Vlastimil Babka
caster@gentoo.org

Towards "emerge gentoo-kernel"

Automatic kernel configuration and building in Gentoo

Linux Days Prague 2014

Current state

- sys-kernel/*-sources packages
    - gentoo-sources, vanilla-, git-, hardened- ...
- Ebuilds just unpack sources to /usr/src/
    - User configures, builds, installs manually
    - Or uses genkernel, invoked also manually

Current state

- sys-kernel/*-sources packages
  - gentoo-sources, vanilla-, git-, hardened- ...
- Ebuilds just unpack sources to /usr/src/
  - User configures, builds, installs manually
  - Or uses genkernel, invoked also manually
- THIS IS (NOT) GENTOO!!!
  - Looks like Gentoo installation process
  - Does not look like normal Gentoo usage
  - Why should the kernel be different?

Why is it bad?

- Manual work – boring, error prone, ...
  - No binpkg support
- Space occupied uselessly in /usr/src/
  - Most sources not needed for module building
  - Object files if user doesn't delete them
    - emerge -C won't clear them away for you
  - 3.16.3: **635MB** sources, **365MB** objects
- Packages depending on kernel config
  - emerge warns (or dies) when something not enabled, user has to adjust+rebuild

Why is it like that?

- Building itself is simple
  - Make -jX
- Installing could be more tricky than your average ebuild's make install
  - initrd creation, grub config, ...
  - But let's assume genkernel works
- The obvious caveat is *configuration*
  - Thus the focus of this talk

Kernel configuration

- The proper Gentoo way: USE flags!
  - openSUSE 3.16.3 .config has 6593 lines...
- Very system- and user-specific
  - Drivers, features, tuning, debugging, ...
  - Wrong .config → unbootable system
- But binary distros manage this somehow?
  - One (or few) configs enough for *everyone*
  - We could just package such distro kernel?
  - That would be lame, so just steal the .config!

Step 1: Gentoo .config

- Let's create a generic configuration!
  - Maintained by the Gentoo kernel team
  - Possibly starting with e.g. openSUSE .config
  - Hopefully compatible with all ebuilds
- Ebuild will compile with it and install
  - Providing binaries would be non-Gentooish :)
  - You can choose gcc version as usual
  - Some customization could be possible
    - USE flags for desktop/server etc...
    - Maybe infer processor type from CFLAGS?

This has been done before!

- Funtoo has USE=binary for some kernels
  - debian-sources and openvz-rhel6-stable
  - Own enhanced fork of genkernel
- Calculate linux has USE=vmlinuz
  - calculate-sources
  - cl-kernel instead of genkernel for own build
- Various forums posts asking about this
  - Bug 491864 – use genkernel in ebuild
  - GLEP 26 (2004?) - just about the building

How to do it?

- USE=”binary” emerge gentoo-sources?
  - Funtoo and Calculate Linux do that, but...
  - Why leave the sources around?
    - Most not necessary to build e.g. kernel modules
    - openSUSE kernel-devel: 85MB (vs 635MB full)
    - Not installing sources by “*-sources” is weird
  - Complicated maintenance
    - Need to bump genpatches and config at the same time
    - Stabilization also at the same time

How to do it?

- A new package such as “gentoo-kernel”
  - Build in /var/tmp/portage, install vmlinuz etc.
  - Same distfiles as *-sources, plus config
    - Config maintained by Gentoo Kernel team
      - Needs updates for version bumps (esp. major)
      - Possible to do more variants per USE flags
  - /usr/src/linux-*: just files for module building
    - Similar to kernel-devel packages on binary distros
  - binpkg support should be possible
    - grub, initrd creation in pkg_postinst (genkernel?)
    - removal in pkg_postrm

Step 1: Pros and Cons

- Pro: **Very simple** for the user
  - Much simpler than now!
  - Better confidence in bug reports
  - But, more work for the kernel team :)
- Con: not custom enough for many users
- Con: large .config → long build times
  - openSUSE config: 40 min on i5 (ssd/tmpfs)
    - 6.5 min with trimmed down custom config
  - Funtoo page says 1 hour on i7 for debian-src

