Bedrock Linux

Daniel “paradigm” Thau
The Ohio State University
Open Source Club

2012-05-24

Bedrock Linux

Table of Contents

Legal Disclaimer

In the United States, the name “Linux” is a trademark registered to Linus Torvalds, and currently “Bedrock Linux” does not have the rights to use “Linux” in its name. Plans to acquire a license to use “Linux” are in place. While the discussed project is Linux-based, it is not officially endorsed by or otherwise affiliated with Linus Torvalds.

Slideshow Quality Disclaimer

It is intended that these slides be self-sufficient without the presenter. This means they are very wordy. I apologize.

tl;dr: too many words, sorry

The Perfect Distro

Decided to give this Linux thing a try.
Which distro should I use?

The Perfect Distro

Debian, RHEL clones

- \+ Stable (both reliable and unchanging)
- – Out of date

Arch, Sid, Rawhide

- \+ Access to cutting-edge packages
- – Unstable (less reliable, changes often)

Gentoo, LinuxFromScratch

- \+ Customizable
- – Work to setup/maintain

LinuxFromScratch, Tinycore

- \+ Minimal
- – Work to setup/maintain

Ubuntu, Mint

- \+ User-friendly

Knoppix, Slax

- \+ Portable

The Perfect Distro

So which should I chose?

The Perfect Distro

What features are the most important?
Which features can I give up?

The Perfect Distro

I choose *everything*.

The Perfect Distro

Bedrock aims to make most of the (often seemingly mutually-exclusive) benefits of various other Linux distributions available simultaneously and transparently, with as little overhead as possible.

- ✓ Debian or RHEL-clone's rock solid stability?
- ✓ Arch's cutting-edge packages? AUR?
- ✓ Gentoo's compilation automation options?
- ✓ Ubuntu's Unity? Mint's Cinnamon?
- ✓ Your-favorite-distro's your-favorite-feature? [1]

All at the same time, transparently, with effectively zero overhead.

[1] Well, some features, but definitely not all. At the moment Bedrock can not be honestly considered "user-friendly."

Bedrock-Only Features

In addition to doing (almost) anything any other distro can do, there are a number of things Bedrock can do which no other distro can.

Bedrock-Only Features

You can do a distro-upgrade (Debian 5→6, Ubuntu 12.04→12.10, etc) live, with almost no downtime - no need to stop your apache server, reboot, configure things while server is down, etc.

Bedrock-Only Features

If a distro-upgrade breaks anything, no problem—old distro's setup (can be) still there, ready to go.

Bedrock-Only Features

Minimal stress from any given package failing to work—just use one from another distro.

Packages feel disposable, like toothpicks. No need to fret over one breaking, just use another.

Testimonials

All of Bedrock's user base[2] is raving about how amazing it is!

[2] Well, erm, there is only one user at the moment—Bedrock's developer, who is admittedly quite biased—but these are 100% true stories.

Testimonials

“ When Quake Live’s Linux release came out, there was a bug with how it interacted with Debian’s X11. Apparently they focused on getting it to work with Ubuntu—so I just used Ubuntu’s X11 (while keeping everything else Debian) and everything worked great! ”

—Daniel “paradigm” Thau

Testimonials

“ When Debian 6 came out, my touchpad stopped working properly. Some other guy with similar hardware had to debug it and find some obscure X11 settings to fix it. I just had Bedrock continue using Debian 5’s X11 until I the solution was found while the rest of the system was Debian 6. Bedrock just gives me so much *choice!* ”

—Daniel “paradigm” Thau

Testimonials

“ I was recently discussing the uses of LD_PRELOAD with an Arch user, who said he uses it to fix a problem with Quake Live not supporting his cutting-edge libraries. I just use Bedrock have Quake Live use another distro’s libraries. ”

—Daniel “paradigm” Thau

Testimonials

“ I needed to get compiz working to give a presentation at some club I attend. Thing is, Arch’s compiz is apparently broken for reasons I don’t care to debug, and Debian’s X11 is to old to support compositing on my newish hardware. So I just ran Debian’s compiz in Arch’s X11—had it all up and running within a few minutes of setting out to do that. Good luck handling that situation so smoothly with other than Bedrock. ”

—Daniel “paradigm” Thau

Testimonials

“ I really like a math program called ‘Sage.’ The only distro I know which had it in its repos is Arch, so I just pacman’d it from Arch. However, apparently the Sage devs don’t care to ensure it works on Arch and broke something critical. Well, they test it on Ubuntu, so I just have it use the Ubuntu libraries now. Down time when Arch removed it from its repos[3]? Just the time to re-download the Ubuntu-packaged version. ”

—Daniel “paradigm” Thau

[3] Sage is available in AUR on Arch, but the build time is substantial.

