GENTOO WORKSHOP

Mark Platek
Clarkson University
Fall 2010

Resources

Gentoo Handbook:
x86: http://www.gentoo.org/doc/en/handbook/handbook-x86.xml
amd64: http://www.gentoo.org/doc/en/handbook/handbook-amd64.xml

Gentoo Forums:
http://forums.gentoo.org/

Myself – ask if you have questions!

Email: platekme89@gmail.com
IM: backslash54 v2

About Gentoo

Gentoo is a “metadistribution”, due to its nearly unlimited configurability.

All programs, libraries, etc. are compiled from source, similar to LFS. However, Gentoo includes Portage, a tool that automates some compilation tasks.

Gentoo performs marginally better than binary distributions due to compiled programs being tuned to the host machine, but that's not as important as the extremely high degree of control over the system that Gentoo affords.

The cost of this power is complexity and effort. But, it's hard to go back to binary distributions once you grok Gentoo. :)

About Portage

Portage is Gentoo's analogue of a package manager (apt, yum, etc). But, portage automates the compilation and installation of a particular program or utility, rather than just the installation of a precompiled binary for that utility.

An “ebuild” is similar to a package in apt or yum. Each ebuild contains some metadata and a bash script that gets the package source and compiles it.

You can set compiler flags (CFLAGS) for gcc, and set per-package “use flags” that specify what functionality should be compiled-in. Portage will resolve package dependencies.

All ebuilds exist in the “portage tree”, located at /usr/portage. ebuilds are updated by pushing out changes to the portage tree. The portage tree can be extended using “overlays”, but we'll get to that much later.

To update the system, the local portage tree is synced with a fresher mirror, and modified ebuilds (and any relevant dependencies) are recompiled.

About this talk

I'll be running an example installation of Gentoo that you can follow along with on your own machines.

It is strongly recommended that you dual-boot or find some other way to perform the installation natively, rather than in a virtual machine.

Certain steps (ones involving lots of compilation) will take a very long time. We'll try to end our weekly meetings on these steps, so that you can do them in your own time. In fact, you can expect to find yourself waiting a lot. Be patient, good things come in time.

We will approximately follow the installation handbook, though some steps will be done out of order. Don't worry, it won't break anything.

If I am referring to a shell command, I'll put it in backticks (e.g. `pwd`).

Let's Begin!

- Make sure your machine is connected to the external Clarkson network via ethernet.
- Boot the liveCD. You can do this with a USB drive, too, if you want. Enter `gentoo` at the boot prompt. You can probably accept the default keymap.
  - NOTE: If you want to do a 64-bit installation (highly recommended if your hardware supports it), you must use a 64-bit liveCD!
- If you get udev errors, don't worry – it hasn't hung up.
- Eventually, you are dropped at a bash command line.

Network Test

- Do you have a network connection? Try `ping -c www.google.com`.
- If not:
  - Are you connected to the internal or external network?
  - Use `ifconfig` to see if you got an IP.
  - Try to get an IP using `dhcpcd eth0` if your primary NIC is eth0.

Partitioning

- Use 'fdisk -l' to see your drives and partitions. Find the one you want to use for Gentoo (or the free space you have set aside). Let us assume the drive is /dev/sda.
- You will need the following partitions at a minimum:
  - A >=15G partition for /
  - A ~100MB partition for /boot
  - A swap partition
- If you don't have a separate partition for /boot, you should probably make one – use `fdisk /dev/sda`. Don't forget to make the /boot partition bootable!
- So, your disk should not have the following partitions ready:
  - /dev/sda1: /boot partition
  - /dev/sda2: swap partition
  - /dev/sda3: / partition
- It's likely that everyone's partition numbers will be different, but I will use the above names to refer to each partition, in the same way as the handbook does.