Step 1: Pros and Cons

- Con: large .config → modules eat disk
  - oS config: 5.2MB vmlinuz + 172MB modules
    - 5.5MB vmlinuz + 7.7MB mods with custom config
    - 2.4GB modules with DEBUG_INFO enabled!
      - Probably need to introduce debug USE flag...
- Con: large .config → temp build space
  - oS config: 1.4 GB (8.9GB with DEBUG_INFO)
    - Custom config: 365MB (w/o DEBUG_INFO)
  - Funtoo says 14GB tmpdir for debian-sources
  - Problem for tmpfs builds with <16GB RAM

Step 1: Pros and Cons

- Con (?): everything needs to be modules
  - Otherwise generic kernel image too large
  - Therefore, initrd is always needed
    - Some opportunity for trouble
- Con (?): what if the generic .config does not satisfy all portage ebuilds?
  - Ebuilds might request conflicting features?
- Can we deal with these disadvantages?
  - And still keep it relatively simple for user?

Step 2: User Config

- To deal with the cons mentioned, but stay simple, we need a way so that:
  - Users can state their .config requirements
  - Ebuilds can state their .config requirements
  - Things keep working on version bumps
- First idea: let user provide own .config
  - Possibly start with Gentoo generic .config
  - Remove unwanted drivers, set CPU type etc.
  - What about version bumps?

So What Can Go Bump?

- New .config options appear (all the time)
  - We don't want to go interactive in emerge
- Obsolete (deprecated) options disappear
- Special case: options can be renamed?
  - Or drivers replaced, such as cciss → hpsa
- Opts hidden behind new umbrella option
- Default value changes (SLAB → SLUB)
- Dependencies between options change

The Proposed Solution

- User says which config options she cares about having enabled/disabled/module...
  - E.g. start with generic gentoo config, specify CPU type, disable unwanted drivers...
  - Make some drivers built-in (thus no initrd)
  - Store result in /etc, kernel ebuild reads it
- Options not specified by user are taken from the generic Gentoo .config (*default*)
  - Remember, the Gentoo .config is always updated by us for the given kernel version

Practical Issues

- How to distinguish “options not specified by user” that should get default value?
  - For options where default matches user config, did user want that or just didn't care?
  - The default value might change in a new version, but the old value in user config wins?
- Before discussing solution, let's look at how .config files work internally
  - And how build .config will be created

How does .config work?

```
drivers/usb/Kconfig:

config USB_STORAGE
        tristate "USB Mass Storage support"
        depends on SCSI

config USB_STORAGE_DEBUG
        bool "USB Mass Storage verbose debug"
        depends on USB_STORAGE

config USB_STORAGE_REALTEK
        tristate "Realtek Card Reader support"
        depends on USB_STORAGE
```

```
.config example (module, enabled, disabled):

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
```

Build with user config

```
User .config (based on e.g. 3.12):

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
```

```
Gentoo .config (based on 3.13):

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
CONFIG_USB_STORAGE_REALTEK=m
CONFIG_USB_STORAGE_DATAFAB=m (new option)
```

```
Build .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
CONFIG_USB_STORAGE_DATAFAB=m
```

Build with user config

```
User .config (based on e.g. 3.12):

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
```

```
Gentoo .config (based on 3.14):

CONFIG_USB_STORAGE=m
# CONFIG_USB_STORAGE_DEBUG is not set (changed)
CONFIG_USB_STORAGE_REALTEK=m
CONFIG_USB_STORAGE_DATAFAB=m (new option)
```

```
Build .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
CONFIG_USB_STORAGE_DATAFAB=m
```

Practical Issues

- How to distinguish “options not specified by user” that should get default value?
  - For options where default matches user config, did user want that or just didn't care?
  - The default value might change in a new version, but the old value in user config wins?

