Overview of Gentoo

Richard Freeman <rich0@gentoo.org>
Gentoo Developer

PLUG North

Mar 09, 2010

Content

- About Me
- About Gentoo
- What makes Gentoo different?
- Are you ready for Gentoo?
- Using Gentoo
- Installing Gentoo
- How It Works

About Me

- Background in Biochemistry
- First Linux experience was early 90s Slackware CD – on UMSDOS and a 486.
- Later Mandrake, and then Gentoo
- Currently in Pharmaceutical IT
- Member of the Gentoo amd64 arch team, and I maintain a few packages

About Gentoo

Social Contract

- Gentoo is and will remain Free Software
- We will give back to the Free Software Community
- We will not hide problems

http://www.gentoo.org/main/en/contract.xml

Meta-Structure

- Two Governing Bodies
  - Gentoo Trustees – governs Gentoo Foundation, which owns and manages all Gentoo property.
  - Gentoo Council – manages all Gentoo projects and the distribution.
- Many Project Teams
  - Self governing, but fall under Council or Trustees

What Makes Gentoo Different?

A Source-Based Distro

- All packages built from source on the local PC
- Source typically distributed identical to upstream, and patches applied locally
- A script automates the build and install process for every package
- Strong dependency management

What's the Bottom Line?

- End users have a lot more choice
  - Use Flags
  - CFLAGS/LDFLAGS/etc
- No distro releases – per-package QA
  - Few restrictions on library versions
- Core distro packages are a minimal set + toolchain
- Can both patch and brand upstream apps (no iceweasels)
- MUCH slower installs for big packages

What's the Bottom Line?

- Users can run into build issues – QA is more complex due to heterogeneity.
- Developers and testers often run into gcc/etc bugs.
  - Good for GCC
- Documentation and Forums tend to be strong (varies)
- Testing-level packages tend to be very current
- Tends to push the envelope (openrc, --as-needed)
- Config-file maintenance tends to be smooth
- Good dependency management – requested vs incidental installs

Are You Ready For Gentoo?

YES!!! If...

- You're willing to learn some of the guts of linux.
- You're able to follow some install scripts.
- You're willing to Google/ask for help.

- Gentoo isn't always the right solution, but it probably isn't because you can't handle it.

Using Gentoo

Day-to-day Maintenance

- Sync packages: emerge –-sync
- Update packages: emerge -u world
- Install package: emerge apache
- Update config files – etc-update or cfg-update

Installing Gentoo

What you Need

- Go to the Gentoo website and choose Get Gentoo!
- Consider printing the arch handbook, but you will have web access during the install
- Download/burn an install CD - updated weekly
- Download a portage snapshot to speed up the first sync (optional)

Install Overview

- Installation is completely manual, but scripted.
- The script covers a lot of atypical situations, many steps are skipped by most users.
- For a new user plan 1-2 hours, system will be usable after this, but minimal

Install Steps

- Get the install env working (network/etc) – typically automatic
- Partition
- Install the stage3 and package repository
- Configure portage and chroot
- Configure kernel, system tools, bootloader
- Configure users and cleanup

How It Works

The Portage Tree

- Based on BSD Ports System
- /usr/portage contains a collection of packages, defined by ebuilds
- Ebuilds are essentially automated build scripts
- /usr/portage also contains lots of distro metadata, and libraries used for builds

Irssi Ebuild

```
# Copyright 1999-2009 Gentoo Foundation
# Distributed under the terms of the GNU General Public License v2
# $Header: /var/cvsroot/gentoo-x86/net-irc/irssi/irssi-0.8.14.ebuild,v 1.9 2009/11/22 21:58:41 swegener Exp $

EAPI="2"

inherit perl-module

DESCRIPTION="A modular textUI IRC client with IPv6 support"
HOMEPAGE="http://irssi.org/"
SRC_URI="http://irssi.org/files/${P}.tar.bz2"

LICENSE="GPL-2"
SLOT="0"
KEYWORDS="alpha amd64 arm hppa ia64 ~mips ppc ppc64 s390 sh sparc x86 ~x86-fbsd"
IUSE="ipv6 +perl ssl socks5"

RDEPEND="sys-libs/ncurses
        >=dev-libs/glib-2.2.1
        ssl? ( dev-libs/openssl )
        perl? ( dev-lang/perl )
        socks5? ( >=net-proxy/dante-1.1.18 )"
DEPEND="${RDEPEND}
        >=dev-util/pkgconfig-0.9.0"
RDEPEND="${RDEPEND}
        perl? ( !net-im/silc-client )
        !net-irc/irssi-svn"

src_prepare() {
        epunt_cxx
}
```

Irssi Ebuild

```
src_configure() {
	econf \
		--with-proxy \
		--with-ncurses \
		--with-perl-lib=vendor \
		$(use_with perl) \
		$(use_with socks5 socks) \
		$(use_enable ssl) \
		$(use_enable ipv6) \
		|| die "econf failed"
}

src_install() {
	emake \
		DESTDIR="${D}" \
		docdir=/usr/share/doc/${PF} \
		install || die "make install failed"

	use perl && fixlocalpod

	dodoc AUTHORS ChangeLog README TODO NEWS || die "dodoc failed"
}
```

