Gentoo Ebuilds 101

Arun Raghavan

Foss.in – November 30, 2005

Abstract

A basic introduction to Gentoo Linux' ebuild scripts, what they contain, how they work, and most importantly – how to write one.

Overview

- What this talk provides
  - ✔ An introduction to what ebuilds are
  - ✔ A quick guide to starting off on your own ebuilds, or modifying existing ones
- What it isn't
  - ✘ A detailed list of all the variables, function, eclasses[1] you can use
  - ✘ Going to tell you how to do the more esoteric things that you can
- The idea is to get you on your way to hacking ebuilds, not to bog you down with minutiae

[1] We'll just get to eclasses in a bit

- The information is not always complete if I’m trying to keep things simple

What is this ebuild you speak of?

- Basically just a `bash` script
- Set of commands to fetch, unpack, compile, and install a package
- . . . and then some ;-)
- File name looks like `<package-name>-<version>[-revision-number].ebuild`
- Organized by category under `/usr/portage/<category>/<package>`, like `media-video/mplayer` and `sys-kernel/gentoo-sources`
- Subdirectory `files` for additional files, like patches

- `metadata.xml` to list the maintainer and herd for this package
- `Manifest` – automatically created set of MD5 to verify all associated files

Peeping Under The Hood

What the little pixie gnomes do when you run `emerge ...`

1. Calculate the set of packages (specified and dependencies) to be installed

2. For each package:

   (a) Create `PORTAGE_TMPDIR/portage/full-package-name`, subdirectories `work`, `temp` and `image`
   (b) Fetch any files that require fetching
   (c) Check MD5 digests
   (d) Perform setup operations
   (e) Unpack the package
   (f) Compile the package

(g) Perform test steps if defined
(h) Install into image directory
(i) Merge from the image directory to your system

Ebuild Contents: The Header

- A header – only needs to be touched for new ebuilds
- Can be copied from `/usr/portage/header.txt`

```
# Copyright 1999-2005 Gentoo Foundation
# Distributed under the terms of the GNU General Public License v2
# $Header: $
```

Ebuild Contents: Predefined Variables

P, PN, PV, ...	Provide various combinations of the package name, version, revision, etc.
D	Path to install image of the package
T	Directory where the ebuild can create temporary files
WORKDIR	Path to the `work` directory where the package will be built – this is where you are when the ebuild script runs
FILESDIR	Path to the `files` subdirectory
DISTDIR	Path to the portage distfiles

Ebuild Contents: Required Variables

DESCRIPTION A short description of the package

SRC_URI URI from where the package files can be downloaded

HOMEPAGE Website of the package

KEYWORDS Specify the architectures the package has been tested on, and the degree of stability

SLOT Defines a “slot”. This allows multiple versions of a package to coexist (think GTK+ v1 and v2). Set to 0 if there are no slots

LICENSE — License under which the package is distributed – must be present in `/usr/portage/licenses`

IUSE — Specifies the set of USE flags that allow the build to be customized
These should be in
`/usr/portage/profiles/{use.desc|use.local.desc}`

Ebuild Contents: Optional Variables

S	Path to directory where the package is to be built (defaults to `${WORKDIR}/${P}`)
DEPEND	Set of packages on which this package depends for building – for example `>=x11-libs/gtk+-2`
RDEPEND	Set of packages on which this package has runtime dependencies
PROVIDE	For any “virtuals” provided by this package

Common Functions

use	`use foo` returns true if the `foo` USE-flag is set, false otherwise `use !foo` returns true if the `foo` USE-flag is not set, false otherwise
unpack	Useful function to unpack your gz/bz2/rar/zip/jar packages (just pass it the filename to unpack – picks it up from DISTDIR by default)
econf	Gentooized `./configure` You can pass arguments to this just like `./configure`
use_with	`use_with ogg` returns the string `--with-ogg` if the `ogg` USE-flag is set `use_with X x11` returns the string `--with-x11` is the `X` USE-flag is set
use_enable	Similar to `use_with`

emake	Runs `make` using the MAKEOPTS variable You can pass arguments/targets as with `make`
einstall	Gentooized `make install`
doins, dodoc, dobin	A bunch of helper functions for installing files, documentation, etc.
epatch	Use this to patch the source code before compiling (works with compressed patches too)[2]
einfo	Use this to display information messages
ewarn	. . . and this for warnings

[2] This is provided by `eutils.eclass`

die	Use this when you want to kill the script (like `emake || die`)

The Workhorse (Standard) Functions

- The ebuild script is basically a set of functions executed in a particular order

pkg_setup Do anything you have to do before starting (like add users/groups)

src_unpack Unpack the package, apply patches (this is run in WORKDIR)

src_compile Do that actual package compilation (this is run in S)

src_test Test the package

src_install Installs all files to D (basically a `make DESTDIR=${D} install`)

pkg_preinst Some pre-installation steps

pkg_postinst Some post-installation steps

- There are functions that can be run while unmerging a package as well – discovery of these is left as an exercise for the reader ;-)