Practical Issues

- How to distinguish “options not specified by user” that should get default value?
  - For options where default matches user config, did user want that or just didn't care?
  - The default value might change in a new version, but the old value in user config wins?
- Solution: tool which compares resulting user config with the default and *trims* it
  - Only options that differ stored as user config
  - Rest added from default → same final config

User config trimming

```
User .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
```

```
Gentoo .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
CONFIG_USB_STORAGE_REALTEK=m
```

```
Trimmed user .config:

# CONFIG_USB_STORAGE_REALTEK is not set
```

Build with user config

```
Trimmed user .config:

# CONFIG_USB_STORAGE_REALTEK is not set
```

```
Gentoo .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
CONFIG_USB_STORAGE_REALTEK=m
```

```
Build .config (same as before trim!)

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
```

Build with user config

Trimmed user .config:

```
# CONFIG_USB_STORAGE_REALTEK is not set
```

Gentoo .config **(new version)**:

```
CONFIG_USB_STORAGE=m
# CONFIG_USB_STORAGE_DEBUG is not set
CONFIG_USB_STORAGE_REALTEK=m
```

Build .config:

```
CONFIG_USB_STORAGE=m
# CONFIG_USB_STORAGE_DEBUG is not set
# CONFIG_USB_STORAGE_REALTEK is not set
```

Practical Issues

- What if some user options don't differ from default *now*, but users wants to override future default changes?
  - Add them to the “trimmed” config manually
  - Maybe won't happen in practice anyway
  - See if it's worth any tool support
    - Such as extended make menuconfig

User config adjustment

```
Gentoo .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
CONFIG_USB_STORAGE_REALTEK=m
```

```
User .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
# CONFIG_USB_STORAGE_REALTEK is not set
```

Trimmed **and edited** user .config:

```
# CONFIG_USB_STORAGE_REALTEK is not set
CONFIG_USB_STORAGE_DEBUG=y
```

Practical Issues

- What happens to options missing in user config due to dependencies? Such as a prerequisite option disabled by the user?
  - There is no “`#CONFIG_FOO is not set`” entry at all in the resulting config
  - Gentoo config will supply its own defaults, most likely enabled or module for drivers
  - Thus, these defaults will fail to be enabled
    - We want to warn about such cases (see later)
    - There would be lots of false warnings due to this

Masked Options Issue

```
Gentoo .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
CONFIG_USB_STORAGE_REALTEK=m
```

```
No mention of
CONFIG_USB_STORAGE_DEBUG and
CONFIG_USB_STORAGE_REALTEK
as they depend on USB_STORAGE
```

```
User .config (before trimming!):

# CONFIG_USB_STORAGE is not set
```

```
Trimmed user .config:

# CONFIG_USB_STORAGE is not set
```

We trim _DEBUG and _REALTEK away (missing in user .config also means "different value" than Gentoo config).

Masked Options Issue

```
Build .config (trimmed user + Gentoo defaults)

# CONFIG_USB_STORAGE is not set  (from user config)
CONFIG_USB_STORAGE_DEBUG=y     (from Gentoo default)
CONFIG_USB_STORAGE_REALTEK=m   (from Gentoo default)
```

`make oldconfig`
removes everything
not satisfied by deps

Gentoo defaults supplied for
unspecified options as usual.

```
Build .config after make oldconfig

# CONFIG_USB_STORAGE is not set
```

Here we compare with both user and Gentoo configs
and warn that _DEBUG and _REALTEK are missing because
there is a deps problem. But it's not useful in this case!

Masked Opts Solution?

```
Gentoo .config:

CONFIG_USB_STORAGE=m
CONFIG_USB_STORAGE_DEBUG=y
CONFIG_USB_STORAGE_REALTEK=m
```

```
User .config (before trimming):

# CONFIG_USB_STORAGE is not set
```

```
Trimmed user .config:

# CONFIG_USB_STORAGE is not set
# CONFIG_USB_STORAGE_DEBUG is not set
# CONFIG_USB_STORAGE_REALTEK is not set
```

Adjusted trimming:

Explicitly mark missing options as if they were disabled by user. (They effectively were!)

Masked Opts Solution?

```
Build .config (all effectively from user config)

# CONFIG_USB_STORAGE is not set
# CONFIG_USB_STORAGE_DEBUG is not set
# CONFIG_USB_STORAGE_REALTEK is not set
```

`make oldconfig` removes everything not satisfied by deps

Gentoo defaults have nothing to add in this case.

```
Build .config after make oldconfig

# CONFIG_USB_STORAGE is not set
```

Here we compare with both user and Gentoo configs. Gentoo config is no-op since it supplied no values here. User config supplied options that are now missing, but it's OK, since they were explicitly disabled ("not set").

