

Portage - A modern package manager

Marius Mauch <genone@gentoo.org>

FOSDEM 2005



Outline

Package Managing in General

Portage - The Current Implementation

Features

The Portage Tree

The Future

Short Term

Long Term

Conclusion



What is a Package?

- Simplest case: tarball of files



What is a Package?

- Simplest case: tarball of files
- More general:
 - Payload: actual source or binary files
 - Metadata: descriptive information (homepage, dependencies, ...)
 - Instructions: build and install scripts



What is a Package?

- Simplest case: tarball of files
- More general:
 - Payload: actual source or binary files
 - Metadata: descriptive information (homepage, dependencies, ...)
 - Instructions: build and install scripts
- Different kind of packages: binary vs. source (vs. installed)



What is a Package?

- Simplest case: tarball of files
- More general:
 - Payload: actual source or binary files
 - Metadata: descriptive information (homepage, dependencies, ...)
 - Instructions: build and install scripts
- Different kind of packages: binary vs. source (vs. installed)
- Common formats: Tarball, RPM



Tasks of a Package Manager

- Installation and removal of packages



Tasks of a Package Manager

- Installation and removal of packages
 - Dependency handling



Tasks of a Package Manager

- Installation and removal of packages
 - Dependency handling
- Searching for packages



Tasks of a Package Manager

- Installation and removal of packages
 - Dependency handling
- Searching for packages
- Querying metadata from packages



Tasks of a Package Manager

- Installation and removal of packages
 - Dependency handling
- Searching for packages
- Querying metadata from packages
- Conversion between package formats



Existing Package Managers

- RPM - Redhat, Suse, Mandrake, ...
- dpkg - Debian
- Portage - Gentoo
- Ports - FreeBSD, OpenBSD, NetBSD
- MSI - Windows
- ...



Outline

Package Managing in General

Portage - The Current Implementation

Features

The Portage Tree

The Future

Short Term

Long Term

Conclusion



USE Flags

- Specify optional behavior



USE Flags

- Specify optional behavior
- Abstraction over `./configure` arguments



USE Flags

- Specify optional behavior
- Abstraction over `./configure` arguments
- USE flags can define optional dependencies



USE Flags

- Specify optional behavior
- Abstraction over `./configure` arguments
- USE flags can define optional dependencies
- **Example:** `USE=ssl`
 - Enables optional OpenSSL support in all packages
 - Adds OpenSSL as additional dependency



The Concept of Masking

- Only one repository for all branches and versions



The Concept of Masking

- Only one repository for all branches and versions
- Each package can have multiple versions available



The Concept of Masking

- Only one repository for all branches and versions
- Each package can have multiple versions available
- Current version selected by user configuration



The Concept of Masking

- Only one repository for all branches and versions
- Each package can have multiple versions available
- Current version selected by user configuration
- Influencing factors: platform, base profile, branch, global mask list



The Concept of Masking

- Only one repository for all branches and versions
- Each package can have multiple versions available
- Current version selected by user configuration
- Influencing factors: platform, base profile, branch, global mask list
- Manual selection also possible



Emerge - The Convenient User Interface

- One command to search, download, build, install, uninstall



Emerge - The Convenient User Interface

- One command to search, download, build, install, uninstall
- **Example:** `emerge gphoto2`



Emerge - The Convenient User Interface

- One command to search, download, build, install, uninstall
- **Example:** `emerge gphoto2`
 - Check repository for current version of gphoto2



Emerge - The Convenient User Interface

- One command to search, download, build, install, uninstall
- **Example:** `emerge gphoto2`
 - Check repository for current version of gphoto2
 - Automatically install all dependencies of gphoto2



Emerge - The Convenient User Interface

- One command to search, download, build, install, uninstall
- **Example:** `emerge gphoto2`
 - Check repository for current version of gphoto2
 - Automatically install all dependencies of gphoto2
 - Download, compile and install gphoto-2.1.5.tar.gz



Emerge - The Convenient User Interface

- One command to search, download, build, install, uninstall
- **Example:** `emerge gphoto2`
 - Check repository for current version of gphoto2
 - Automatically install all dependencies of gphoto2
 - Download, compile and install gphoto-2.1.5.tar.gz
- Same tool to update the repository: `emerge sync`



The Portage Tree Structure

- Main database for Portage



The Portage Tree Structure

- Main database for Portage
- Everything provided in plaintext



The Portage Tree Structure

- Main database for Portage
- Everything provided in plaintext
- All information in one place (exception: payload)



The Portage Tree Structure

- Main database for Portage
- Everything provided in plaintext
- All information in one place (exception: payload)
- Components:
 - Ebuilds: The actual packages



The Portage Tree Structure

- Main database for Portage
- Everything provided in plaintext
- All information in one place (exception: payload)
- Components:
 - Ebuilds: The actual packages
 - Eclasses: Common code



The Portage Tree Structure

- Main database for Portage
- Everything provided in plaintext
- All information in one place (exception: payload)
- Components:
 - Ebuilds: The actual packages
 - Eclasses: Common code
 - Profiles: Basic configuration