Testimonials

“ I used awesome from the Arch[4] repos as my window manager a number of years ago. Well, an Arch update completely changed awesome’s config syntax—it suddenly used some crazy programming language I didn’t know at the time. All of my configs were broken, and I didn’t have the time to remake them. So I just used awesome from another distro and kept on chugging with my old configs for a while longer. Thanks Bedrock! ”

—Daniel “paradigm” Thau

[4] The config changed happend in 2008, yet Arch seems to have added awesome into its repository in 2009; perhaps my memory was faulty and it was another distro. The principle explained here should stand irrelevant of distro, but I want to avoid providing misinformation.

How It Works

Bedrock's magic is based around filesystem and PATH manipulation.

How It Works — Chroot

A *chroot* changes the apparent filesystem layout from the point of view of programs running within it. Specifically, it makes a directory appear to be the root of the filesystem.
For example:

- Firefox is located in `/var/chroot/arch/usr/bin/firefox`
- If one runs: `# chroot /var/chroot/arch /usr/bin/firefox`
- Firefox thinks it is located at `/usr/bin/firefox`
- When firefox tries to load `/usr/lib/libgtk2.0-0`
- It will actually read load `/var/chroot/arch/usr/lib/libgtk2.0-0`

How It Works — Chroot

Some people consider this a light-weight vitalization. This is false, for two main reasons:

- It is possible to deliberately break out of a chroot, especially as root.
  - \+ If you *want* to have programs in chroots interact with programs out of chroots, you can do so quite easily. This is much harder to do effectively with actual virtualization.
  - – However, this makes it a poor sandboxing tool. Actual virtualization is better for this.
- Programs in chroots still have the same access to your hardware as programs outside of them.
  - \+ Minimal overhead—can, for example, do 3D just as well as non-chroot.
  - – Again, possible security issues if you're trying to sandbox.

How It Works — Chroot

Bedrock has the full filesystem of other distros available on-disk, each in their own directory. If one would like to run a program from that distro, via chroot, the program can be tricked into thinking it is running in its native distro. It would read the proper libraries and support programs and, for the most part, just work.

How It Works — Bind Mounts

Linux can take mountable devices—such as usb sticks—and make their filesystems accessible at any folder on the (virtual) filesystem. Mounting usb sticks to places such as `/media/usbstick` or `/mnt/usbstick` are typical, but not required—just about any directory will work. Linux can also mount virtual filesystems, such as `/proc` and `/sys`. These don't actually exist on the harddrive—they're simply a nice abstraction.

How It Works — Bind Mounts

Moreover, Linux can *bind mount* just about any directory (or file, actually) to any other directory (or file). Think of it as a shortcut. This can "go through" chroots to make files outside of a chroot accessible inside (unlike symlinks).

How It Works — Bind Mounts

With bind mounts you can, for example, ensure you only have to maintain a single `/home` on Bedrock. That `/home` can be bind mounted into each of the distros chrooted filesystems so that they all share it. If you arbitrarily decide to stop using one distro's firefox and start using another's, you can keep using your same `~/.mozilla`—things will "just work."

How It Works — Bind Mounts

Through proper usage of chroots and bind mounts, Bedrock can tweak the filesystem from the point of view of any program to ensure they have access to the files they need to run properly while ensuring the system feels integrated and unified.

How It Works — PATH

Programs read your PATH environmental variable to see where to look for executables, and your LD_LIBRARY_PATH for libraries.
For example, with
`PATH="/usr/local/bin:/usr/bin:/bin"`
when you attempt to run "firefox", the system will check for firefox in the following locations (in the following order):

- `/usr/local/bin/firefox`
- `/usr/bin/firefox`
- `/bin/firefox`

How It Works — PATH

Using a specialized PATH variable, Bedrock can have a program attempt to run a (chrooted) program in another distro rather than only looking for its own versions of things. By changing the order of the elements in the PATH variable, search order (ie, priorities) can be given.

How It Works — PATH

Currently Bedrock prioritizes the “native” executables before searching through the other distros, and has a hard priority for the other distros. Future versions of Bedrock are planned to support a more capable system for fine tuning of which version of which program is executed where.

