

# Booting Linux off of Google Drive

---

'n [ersei.net/en/blog/fuse-root](https://ersei.net/en/blog/fuse-root)

Published: [ **2024-07-01 09:20 EDT** ]

Categories: [ [programming](#) ]

Tags: [ [linux](#), [filesystems](#) ]

Competitiveness is a vice of mine. When I heard that a friend got Linux to boot off of NFS, I had to one-up her. I had to prove that I could create something *harder*, something *better*, *faster*, *stronger*.

Like all good projects, this began with an Idea.

My mind reached out and grabbed wispy tendrils from the æther, forcing the disparate concepts to coalesce. The Mass gained weight in my hands, and a dark, swirling colour promising doom to those who gazed into it for long.

On the brink of insanity, my tattered mind unable to comprehend the twisted interplay of millennia of arcane programmer-time and the ragged screech of madness, I reached into the Mass and steeled myself to the ground lest I be pulled in, and found my *magnum opus*.

Booting Linux off of a Google Drive root.

## But How?

---

I wanted this to remain self-contained, so I couldn't have a second machine act as a "helper". My mind went immediately to FUSE—a program that acts as a filesystem driver in userspace (with cooperation from the kernel).

I just had to get FUSE programs installed in the Linux kernel initramfs and configure networking. How bad could it be?

## The Linux Boot Process

---

The Linux boot process is, technically speaking, very funny. Allow me to pretend I understand for a moment<sup>1</sup>:

1. The firmware (BIOS/UEFI) starts up and loads the bootloader
2. The bootloader loads the kernel
3. The kernel unpacks a temporary filesystem into RAM which has the tools to mount the real filesystem
4. The kernel mounts the real filesystem and switches the process to the init system running on the new filesystem

As strange as the third step may seem, it's very helpful! We can mount a FUSE filesystem in that step and boot normally.

## A Proof of Concept

---

The initramfs needs to have both network support as well as the proper FUSE binaries. Thankfully, [Dracut](#) makes it easy enough to build a custom initramfs.

I decide to build this on top of Arch Linux because it's relatively lightweight and I'm familiar with how it works, as opposed to something like Alpine.

```
$ git clone https://github.com/dracutdevs/dracut
$ podman run -it --name arch -v ./dracut:/dracut docker.io/archlinux:latest bash
```

In the container, I installed some packages (including the `linux` package because I need a functioning kernel), compiled `dracut` from source, and wrote a simple module script in `modules.d/90fuse/module-setup.sh`:

```
#!/bin/bash
check() {
    require_binaries fusermount fuseiso mkisofs || return 1
    return 0
}

depends() {
    return 0
}

install() {
    inst_multiple fusermount fuseiso mkisofs
    return 0
}
```

That's it. That's all the code I had to write. Buoyed by my newfound confidence, I powered ahead, building the EFI image.

```
$ ./dracut.sh --kver 6.9.6-arch1-1 \  
  --uefi efi_firmware/EFI/BOOT/BOOTX64.efi \  
  --force -l -N --no-hostonly-cmdline \  
  --modules "base bash fuse shutdown network" \  
  --add-drivers "target_core_mod target_core_file e1000" \  
  --kernel-cmdline "ip=dhcp rd.shell=1 console=ttyS0"  
$ qemu-kvm -bios ./FV/OVMF.fd -m 4G \  
  -drive format=raw,file=fat:rw:./efi_firmware \  
  -netdev user,id=network0 -device e1000,netdev=network0 -nographic
```

```
...  
...
```

```
dracut Warning: dracut: FATAL: No or empty root= argument  
dracut Warning: dracut: Refusing to continue
```

Generating "/run/initramfs/rdsosreport.txt"  
You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot after mounting them and attach it to a bug report.

To get more debug information in the report,  
reboot with "rd.debug" added to the kernel command line.

Dropping to debug shell.

```
dracut:/#
```

*Hacker voice* I'm in. Now to enable networking and mount a test root. I have already extracted an Arch Linux root into a S3 bucket running locally, so this should be pretty easy, right? I just have to manually set up networking routes and load the drivers.

```
dracut:/# modprobe fuse  
dracut:/# modprobe e1000  
dracut:/# ip link set lo up  
dracut:/# ip link set eth0 up  
dracut:/# dhclient eth0  
dhcp: PREINIT eth0 up  
dhcp: BOUND setting up eth0  
dracut:/# ip route add default via 10.0.2.2 dev eth0 proto dhcp src 10.0.2.15  
dracut:/# s3fs -o url=http://192.168.2.209:9000 -o use_path_request_style fuse  
/sysroot  
dracut:/# ls /sysroot  
bin  dev  home  lib64  opt  root  sbin  sys  usr  
boot  etc  lib  mnt  proc  run  srv  tmp  var  
dracut:/# switch_root /sysroot /sbin/init  
switch_root: failed to execute /lib/systemd/systemd: Input/output error  
dracut:/# ls  
sh: ls: command not found
```

