Portage - A look under the surface

dev.gentoo.org/~genone/docs/fosdem-2007-talk.pdf

Marius Mauch <genone@gentoo.org>

Free and Open Source Developers European Meeting, 2007

Contents

- Recently added features
- The EAPI problem
- Backend technologies
- Popular requests and their problems
- Discussion

Marius Mauch <genone@gentoo.org>

Recently added features

elog

Resolver improvements

Manifest2

License filtering

Per-package default USE flags

Others

Where to find new features

- `/usr/share/doc/portage-$version/NEWS.gz`
- sticky topics on the “Portage and Programming” forum
- GWN
- gentoo-portage-dev mailing list
- gentoo-dev mailing list
- #gentoo-portage IRC channel

elog: What is elog?

- A framework for logging ebuild generated messages
- An ebuild function to send messages to the user
- A set of modules to dispatch messages to the user

elog: What is elog?

- A framework for logging ebuild generated messages
- An ebuild function to send messages to the user
- A set of modules to dispatch messages to the user
- A solution to an annoying problem

elog: How does it work?

1. Ebuild sends a message via einfo/elog/ewarn/eerror
2. At merge time of each package portage collects these messages
3. Collected messages are filtered and relayed to dispatch modules
4. Dispatch modules process the messages
5. At exit time portage calls finalization code of dispatch modules

elog: Available modules

save	Saves messages in files (one file per package)
mail	Sends messages by mail (one mail per package)
syslog	Inject messages into the syslog facility
custom	Pass messages to a custom logging tool/script
???	New modules are easy to create

elog: Problems

- Addition of new preferred loglevel
- Not compatible with old enotice hack
- No "official" tools for handling logfiles
- Sometimes confusing configuration
- Messages not always in chronological order

Resolver improvements: Blockers

- Blockers from installed packages are now also considered
  - `DEPEND="!foo"` in package `bar` automatically adds `DEPEND="!bar"` in package `foo`
- In some cases blockers can be handled automatically without manual unmerge
  - Only works when the blocker is made irrelevant by an update in the same graph

Resolver improvements: Full dependency graph

- Enables detection of circular dependencies
- Avoids up-/downrade cycles
- Ensures correct merge order
- Includes multiple SLOTs of a package

Manifest2: Overview

- A new container format for ebuild related checksums
- Expected to replace the old digest system in the near future
- Fully backwards compatible
- Adds more control about checksum verification

Manifest2: Motivation

The old checksum system was very inefficient:

- Filename and size had to be repeated for each checksum
- Needed many tiny files - high filesystem overhead
- All files equal

The new system handles this much better:

- Each file is only listed once
- All checksums are stored in a single file (per package)
- Allows specifying a filetype that can be used for filtering

License filtering: What is that?

- More commonly known as ACCEPT_LICENSE
- New visibility filter on top of keyword filtering (later)
- Replacement for check_license ebuild function

License filtering: What is that?

- More commonly known as ACCEPT_LICENSE
- New visibility filter on top of keyword filtering (later)
- Replacement for check_license ebuild function
- Implemented but not yet merged

License filtering: Motivation

- People want to build “free” systems
- Replace check_license with a better system
- Generally make use of the LICENSE variable

Default IUSE flags

- Allows recommending USE flags per package
- Extends IUSE syntax with Example: `+flag`
- Needs EAPI bump before usage (later)
- Lowest config layer - overridden by all other layers

Per profile package.use

- Like normal package.use
- Useful for fine-graining profile use flags
- Overridden by flags in make.conf (unlike normal package.use)
- Less compability issues than default IUSE flags

Directories-as-files: WTF does that mean?

- Most config files can be splitted in multiple files
- Allows a more flexible organization
- Files can also be symlinks - useful for shared configuration
- Can reserve filenames for special uses
- Also supports subdirectories

SLOT dependencies

- Allows to match only ebuilds in a specific SLOT
- Syntax: Example: `foo/bar:2` to only select ebuilds with SLOT=2
- Possible replacement for version ranges in some cases
- Needs EAPI bump before usage in ebuilds (later)

New FEATURES

parallel-fetch performs downloading and compiling in parallel

userfetch downloads distfiles as unpriviledged user

stricter Makes QA checks more severe (aborts build if checks fail), not intended for normal usage

installsources Install sourcecode as well as compiled files

splitdebug Store debug symbols in seperate files when stripping binaries (needs installsources)