Design Choices

Due to Bedrock's unusual goals, several unusual design choices were made. These choices were the reason Bedrock's system requires its own distribution to be fully utilized rather than simply being grafted onto another distribution.

Design Choices — Simplicity

Understanding Bedrock's filesystem layout (with the chroots, bind mounts, and dynamic PATH) can be quite confusing. Additionally, due to the limited development team, no user-friend installer will be available for quite some time; users will be required to compile Bedrock Linux "from scratch." Moreover, users will have to maintain things on a very low level; they will be expected to, for example, hand-edit the init files (reasoning explained later).

In order to ensure Bedrock is viable for as many users as possible, everything which doesn't have to be confusing or complicated should be made as simple as possible.

Design Choices — Simplicity

Bedrock thus choses some unusual packages. GRUB, the de-facto bootloader for the vast majority of major Linux distributions, is a tad complicated. Syslinux is significantly easier to setup and maintain by hand, and thus is the “official” choice for Bedrock. However, GRUB should work fine, if the user wants to figure out how to install and manage it himself.

Design Choices — Minimalism and Deferring Features

Most major Linux distributions have much larger and more experienced teams. Where directly comparable, they are most likely better than the Bedrock developer and Linux-distro-making. Thus, where possible, it is preferable to use a client distro rather than Bedrock itself. If something can be deferred to a client distro, it will be; Bedrock only does what it has to do to enable the integration of other distros.

Design Choices — Statically-Linked Compilation

Typically, most executables refer to other libraries for their components. If this is done at runtime, this is known as *dynamic linking*. By contrast, one can (sometimes) *statically link* the libraries into the executable when compiling.

Design Choices — Statically-Linked Compilation

When using dynamically linked executables, the libraries for the executable must be available at run time. This is why you can't just take an executable from one distro and run it on another — if the libraries don't match what it was compiled against, it won't work. Statically linked executables can, however, run just about anywhere irrelevant of libraries (of course, one still needs the same kernel, CPU instruction set, etc)

Design Choices — Statically-Linked Compilation

In order to ensure the following items, Bedrock's core components are all statically linked:

- Run a core Bedrock executable directly in any of the client distros without worrying about chroot.
- Compile a Bedrock core component in any distro and simply dump it in place to update the component.

Note that client distros may freely use dynamically linked executables; this is only important for core Bedrock components.

Design Choices — Statically-Linked Compilation

It should be noted that statically linked compiling is frowned upon by many very people who are knowledgeable on the subject.

"Static linking is emphatically discouraged for all Red Hat Enterprise Linux releases. Static linking causes far more problems than it solves, and should be avoided at all costs."

https://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Developer_Guide/lib.compatibility.static.html

"Conclusion: Never use static linking!"

http://www.akkadia.org/drepper/no_static_linking.html

The Bedrock developer believes that Bedrock's unique situation creates a justifiable exemption, but do your own research. Another distro-in-progress, stali from suckless, also makes heavy use of static compilation: http://dl.suckless.org/stali/clt2010/stali.html

Design Choices — Manual Client Init Scripts

Most Linux distributions automatically manage the programs which are run at startup and shutdown. It is quite possible (and, in fact, likely) that multiple client distros will have startup and shutdown scripts which conflict with those from other distros. Moreover, there are a variety of Linux init systems, each of which have their own system for ensuring the programs are launched in the proper order to meet their prerequisites.

Design Choices — Manual Client Init Scripts

The Bedrock developer has been unable to think of any sane way of determining which init script to run when the client distros conflict (which CUPS daemon should run, if multiple are available?). Additionally, an automated way to determine the launch order from all of the possible systems it will run into seems far to challenging of a project.

Thus, Bedrock requires manually setting which programs from which distro's init is launched when.

Design Choices — Self-Sufficient Booting

The Bedrock developer feels strongly that

- Bedrock should be able to boot and do (very) basic tasks without any client distros.
- Bedrock should be able to boot even if the client distros unexpectedly break.

This means that if one would like a client distro to do something required when booting (for example, manage /dev)

1. Bedrock will everything else essential for booting
2. Bedrock will later, after the essentials are done and the system is functional, stop and let a client distro take over.

Package choices — Kernel: Linux

No other OS kernel has such a variety of userland options which could benefit from Bedrock's approach.