Ebuild Metadata

```
# Copyright 1999-2009 Gentoo Foundation
# Distributed under the terms of the GNU General Public License v2
# $Header: /var/cvsroot/gentoo-x86/net-irc/irssi/irssi-0.8.14.ebuild,v 1.9 2009/11/22 21:58:41 swegener Exp $

EAPI="2"
```

- The version of the PMS ebuild specification in use – used by package manager to parse ebuild.

```
inherit perl-module
```

- This is an eclass – a library of script functions and overrides useful for various ebuilds

```
DESCRIPTION="A modular textUI IRC client with IPv6 support"
HOMEPAGE="http://irssi.org/"
SRC_URI="http://irssi.org/files/${P}.tar.bz2"
LICENSE="GPL-2"
```

- About the package and upstream – including where to get the source. The source will be mirrored, and mirrors will be used automatically.
- Users who are picky about licenses can configure their system to not install packages that don't meet their needs. Portage contains the full text of every license used by a package in /usr/portage/licenses
- P is set to the package name and upstream version (irssi-0.8.14)

```
SLOT="0"
```

- If two versions of a package have different SLOTS they can be installed side-by-side (glibc, kernel-sources, etc)

```
KEYWORDS="alpha amd64 arm hppa ia64 ~mips ppc ppc64 s390 sh sparc x86 ~x86-fbsd"
```

- “arch” is stable, “~arch” is unstable/testing, “-arch” is known to fail, no listing is unsupported/unknown
- (BTW, look at all the archs!)

```
IUSE="ipv6 +perl ssl socks5"
```

- The ebuild supports the listed use flags to customize behavior, and perl is enabled by default if the user did not explicitly disable it.

Ebuild Dependencies

RDEPEND="sys-libs/ncurses

- REDEPEND = needed to run package, but not to build it
- Any version of ncurses will do

>=dev-libs/glib-2.2.1

- Any version of glib >= 2.2.1 is fine

ssl? (dev-libs/openssl)

- If the ssl use flag is enabled, then the package needs openssl to run

perl? (dev-lang/perl)
socks5? (>=net-proxy/dante-1.1.18)"
DEPEND="${RDEPEND}
>=dev-util/pkgconfig-0.9.0"

- DEPEND = needed to build the package, but not to run it
- In this case, the run-dependencies are needed to build as well

RDEPEND="${RDEPEND}
perl? (!net-im/silc-client)
!net-irc/irssi-svn"

- These are blockers – the package cannot be installed if irssi-svn is installed (same package, but the live svn version), and if perl is enabled it conflicts with silc-client
- Most common reason for blockers is two packages that create the same file, with no easy workaround.

Ebuild Execution

- The package manager sources an ebuild to learn about its metadata
- To install, the package manager calls a number of functions in sequence – some have default definitions
- Common functions overridden in ebuilds include: pkg_setup, src_unpack, src_prepare, src_configure, src_compile, src_test, src_install, pkg_preinst, pkg_postinst, pkg_prerm, pkg_postrm
- Ebuilds run in a sandbox – usually non-root for most phases – they install to a temporary directory
- The package manager copies the files onto the actual system so that it can manage them

The Build Itself

```
src_prepare() {
	epunt_cxx
}
```

- Disable checks for a c++ compiler.

```
src_configure() {
	econf \
```

- Econf is a wrapper for configure, with overrides for install locations / etc so that it stays in the sandbox – it passes additional parameters through to configure:

```
		--with-proxy \
		--with-ncurses \
		--with-perl-lib=vendor \
		$(use_with perl) \
```

- use_with and use_enable are functions that will pass on a –-with or –-enable based on whether the indicated use flag is enabled.

```
		$(use_with socks5 socks) \
		$(use_enable ssl) \
		$(use_enable ipv6) \
		|| die "econf failed"
}
```

The Build Itself

```
src_install() {
        emake \
```

- Emake is also a wrapper for make, to play nicely in the sandbox
- Ugh, emake probably should be in src_compile() - for readability only

```
                DESTDIR="${D}" \
```

- D is set to the sandbox install directory

```
                docdir=/usr/share/doc/${PF} \
```

- PF is the full package name and version

```
                install || die "make install failed"

        use perl && fixlocalpod
```

- This is defined in the perl-module eclass and it nukes pod files

```
        dodoc AUTHORS ChangeLog README TODO NEWS || die "dodoc failed"
```

- dodoc takes the files listed and puts them in the documentation directory

```
}
```

Gentoo On Your Server

What, are you nuts?

Maybe, maybe not...

But, you guys asked...

Why Gentoo On a Server

- Strong dependency management
- USE flags allow more minimal installs
- Split packages allow more minimal installs
- Good config file management
- You can delay updates with some care
- You can easily audit for security bugs
- If you need unusually-built packages, very easy to do

Why not?

- Stable on Gentoo is more similar to Ubuntu than Debian – generally no backports, and no LTS
- Gentoo is designed for frequent small updates – you can batch but don't do it once a year
- Needs some care (staging binaries / dev-test-prod / etc)

Bottom Line

- Understand it before you decide!
- There are successful Gentoo datacenter deployments
- You can't deploy and forget for a few years, but if you can get economy of scale that may not be a problem
- This topic alone could fill an hour

Acknowledgments

- Luca Barbato and Lars Weiler for the presentation template.