How Does It Solve Things?

- New .config options appear on bump
  - Did not exist when making user config
  - Drivers likely to be enabled in Gentoo .config
    - Potentially unneeded modules will be installed
      - Gradually increasing number over time
    - Once in a while, user can update own config
    - Or could we distinguish drivers and disable them?
    - Could be masked by user disable umbrella option
      - Config dependency problem, discussed later
  - Other options - according to Gentoo .config
    - Should not result in misconfigured system

How Does It Solve Things?

- Options disappear on version bump
  - User did not care about? No problem.
  - User explicitly enabled? Issue warning.
    - User can decide not to boot the new kernel
- Options being renamed
  - Issue warning about old option gone
  - New option according to Gentoo .config
  - Or, there could be a list of known instances
    - Determined by us updating the Gentoo .config

How Does It Solve Things?

- Complete driver replacement
  - Warning would get issued
  - New driver would likely be enabled
    - But as a module – be careful!
  - Can't help if related configuration is different
    - No automatic solution to that...
    - Do not delete old kernels too quickly :)
- New umbrella option appears on bump
  - Gentoo .config has it likely enabled
  - If not, it's an option dependency problem

How Does It Solve Things?

- Default values changing on bump
  - Upstream changes masked by Gentoo .config
  - But the Gentoo .config may change values
  - User-specified values will override that
    - Tricky to issue some kind of warning here
  - Unspecified values will just change
    - Since the user did not care before to set the previous default explicitly, the new default should still work for him?

How Does It Solve Things?

- Dependencies between options change
  - User's or default options no longer have their deps satisfied or conflict with other options
    - Includes “used-disabled umbrella for new default-enabled option“ and “new umbrella for user-enabled option not enabled by default”
  - The safe solution here is to abort build for user's options and warn for default options
    - Do it in pkg_pretend phase to prevent surprises in the middle of a long emerge?
    - Experience will show how often this happens
    - Possibly handle some of this automatically?

Possible Improvements

- Updating user config to a new kernel
  - Should not be necessary as much as possible, but still helpful once in a while
    - Silence warnings due to options that are gone
    - Disable new drivers that came from the default
  - A tool could assist with the update
    - To see which drivers are “new”, it will need the original untrimmed user config – so it should just be kept around after trimming
    - Just run “make oldconfig” on the untrimmed user config, and store+trim the result
      - Caveat: make oldconfig will not propose Gentoo defaults

Could All This Be Simpler?

- Can't we just run “make localmodconfig” on the Gentoo default .config during each build? Don't think so...
  - Not reliable enough (?)
  - Will disable modules not currently loaded...
    - USB devices not plugged in since reboot
    - Network protocols not used yet
  - Would not allow other kinds of configuration changes

Step 3: Ebuild dependencies

- With user configs in place, supporting config requirements from ebuilds is easy
  - Ebuilds would install config snippets in /etc
    - Just reimplement linux-info.eclass functions ?
  - When creating final .config, process in following order:
    - Trimmed user .config copied as a whole
    - ebuild snippets add options unspecified by user
      - Warn for options specified differently by user
    - Default Gentoo config adds options not specified by user nor ebuild requirements

Problems With Ebuilds

- ebuilds may want conflicting options
  - If such exist, conflict in ebuild's DEPEND too!
- Kernel may need to be rebuilt and booted after new config snippets are installed
  - No way to trigger rebuild as subslots now do
  - Triggering reboot of course not an option :)
- Kernel options might need to be satisfied at build time already! (→ no install to /etc)
  - Keep using pkg_pretend, tell user to create temporarily the needed snippet manually

So What's The Plan?

- Create a gentoo-kernel package (step 1)
  - Build and install kernel from ebuild with single .config, support at least simple and common boot configurations
  - Try reuse experience from Funtoo/CL
- Prototype config manipulation (step 2)
  - Put it to some testing, see what was missed
- Config snippets from ebuilds (step 3)
  - Change eclass internals

Thank you.