Create Filesystems

- Let's put some shiny new filesystems on those partitions!
  - Create an ext2 partition on the boot partition (use `mkfs.ext2`)
  - In fdisk, set the partition label on the swap partition to 82 or “Linux swap/Solaris”
  - Create an xfs partition on the root partition (use `mkfs.xfs`)
- You can use ext4 if you want, instead of xfs.
- When you finish creating these filesystems, mount them like so:
  - /dev/sda3 (root partition) at /mnt/gentoo
  - `mkdir /mnt/gentoo/boot`
  - /dev/sda1 (boot partition) at /mnt/gentoo/boot
- Initialize the swap partition:
  - `mkswap /dev/sda2`
  - `swapon /dev/sda2`

A quick note about stages

- There are three possible “stages” from which to install Gentoo:
  - stage1: closer to LFS, you must compile the entire toolchain from scratch before you can proceed. (no longer supported - *officially*)
  - stage2: no longer supported at all
  - stage3: the toolchain is provided in a binary form
- The stage3 tarball we are about to download contains binaries of GCC and other toolchain utilities. This is OK, because they were compiled for x86 systems and will still run quite fast.
- However, you can recompile the entire toolchain to emulate a stage1 install from a stage3. This process will give you a toolchain built for your specific machine by the most recent version of GCC.
- The downside? It takes about 6 hours if you have a good machine.
- In the interests of time, we will only do a stage3 install.

Download base system

- Go to where you mounted the root drive: `cd /mnt/gentoo/`
- Check that your system clock is correct with `date`. If you need to fix it, instructions are in the handbook.
- Now for a bit of text-based web browsing! `links http://mirror.clarkson.edu/gentoo`
  - Go to releases/x86/current-stage3 and download the latest stage3 tarball.
  - You can use wget too, if you're lazy. The URL for the latest release is: http://mirror.clarkson.edu/gentoo/releases/x86/current-stage3/stage3-i486-20101012.tar.bz2
  - Or, if you are doing a 64-bit install: http://mirror.clarkson.edu/gentoo/releases/amd64/current-stage3/stage3-amd64-20101007.tar.bz2

Download base system (cont'd)

- Now, we need to grab a snapshot of the latest portage tree.
- Again, we'll use links: `links http://mirror.clarkson.edu/gentoo`
  - This time, descend into snapshots/ and download the latest snapshot
  - And if you are lazy and using wget: http://mirror.clarkson.edu/gentoo/snapshots/portage-20101010.tar.bz2
- Hit ESC to quit links after you've downloaded everything.

Create base system

- You should still be in /mnt/gentoo. Invoke these commands to extract the stage3 installation:
  - `tar xjvpf <stage3>.tar.bz2`
  - `tar xjf <snapshot>.tar.bz2 -C /mnt/gentoo/usr`
- The stage3 tarball will create the skeleton of a linux system. If you `ls` in /mnt/gentoo you'll see the standard layout (/home, /etc, /bin, etc.). It also contains the binary toolchain mentioned earlier.
- The portage snapshot contains a recent portage tree – note that it lives in /usr! We will need to freshen it before we use it to install anything (but, we'll get to this later).
- If you were going to perform a stage1 install, now would be the time to re-bootstrap and recompile the toolchain.

make.conf

- /etc/make.conf is Portage's main configuration file. In it, you specify things like compiler flags and Portage variables. We'll start writing it now.
- CFLAGS are your flags for gcc. CXXFLAGS are your flags for g++. We'll start with the following CFLAGS:
  - -O2: causes gcc to create optimized code that is generally safe. -O1 is default optimization and -O3 is not considered safe (but you can try if you are adventurous)
  - -fomit-frame-pointer: causes programs that don't need a frame pointer to be compiled without one. This frees up a register, but makes debugging difficult to impossible.
  - -pipe: causes gcc to pass temporary files through pipes rather than storing them to disk. It is very safe, but causes gcc to use slightly more memory.
  - -march=foo: is the most important! It causes gcc to optimize programs for your specific processor. Set foo to “i686” if you have a P4, “core2” if you have a Core 2 Duo, or “athlon64” if you have an Athlon.
- Please use the following CFLAGS and CXXFLAGS variables:

```
CFLAGS=“-02 -pipe -march=foo -fomit-frame-pointer”

CXXFLAGS=“${CFLAGS}”
```

- Check out the compiler optimization guide for info on CFLAGS: http://www.gentoo.org/doc/en/gcc-optimization.xml

make.conf (cont'd)

- We can also set the MAKEOPTS variable to tell gcc how many compilation threads to run at once.
  - The heuristic is to set the number of threads to one larger than the number of CPUs (or cores).
  - So, for a dual-core processor, it would be:

    ```
    MAKEOPTS="-j3"
    ```
- DON'T CHANGE YOUR CHOST! Just don't.
- We'll set a few other variables to tell portage which mirrors to use:

  ```
  GENTOO_MIRRORS="http://mirror.clarkson.edu/gentoo"
  SYNC="rsync://mirror.clarkson.edu/gentoo-portage"
  ```

make.conf (cont'd)

- Finally, we will set the USE variable. The USE variable specifies which use flags to always compile programs with.
- use flags are a way to manage which components are compiled-in to any given program. For example, if you wanted to compile gtk+ with Gnome support, you would set the 'gnome' use flag for the gtk+ ebuild.
- We will be using Gnome in this workshop, so we can all start with a common USE variable.
- Please use the following USE variable to start with:

```
USE="mms sse sse2 gnome gtk alsa X acpi -qt3 -qt4 -kde -arts
-pulseaudio -oss -esd
```

Chrooting

- So far we've been working on the installation from the liveCD's Gentoo environment. But, wouldn't it be nice to use our new system as if it were already up and running?
- We can do that, by using `chroot`! What is does is make the liveCD system think that it's rooted at some directory we specify – if that happens to be the root directory of our new installation, we can make the liveCD system perform the functions our new installation can't yet (like run a kernel).
- Perform the following steps in order:
  - `cp -L /etc/resolv.conf /mnt/gentoo/etc/` (copy over DNS information)
  - `mount -t proc none /mnt/gentoo/proc` (creates the /proc directory)
  - `mount -o bind /dev /mnt/gentoo/dev` (creates the /dev directory)
  - `chroot /mnt/gentoo /bin/bash` (change the filesystem root to /mnt/gentoo)
  - `env-update` (refresh some environment variables to reflect the new root)
  - `source /etc/profile` (export those new variables)
  - `export PS1="(chroot) $PS1"` (remind yourself that you are chrooted)
- That's it! You are now at a shell in your new Gentoo installation!

A little housekeeping...

- The first thing we'll do with out new system is to specify its locales.
  - Edit the file /etc/locale.gen, uncomment the line “en_US.UTF-8 UTF-8” to specify an English UTF-8 locale.
  - Then, run `locale-gen` to create the files needed to support the locales you just specified.
- Next, we will configure the timezone:
  - Do an `ls /usr/share/zoneinfo` to see all available timezones. We're going to use EST.
  - So, invoke `cp /usr/share/zoneinfo/EST /etc/localtime` to copy over the appropriate data.

Portage commands

- While we're technically working in the new installation, we have a ways to go yet! Let's review a few portage commands before we go any further.
  - `emerge` is the command used to install packages
  - `equery` is used to find out what use flags a package has available
  - `eselect` is used to configure system parameters (like which kernel is being used, or which version of python is in use)
  - `revdep-rebuild` is used to find packages that were compiled for old versions of libraries, and recompile them to use newer libraries, if they are available
- A few specific examples:
  - `emerge --sync` syncs the local portage tree to the mirror defined in make.conf
  - `emerge -uDN world` updates the entire system (more on this soon)
  - `emerge -s foo` searches the portage tree for an ebuild named foo
  - `eselect kernel list` lists all kernels that are installed on the system
- Invoke `man emerge` to find out a lot more.

Updating the new system

- Finally, we get to emerge something! First, sync the portage tree: `emerge --sync`.
- Check what packages are out of date: `emerge -puDN world`

  -p: “pretend” - don't actually install or change anything

  -u: “update” - check for packages for which a new ebuild is available

  -D: “deep” - resolve dependencies recursively

  -N: “newuse” - look for packages that have new use flags set since they were last compiled
- Finally, if the list looks reasonable, recompile everything: `emerge -uDN world`. It will take a long time. You can do this step now and wait, or do it later at your leisure.

Next week: Kernel compilation

That's all for this week! Why not spend some time playing with your new Gentoo system in the interval?

Next week, we'll be compiling our kernels! In order to do this, you must know your hardware! Please try to find out what chipset your machine's NIC, SATA controller, and sound card use. If you have any other funny hardware, find out what chipset it uses.

Copyright