Honestly, I don't know what I expected. Seems like everything is just... *gone*. Alas, not even tab completion can save me. At this point, I was stuck. I had no idea what to do. I spent days just looking around, poking at the `switch_root` source code, all for naught. Until I

remembered a link [Anthony](#) had sent me: [How to shrink root filesystem without booting a livecd](#). In there, there was a command called `pivot_root` that `switch_root` seems to call internally. Let's try that out.

```
dracut:/# logout
...
[ 430.817269] ---[ end Kernel panic - not syncing: Attempted to kill init!
exitcode=0x00000100 ]---
...
dracut:/# cd /sysroot
dracut:/sysroot# mkdir oldroot
dracut:/sysroot# pivot_root . oldroot
pivot_root: failed to change root from `.` to `oldroot': Invalid argument
```

Apparently, `pivot_root` is not allowed to pivot roots if the root being switched is in the `initramfs`. Unfortunate. The Stack Exchange answer tells me to use `switch_root`, which doesn't work either. However, part of that answer sticks out to me:

```
initramfs is rootfs: you can neither pivot_root rootfs, nor unmount it. Instead delete
everything out of rootfs to free up the space (find -xdev / -exec rm '{}' ';'), overmount
rootfs with the new root (cd /newmount; mount --move . /; chroot .), attach
stdin/stdout/stderr to the new /dev/console, and exec the new init.
```

Would it be possible to manually switch the root *without* a specialized system call? What if I just `chroot`?

```
...
dracut:/# mount --rbind /sys /sysroot/sys
dracut:/# mount --rbind /dev /sysroot/dev
dracut:/# mount -t proc /proc /sysroot/proc
dracut:/# chroot /sysroot /sbin/init
Explicit --user argument required to run as user manager.
```

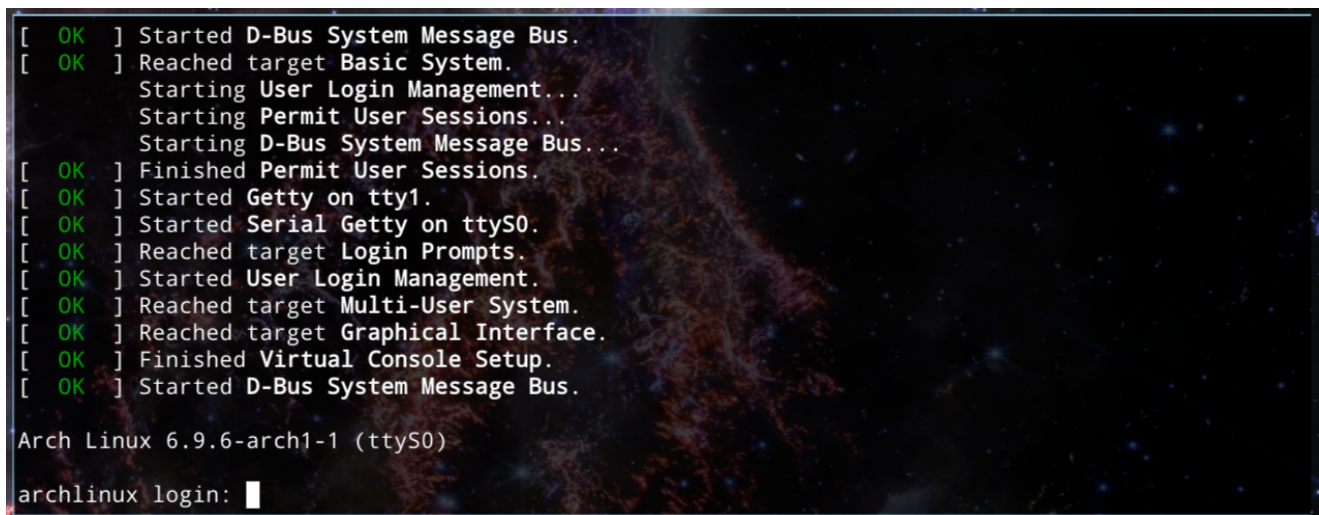
Oh, I need to run the `chroot` command as PID 1 so `Systemd` can start up properly. I can actually tweak the `initramfs`'s `init` script and just put my startup commands in there, and replace the `switch_root` call with `exec chroot /sbin/init`.