Package choices — Bootloader: Syslinux

This is the simplest bootloader the Bedrock Developer knows of.
Setting it up is just a handful of commands

Package choices — Userland: Busybox

Busybox is an all-in-one solution for a minimal(/embedded) Linux userland. It is significantly smaller and easier to set up than most of its alternatives. It is trivial to compile statically.

Package choices — Chroot: Capchroot

The standard chroot command requires root. If setuid'd it is quite possible to use chroot to escalate privileges. Thus, Bedrock requires a specialized chroot package intended to be used by non-root users. The typical choice for such things, schroot, was found to be too, large, complicated, and difficult to compile statically. Instead, Bedrock uses a little-known program called *capchroot*. This still requires some patches to be compiled statically linked, but is otherwise ideal.

Package choices — Shell Scripts

Additionally, Bedrock uses some of its own shell scripts (using busybox's /bin/sh) for things such as booting and integrating the system. Since busybox was already chosen, using its shell scripting option was an obvious choice.

Bedrock Scripts — brc

`brc` is a front-end for capchroot and is the main way users will manually run commands from other client distros. For example

```
brc fedora firefox
```

Will run Fedora's firefox, even if the `firefox` command would normally default to another distro's firefox. It will detect when preparation for setting up a distro client is needed, and automatically default to running the shell if no arguments are given.

Bedrock Scripts — brp

Early versions of Bedrock would detect if you tried to run a command which isn't available and, on the fly, attempt to find the command in a client distro. This proved to slow. Instead, Bedrock's `brp` command will hash the available commands in the client distros. This can take a few seconds.

Bedrock Scripts — brl

The `brl` command will run its argument in all available client distros. If, for example, you want to test to ensure that all of your distros have internet access:

```
brl ping -c 1 8.8.4.4
```

Bedrock Scripts — bru

Updating all of the client distros is a very common task, and so `bru` was created to make it a simple one. Running `bru` will update all client distros sequentially.

Early versions of Bedrock attempted to update all of the client distros simultaneously, but there were potential issues of multiple programs changing the same (shared) file simultaneously.

Bedrock Scripts — brsh

Due to its purposeful minimalism, the core Bedrock install only includes busybox's very limited shells; users will most likely want to use a client distro's shells by default. However, this raises two problems:

1. What if the user needs to log into bedrock's busybox's /bin/sh? For example, maybe the chroot system broke, or he/she is debugging a busybox update.
2. What if the chroot system is fine but the client distro breaks? What if the user forgets that he/she uses the distro client's shell and removes the distro?

Bedrock Scripts — brsh

Option 1:
Bedrock has its own meta-shell, `brsh`, which will log in to a configured distro client's shell, if available. If it is not available, it will automatically drop to Bedrock's /bin/sh.

Bedrock Scripts — brsh

Option 2:
The traditional Unix `/etc/passwd` allows creating multiple entries with different login names and different shells but same password, home, etc, for the same user.

```
root:x:0:0:root:/root:/opt/bedrock/bin/brsh
brroot:x:0:0:root:/root:/bin/sh
```

Current Issues — /etc/ File Syncing

In addition to ensuring directories like `/home` are shared between client distros, Bedrock must also ensure some files in `/etc` are shared. Specifically, user management files such as `passwd` and `shadow` need to be the same in the client distros. However, other files in `/etc`—such as `issue`—have to remain distinct, so not all of `/etc` can be shared; just the individual few files.

Bind mounting works wonderfully on directories where the *contents* of the directories are changed but the directory *itself* is not. Bind mounting also works great on files which are read-only, or edited, but not overwritten. However, issues arise when individual files are overwritten, and this happens a lot in `/etc`.

Current Issues — /etc/ File Syncing

In order to ensure important files such as `passwd` are never in a partially written state, `passwd` management programs copy it, alter the copy, then rename the new file over the old one. This way if the machine dies, one is almost always left with either the new copy or the old, and almost never a corrupt partially-written version.

Attempting to move a file onto a mount point, however, is forbidden, and results in errors.

Current Issues — /etc/ File Syncing

So bind mounts are out. Alternatives? Hard links are out because overwriting those removes the hard link—it would no longer be synced.

Symlinks *should* properly handle overwriting, but they don't break through chroots.

In theory, symlinking to a directory which is bind mounted should work. However, `passwd` management programs like GNU's adduser refuse to function on a symlink.

Current Issues — /etc/ File Syncing

Workaround:

- Manually manage syncing of the files

Plans to resolve this:

- Contact the GNU people to see why it refuses to act on symlinks.
- Find where the really, really good people are for this to ask—maybe LKML?

