会社
瀧　康史