Others

per package use.mask Allows profiles to mask USE flags per package

forced USE flags Allows profiles to force USE flags as “on”

extended versioning syntax Support for cvs versions and multiple suffixes

bashrc hooks Can run custom code before/after each ebuild phase

rsync option handling New variables to send options to rsync in a generic way

The EAPI problem

EAPI description

EAPI problems

EAPI: Motivation

- Introduction of new features
  - Changes to ebuild phases (src_configure)
  - Metadata syntax changes (default IUSE)
  - New ebuild functions (elog)
- Removing legacy stuff
  - RESTRICT=noXXX
  - PROVIDE entries

EAPI: Implementation

- New metadata key
- Packages with unsupported EAPI value are masked
- Switching semantics based on EAPI value

EAPI: Implementation

- New metadata key
- Packages with unsupported EAPI value are masked
- Switching semantics based on EAPI value
- Mostly untested so far

EAPI: Limitations

- Limited to ebuild internals
- Not only ebuilds need versioning (repo layout, support files)
- No clear syntax definition
- Alternative: Repository versioning

EAPI: Problems

- Current semantics are mostly undefined (EAPI=0)
- Planned semantics are undefined (EAPI=1)
- Exact syntax is undefined
- Current implementation is limited to visibility filtering

Backend technologies

The visibility system

The configuration stack

The cache system

Visibility system: Concept

- Usually referred to as “masking”
- Core: All ebuilds available
- Core selection is processed through boolean filters
- Multiple layers stacked on each other
- Visibility vs. Grouping

Visibility system: Layers

Corruption	if metadata is unreadable
EAPI	if ebuild version isn't supported
Profile	if package is incompatible with profile
Package.mask	general purpose (configurable)
Keywords	Stability/Age (configurable)
License	if license is unacceptable (configurable)

- potentially others

Configuration stack: Concept

- Portage configuration assembled from multiple locations
- Files and variables
  - Incremental and Override variables
  - Files are generally incremental
- Locations define responsibility
- Non-unified semantics

Configuration stack: Variable Layers

Make.globals Portage defaults (Portage developers)

Make.defaults Profile settings (profile maintainers, Releng)

Make.conf Global user configuration (user)

Environment General override (user, tools)

Configuration stack: File Layers

$PORTDIR/profiles Global repository settings (ebuild devs)

/etc/make.profile Profile settings (profile maintainers, Releng)

/etc/portage/profile Profile overrides (user)

/etc/portage User settings (user)

Not all files available in all locations!

Cache system: Concept

- Motivation: ebuild parsing is slow
- Parsed metadata variables/values are cached
- Modular

Cache system: Restrictions

- Values must be constant
  - Metadata variables can't be affected by local configuration
- Cache must be checked for staleness
  - Complex due to eclasses
  - Makes access rather slow
- Access patterns are fixed → less optimization potential

Popular requests and their problems

Database backend

Reverse dependency support

USE dependencies

Remote ebuild tree

GPG support

Why not use a database backend?

- Usual problem: way to vague
  - Backend for what?
  - What exactly is a “database”?
- General misassumption: database == fast

Problems of a RDBMS cache backend

- Additional dependencies
- No practical benefit
  - data access patterns aren't designed with a RDBMS in mind
  - portage already does it's own caching
- sqlite/mysql modules already exist(ed)

Portage doesn't have reverse dependency support

- "Reverse dependencies" aren't a feature but a property of a graph edge
- People refer to different features here:
  - Automatic rebuilding of link-level dependencies (`revdep-rebuild`)
  - Recursive unmerging of dependencies (`emerge --depclean`)
  - Unmerging of reverse dependencies
  - Prevent unmerging of dependencies

Portage lacks USE dependencies

- Most wanted feature by ebuild developers
- Just nobody has implemented it yet in portage
- People are scared by resolver code apparently

Why can't portage download the ebuilds when it needs them?

- Lack of atomicity
- Possible integrity issues
- Requires to be always online
- Lack of container format
- Harder to support
- Metadata has to be fetched completely anyway

What's the status of GPG support in portage?

- Original plans date back to 2003 and earlier
- No real specification was ever made
- No key policy was ever created
- Some prototype code was added to portage in 2004
- Current GPG support is very incomplete and unmaintained
- A new specification is in the works (hopefully)

Discussion

Marius Mauch <genone@gentoo.org>