Bedrock Linux

Daniel “paradigm” Thau
at
Ohio LinuxFest

2012-09-28

Bedrock Linux

Table of Contents

The Search for Perfect Distro
Bedrock-Only Features
Real-world Examples of where Bedrock Linux Shines
How It Works
Design Choices
Package choices
Bedrock Linux Scripts
Current Issues
Goals For Future Releases
Upcoming Release
For More Information

The Search for Perfect Distro

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

The Search for 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- – Out of date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 Search for Perfect Distro

So which should I chose?

The Search for Perfect Distro

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

The Search for Perfect Distro

I choose *everything*.

The Search for Perfect Distro

Since no existing distro provided *everything*, I made own:

Bedrock Linux.

The Search for Perfect Distro

Bedrock Linux 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 Linux 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 Linux 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 release can be still there, ready to go. You can easily fall back to the old release in its entirety, or if it is only a single package or two which broke in the update you can use all of the new release except the broken packages, which you can get from the prior release.

Bedrock-Only Features

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

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

Real-world Examples of where Bedrock Linux Shines

On several occasions, I wished to show off compiz. However:

- Debian's xorg is to old to support compiz on my newish laptop
- Arch's compiz seems broken

If I used Debian, I couldn't use compiz. If I used Arch, I couldn't use compiz. However, with Bedrock Linux, I can use Arch's xorg and Debian's compiz, and thus I can use compiz.

Real-world Examples of where Bedrock Linux Shines

My laptop's touchpad did not work in Debian 6, whereas it did in Debian 5. I simply used Debian 5's X11 (with everything else being Debain 6) until I learned about the proper solution.

Real-world Examples of where Bedrock Linux Shines

The only distribution of which I am aware with Sage mathematics in its repositories is Arch Linux. If I want to primarily use Fedora, but still get sage mathematics through a repository, Bedrock Linux lets me just grab it from Arch Linux's repositories.

Real-world Examples of where Bedrock Linux Shines

The game "Force: Leashed" is available pre-compiled against Ubuntu's libraries. If one is not on Ubuntu or Bedrock Linux, one will probably have to compile it. However, with Bedrock Linux, I can just run it as though I'm in Ubuntu and it just works.

This is particularly useful with proprietary software which one could not compile, such as the soon-coming-to-linux Steam platform. Valve is only—thus far—testing against Ubuntu. If you prefer, say, Gentoo, you can just use Gentoo in Bedrock Linux all the time, and when you are running steam just have Steam run against Ubuntu's libraries.

How It Works

Bedrock Linux'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 light-weight virtualization. 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 Linux 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 Linux. 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."

Or for another example, you could ensure you only have to maintain a single `/tmp` on Bedrock Linux. You could have a web browser from one distro save a PDF to `/tmp`, and have a PDF reader from another distro see it and read it from `/tmp` as it would normally.

How It Works — Bind Mounts

Through proper usage of chroots and bind mounts, Bedrock Linux 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 Linux 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 Linux prioritizes the "native" executables before searching through the other distros, and has a hard priority for the other distros. Future versions of Bedrock Linux 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 Linux's unusual goals, several unusual design choices were made. These choices were the reason Bedrock Linux's system requires its own distribution to be fully utilized rather than simply being grafted onto another distribution.

Design Choices — Simplicity

For reasons explained later on, Bedrock Linux must be installed from scratch, and must be maintained at a very low level (such as hand-editing init files).

In order to ensure Bedrock Linux 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 Linux 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 Linux itself. If something can be deferred to a client distro, it will be; Bedrock Linux 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 Linux's core components are all statically linked:

- Run a core Bedrock Linux executable directly in any of the client distros without worrying about chroot.
- Compile a Bedrock Linux 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 Linux 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 Linux developer believes that Bedrock Linux'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 Linux 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 Linux requires manually setting which programs from which distro's init is launched when.

Design Choices — Self-Sufficient Booting

The Bedrock Linuxdeveloper feels strongly that

- Bedrock Linux should be able to boot and do (very) basic tasks without any client distros.
- Bedrock Linux 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 Linux will everything else essential for booting
2. Bedrock Linux 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 Linux's approach.

Package choices — Bootloader: Syslinux

This is the simplest bootloader the Bedrock Linux 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 relatively easy 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 Linux 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 Linux 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 Linux 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 Linux 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 Linux Scripts — brp

Early versions of Bedrock Linux 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 Linux's `brp` command will hash the available commands in the client distros. This can take a few seconds.

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

Bedrock Linux Scripts — brsh

Due to its purposeful minimalism, the core Bedrock Linux 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 Linux Scripts — brsh

Option 1:
Bedrock Linux 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 Linux's /bin/sh.

Bedrock Linux 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 Linux 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

Potential solutions:

- Manually manage syncing of the files
- Perhaps automate `cat /etc/group- > /etc/group`
- Have Bedrock Linux come with a giant `/etc/group` that contains all of the groups which could be installed by client Linux distributions automatically.
- Perhaps something such as UnionFS

Current Issues — Difficulties statically-compiling Busybox

Currently, installing Bedrock Linux requires manually compiling a statically-linked Busybox. A number of people have reported difficulties with this.

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.

Upcoming Release

The upcoming release is 1.0alpha3 "Bosco", aimed at mid 2013.

For More Information

- bedrocklinux.org
- #bedrock on freenode
- reddit.com/r/bedrocklinux
- https://github.com/paradigm/bedrocklinux-website
- https://github.com/paradigm/bedrocklinux-userland