The Sandbox

- Ebuilds runs in a protected “sandbox” environment
- You cannot write files above $WORKDIR
- This is done to make sure that bad ebuilds do not pose a threat to your system
- Does not apply to `pkg_*` functions, so you can touch the live system during setup, preinst, postinst
- This might happen because of bad build systems with hardcoded paths. If this is the case:
  - Try to get these fixed upstream (i.e. by the creators of the package)

  - While it is being fixed, write patches to fix the offending bits

- If there is a legitimate need to touch the live system, use `addread()`, `addpredict()`, and `addwrite()`
  - If you need `addwrite()`, something is probably wrong

Eclasses

- Eclasses enable code reuse in Gentoo
- Again, just a set of `bash` scripts in `/usr/portage/eclass`
- Provide:
  - Functions that perform common tasks, like modifying CFLAGS, adding users/groups, and much more
  - Provide standard ebuild functions (`src_unpack`, `src_compile`, etc.) for non-`autotools` packages (such as Python distutils and packages that need to be got from CVS/Subversion/Arch)
- To use them, call the `inherit` function (just like sourcing the script) immediately after the header . . .

```
inherit eutils flag-o-matic distutils
```

- The standard ebuild functions for the ebuild are taken from the last inherited eclass that provides them

Some Useful Eclasses

eutils	Lots of useful utilities, including `epatch`, `newuser`, `newgroup`
cvs	Used to buld packages straight out of CVS
distutils	Use this for Python packages packaged using distutils
flag-o-matic	Use this to modify user-defined CFLAGS/CXXFLAGS
games	For game ebuilds – controls file locations, ownership
rpm	For extracting packages distributed as RPMs
libtool	Commonly used to fix bad libtool files

toolchain-funcs Utilities to get information about the toolchain programs such as the compiler, linker, etc.

versionator Use this for all your version string manipulation

Building the Manifest (we're almost there)

- You need to build the `Manifest` file with the MD5 digests of all files involved
- This is done to make sure that none of the data used is corrupted or malicious
- All you need to do is . . .

```
ebuild path/to/file.ebuild digest
```

- This will process your ebuild, fetch required files, and then build the Manifest file

A Little Demo

- Some real-life ebuilds
  - `net-misc/logjam`
  - `net-misc/wget`
  - `net-analyzer/ethereal`
  - `sys-boot/grub`
- Let's write a little ebuild
  - `media-video/winki`
  - And digest it (sic)
- Is it a bird? Is it a plane? No it's `repoman`!

Testing

- Need to make sure everything went okay
- Run the actual program, make sure it works fine
- Make sure USE flags cover everything, and do not conflict
- Make sure depends are complete – watch out for circular depends
- Make sure documentation gets installed

Version Bumping

- What happens when a package releases the next version?
- If there isn't any major change in functionality (like new items for USE-flags, or new dependencies), it might be sufficient to just `cp package-oldversion.ebuild package-newversion.ebuild`
- Assuming versions are not hardcoded, doing an `ebuild ... digest` should be sufficient after this
- For major changes, you need to figure out what the changes implt to IUSE, DEPENDS, REDEPENDS and work accordingly

Assorted Tips

- Check bugs.gentoo.org for prior work
- To understand the USE-flag and depends you're going to need, look at the package README, configure.ac, etc.
- Make sure that the package doesn't use it's own compiler/CFLAGS – if it does, replace using `toolchain-funcs.eclass`
- Don't `die` easily – think about the user who started `emerge world` overnight, and comes back to see that your script terminated it half way
- Run `repoman` for QA checks

- Eclasses are neat
  - If you need to do anything strange, see if it's been done already and put in an eclass
  - Eclass man pages are in the `portage-manpages` package
  - You can always call an eclass' standard function like `distutils_src_compile`

References

- The `portage-manpages`
- `man 5 ebuild`
- These are well written files, that are not too difficult to understand
  - `/usr/lib/portage/pym/portage.py`
  - `/usr/lib/portage/bin/ebuild.sh`
  - `/usr/portage/eclass/*` – some of the man pages may be outdated
- Grant Goodyear's not-at-all poor deconstruction of the scripts above – http://dev.gentoo.org/~g2boojum/portage.html

- The Unofficial Gentoo Development Guide – http://dev.gentoo.org/~plasmaroo/devmanual/
- The Gentoo Developer Handbook – http://www.gentoo.org/proj/en/devrel/handbook/handbook.xml
- Some Gentoo sandbox documentation – http://bugday.gentoo.org/sandbox.html
- `#gentoo-dev-help` on `irc.freenode.net`

Q&A (a.k.a. *burn* him!)

Thanks for coming!