Ebuilds

- Simple Bash scripts



Ebuilds

- Simple Bash scripts
- Define metadata and instructions



Ebuilds

- Simple Bash scripts
- Define metadata and instructions
- Instructions separated in phases, each phase being one bash function



Ebuilds

- Simple Bash scripts
- Define metadata and instructions
- Instructions separated in phases, each phase being one bash function
- Defaults provided by Portage, often no/little instructions required



Eclasses

- Same syntax and semantics as ebuilds



Eclasses

- Same syntax and semantics as ebuilds
- Common code for multiple ebuilds



Eclasses

- Same syntax and semantics as ebuilds
- Common code for multiple ebuilds
- Also template for related ebuilds (perl modules, java libraries)



Eclasses

- Same syntax and semantics as ebuilds
- Common code for multiple ebuilds
- Also template for related ebuilds (perl modules, java libraries)
- Easy to extend Portage



Profiles

- Default configurations



Profiles

- Default configurations
- Important settings: base system, default USE flags



Profiles

- Default configurations
- Important settings: base system, default USE flags
- Different profiles based on version and platform



Profiles

- Default configurations
- Important settings: base system, default USE flags
- Different profiles based on version and platform
 - **Example:** `default-linux/amd64/2004.3`



Outline

Package Managing in General

Portage - The Current Implementation
Features
The Portage Tree

The Future
Short Term
Long Term

Conclusion



Next Portage Versions

- Portage-2.1:
 - Maintenance release to solve long standing issues
 - Enhancing current dependency resolver
 - Major performance improvements
 - Security improvements
 - Code cleanup and documentation



Next Portage Versions

- Portage-2.1:
 - Maintenance release to solve long standing issues
 - Enhancing current dependency resolver
 - Major performance improvements
 - Security improvements
 - Code cleanup and documentation
- Portage-3.0:
 - New dependency resolver
 - Modular design
 - Multi-repository support
 - Non-ebuild package support



Component Based Architecture

- Goal: build framework for package managers



Component Based Architecture

- Goal: build framework for package managers
- Replacable components



Component Based Architecture

- Goal: build framework for package managers
- Replacable components
- Communication over well-defined interfaces



Component Based Architecture

- Goal: build framework for package managers
- Replacable components
- Communication over well-defined interfaces
- Enable user-costumization



Long Term

Component Based Architecture

- Goal: build framework for package managers
- Replacable components
- Communication over well-defined interfaces
- Enable user-costumization
- Vision: common package management protocol



Abstract Packages

- Goal: enable users to use multiple package formats



Abstract Packages

- Goal: enable users to use multiple package formats
- Find common set of metadata for package formats




Abstract Packages

- Goal: enable users to use multiple package formats
- Find common set of metadata for package formats
- Use OOP to abstract from package formats




Abstract Packages

- Goal: enable users to use multiple package formats
- Find common set of metadata for package formats
- Use OOP to abstract from package formats
- Conversion between package formats by transformation engines 



Abstract Packages

- Goal: enable users to use multiple package formats
- Find common set of metadata for package formats
- Use OOP to abstract from package formats
- Conversion between package formats by transformation engines 
- Vision: define packages by quality, not format



Outline

Package Managing in General

Portage - The Current Implementation

Features

The Portage Tree

The Future

Short Term

Long Term

Conclusion



Summary

- There are more package managers than RPM



Summary

- There are more package managers than RPM
- Main goals of Portage: convenient usage and easy development for package maintainers



Summary

- There are more package managers than RPM
- Main goals of Portage: convenient usage and easy development for package maintainers
- Current codebase is grown and hard to maintain



Summary

- There are more package managers than RPM
- Main goals of Portage: convenient usage and easy development for package maintainers
- Current codebase is grown and hard to maintain
- Future:
 - Short term: 2.1: Maintenance release, 3.0: Major redesign
 - Long term: Build a generic package management framework



Questions



Examples



Ebuild-Example: Zoinks (1)

```
inherit eutils
```

```
DESCRIPTION="programmer's text editor and development environment"  
HOMEPAGE="http://zoinks.mikelockwood.com/"  
SRC_URI="http://zoinks.mikelockwood.com/download/${P}.tar.gz"  
LICENSE="GPL-2"  
SLOT="0"  
KEYWORDS="x86 ppc ~amd64"  
IUSE="nls imlib"  
DEPEND="nls? ( sys-devel/gettext )  
        imlib? ( media-libs/imlib )  
        virtual/x11"
```



Ebuild-Example: Zoinks (2)

```
src_compile() {  
    epatch ${FILESDIR}/xorg-library-configure.patch  
    econf $(use_enable nls) $(use_enable imlib) || die  
    emake || die  
}  
  
src_install() {  
    make DESTDIR="${D}" install || die  
    dodoc README INSTALL AUTHORS NEWS ChangeLog  
}
```



Graphs