Current Issues — Argument Passing

There is currently an issue where arguments to some programs do not appear to be properly passed across client distros when they contain spaces. For example, if mplayer is in a different client distro from the one the shell is running in

```
$ mplayer Big\ Buck\ Bunney.ogv
```

makes mplayer thing it is getting three movies to present, not one with spaces. Should be fixable, just haven't gotten to it yet.

Current Issues — Delays with new client distro history

When a program in one client distro tries to run a program in another client distro, some preparation must be done. This only needs to be done once per distro history. For example:
Bedrock $\rightarrow$ Debian $\rightarrow$ Arch $\rightarrow$ Debian
Would require setup for each step of that history, but when none when launching another program with the same history.
This preparation takes some time (about half a second). Ideally, more of the setup should be moved to boot time rather than delayed until called.

Current Issues — Locale Support

Locale support is currently limited to timezones. Everything else is US-English-QWERTY.

If you know how to setup French or DVORAK, you are more than welcome to; however, Bedrock does not officially support either at the moment.

Current Issues — On-Battery Filesystem Checks

Most Linux distributions check to ensure one is not running on batter when running `fsck`. Bedrock does not currently have this check.

Goals For Future Releases — Package Manager Manager

A package manager manager.
Currently, managing packages in client distros chrooting to that distro and using the distros own package manager. This means a non-unified user interface is required for each distro. Sometimes apt-get, sometimes pacman, sometimes yum. It would be nice if a shell script was provided which wrapped around these programs, providing a unified interface.

Goals For Future Releases — Package Manager Manager

For example:

```
# pmm install arch firefox
```

Would call pacman in Arch Linux and install firefox.

```
# pmm install any sage-mathematics
```

Would install sage-mathematics in the first client distro it finds which has that package in its repos.

```
# pmm install newest libreoffice
```

Would look through all of the client distros for which has the newest copy of libreoffice in its repos and install that one.

Goals For Future Releases — File System Jump Warnings

Remember, while some aspects of the filesystem are shared between client distros, others have to be kept separate. This can lead to some confusion. For example:
Debian has vim installed. Arch does not. If the following command is run in Arch

```
# vim /etc/issue
```

It will edit *Debian*'s /etc/issue file rather than Arch's.
An (optional) system is planned to warn the user in the event of such situations.

Goals For Future Releases — Automatic Acquisition Of Client Distros

Currently, one must manually find a way to get a client distro on-disk. Plans are underway for a shell script which will automate this process. It will be, in essence, a wrapper around `debootstrap`, `febootstrap`, and/or `pacman`.

Goals For Future Releases — Proper Locale Support

Support for non US-English-QWERTY locales is planned.

Goals For Future Releases — Init System Rework

An minor overhaul of the init system is planned, with the following changes in mind:

- Parsing a config file for settings (such as whether time is stored in UTC or local, timezone, locale, etc) rather than hardcoding this into the init.
- Combining all of the various init scripts into a single init script
- Moving client distro init components into a separate file

Goals For Future Releases — Shared Subtrees

Typically, mounts do not propagate in bind mounts. For example, if you mount something into `/home`, then bind mount `/home` somewhere else (such as a client distro), the mount you made earlier in `/home` will not be in the new bind mount.

Shared subtrees, however, should propagate mounts in this manner. However, they do not seem to work recursively – if you make a shared bind mount inside of itself, it doesn't propagate recursively.

It may be possible to use shared subtrees instead of the bind mounts to improve efficiency, and it may be used in future versions of Bedrock.

First Release

Releasing a Linux distro is not as simple as tarballing the source with a makefile, throwing it up onto a file server, and handing people the URL.

One must either create an installer (which will not happen for Bedrock for quite some time), or create documentation detailing how one would go about installing the distro (think Linux From Scratch).

First Release

In order to do this properly, it should be done while installing Bedrock step-by-step rather than from memory. This requires taking down a computer for an extended period of time. The Bedrock developer has not had both the time necessary to install Bedrock from scratch as well as a spare computer to have down for an extended period of time.

Additionally, instructions on managing the unusual components of the system must also be made, such as instructions on how to set up client distros or make a CUPS server start at boot.

First Release

Bedrock development has, surprisingly, remained on schedule for the past two years. The first official release ("Appa") has been scheduled for the end of Summer 2012. Testing and feedback will be highly welcome.