I put this in `modules.d/99base/init.sh` in the Dracut source after the `udev` rules are loaded and bypassed the `root` variable checks earlier.

```
modprobe fuse
modprobe e1000
ip link set lo up
ip link set eth0 up
dhclient eth0
ip route add default via 10.0.2.2 dev eth0 proto dhcp src 10.0.2.15
s3fs -o url=http://192.168.2.209:9000 -o use_path_request_style fuse /sysroot
mount --rbind /sys /sysroot/sys
mount --rbind /dev /sysroot/dev
mount -t proc /proc /sysroot/proc
```

I also added `exec chroot /sysroot /sbin/init` at the end instead of the `switch_root` command.

Rebuilding the EFI image and...

A screenshot of a terminal window showing the boot process of Arch Linux. The background is a dark, starry space image. The text is white and green. It shows the system starting the D-Bus System Message Bus, reaching the Basic System target, starting User Login Management, Permit User Sessions, and Serial Getty on ttyS0. It then reaches the Login Prompts target, starts User Login Management, reaches the Multi-User System target, reaches the Graphical Interface target, finishes Virtual Console Setup, and starts the D-Bus System Message Bus again. The prompt is 'Arch Linux 6.9.6-arch1-1 (ttyS0)' and the login prompt is 'archlinux login:'.

```
[ OK ] Started D-Bus System Message Bus.
[ OK ] Reached target Basic System.
Starting User Login Management...
Starting Permit User Sessions...
Starting D-Bus System Message Bus...
[ OK ] Finished Permit User Sessions.
[ OK ] Started Getty on tty1.
[ OK ] Started Serial Getty on ttyS0.
[ OK ] Reached target Login Prompts.
[ OK ] Started User Login Management.
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.
[ OK ] Finished Virtual Console Setup.
[ OK ] Started D-Bus System Message Bus.

Arch Linux 6.9.6-arch1-1 (ttyS0)
archlinux login: █
```

I sit there, in front of my computer, staring. It can't have been that easy, can it? Surely, this is a profane act, and the spirit of Dennis Ritchie ought't've stopped me, right?

Nobody stopped me, so I kept going.

I log in with the very secure password `root` as `root`, and it unceremoniously drops me into a shell.

```
[root@archlinux ~]# mount
s3fs on / type fuse.s3fs (rw,nosuid,nodev,relatime,user_id=0,group_id=0)
...
[root@archlinux ~]#
```

At last, Linux booted off of an S3 bucket. I was compelled to share my achievement with others—all I needed was a fetch program to include in the screenshot:

```
[root@archlinux ~]# pacman -Sy fastfetch
:: Synchronizing package databases...
   core.db failed to download
error: failed retrieving file 'core.db' from geo.mirror.pkgbuild.com : Could not
resolve host: geo.mirror.pkgbuild.com
warning: fatal error from geo.mirror.pkgbuild.com, skipping for the remainder of this
transaction
error: failed retrieving file 'core.db' from mirror.rackspace.com : Could not resolve
host: mirror.rackspace.com
warning: fatal error from mirror.rackspace.com, skipping for the remainder of this
transaction
error: failed retrieving file 'core.db' from mirror.leaseweb.net : Could not resolve
host: mirror.leaseweb.net
warning: fatal error from mirror.leaseweb.net, skipping for the remainder of this
transaction
error: failed to synchronize all databases (invalid url for server)
[root@archlinux ~]#
```

Uh, seems like DNS isn't working, and I'm missing `dig` and other debugging tools.

Wait a minute! My root filesystem is on S3! I can just mount it somewhere else with functional networking, `chroot` in, and install all my utilities!

Some debugging later, it seems like `systemd-resolved` doesn't want to run because it `Failed to connect stdout to the journal socket, ignoring: Permission denied`. I'm not about to try to debug `systemd` because it's too complicated and I'm lazy, so instead I'll just use Cloudflare's.

```
[root@archlinux ~]# echo "nameserver 1.1.1.1" > /etc/resolv.conf
[root@archlinux ~]# pacman -Sy fastfetch
:: Synchronizing package databases...
   core is up to date
   extra is up to date
...
[root@archlinux ~]# fastfetch
```

```
[root@archlinux ~]# fastfetch
root@archlinux
-----
OS: Arch Linux x86_64
Host: KVM/QEMU Standard PC (i440FX + PIIX, 1996) (pc-i440fx-9.0)
Kernel: Linux 6.9.6-arch1-1
Uptime: 2 mins
Packages: 128 (pacman)
Shell: bash 5.2.26
Display (QEMU Monitor): 1280x800 @ 75Hz
Terminal: vt220
CPU: QEMU Virtual version 2.5+ @ 2.92 GHz
GPU: Unknown Device 1111 (VGA compatible)
Memory: 425.07 MiB / 3.82 GiB (11%)
Swap: Disabled
Disk (/): 0 B / 3.98 GiB (0%) - fuse.s3fs
Local IP (ens3): 10.0.2.15/24 *
Locale: C.UTF-8

root@archlinux
-----
OS: Arch Linux x86_64
Host: KVM/QEMU Standard PC (i440FX + PIIX, 1996) (pc-i440fx-9.0)
Kernel: Linux 6.9.6-arch1-1
Uptime: 2 mins
Packages: 128 (pacman)
Shell: bash 5.2.26
Display (QEMU Monitor): 1280x800 @ 75Hz
Terminal: vt220
CPU: QEMU Virtual version 2.5+ @ 2.92 GHz
GPU: Unknown Device 1111 (VGA compatible)
Memory: 425.07 MiB / 3.82 GiB (11%)
Swap: Disabled
Disk (/): 0 B / 3.98 GiB (0%) - fuse.s3fs
Local IP (ens3): 10.0.2.15/24 *
Locale: C.UTF-8

[root@archlinux ~]# mount
s3fs on / type fuse.s3fs (rw,nosuid,nodev,relatime,user_id=0,group_id=0)
```

I look around, making sure that nobody had tried to stop me. My window was intact, my security system had not tripped, the various canaries I had set up around the house had not been touched. I was safe to continue.

I was ready to have it run on Google Drive.

## Google Gets Involved

There's a project already that does Google Drive over FUSE for me already: [google-drive-ocamlfuse](https://github.com/ocamlfuse/google-drive-ocamlfuse). Thankfully, I have a Google account lying around that I haven't touched in years ready to go! I follow the instructions, accept the terms of service I didn't read, create all the oauth2 secrets, enable the APIs, install `google-drive-ocamlfuse` from the AUR into my Arch Linux VM, patch some `PKGBUILDs` (it's been a while), and lo and behold! I have mounted Google Drive! Mounting Drive and a few *very long* `rsync` runs later, I have Arch Linux on Google Drive.

Just kidding, it's never that easy. Here's a non-exhaustive list of problems I ran into:

1. Symlinks to symlinks don't work (very important for stuff in `/usr/lib`)
2. Hardlinks don't work
3. It's so slowwww
4. Relative symlinks don't work at all
5. No dangling symlinks (important for stuff that links to `/proc` and isn't mounted, or stuff that just hasn't copied over yet)
6. Symlinks outside of Google Drive don't work
7. Permissions don't work (neither do attributes)
8. Did I mention it's SLOW

With how many problems there are with symlinks, I have half a mind to change the FUSE driver code to just create a file that ends in `.internalsymlink` to fix all of that, Google Drive compatibility be damned.

But, I have challenged myself to do this without modifying anything important (no kernel tweaking, no FUSE driver tweaking), so I'll just have to live with it and manually create the symlinks that `rsync` fails to make with a hacky `sed` command to the `rsync` error logs.

In the meantime, I added the token files generated from my laptop into the `initramfs`, as well as the Google Drive FUSE binary and SSL certificates, and tweaked a few settings<sup>2</sup> to make my life slightly easier.

```
...
inst ./gdfuse-config /.gdfuse/default/config
inst ./gdfuse-state /.gdfuse/default/state
find /etc/ssl -type f -or -type l | while read file; do inst "$file"; done
find /etc/ca-certificates -type f -or -type l | while read file; do inst "$file";
done
...
```

My Drive > fuse-root ▾



Type ▾ People ▾ Modified ▾

Name ↑	Owner	Last mo... ▾	File size	⋮
bin				⋮
boot	me	Apr 7, 2024	—	⋮
dev	me	Jun 25, 2024	—	⋮
etc	me	Jun 25, 2024	—	⋮
home	me	Apr 7, 2024	—	⋮

It's nice to see that timestamps kinda work, at least. Now all that's left is to wait for the agonizingly slow boot!

```
chroot: /sbin/init: File not found
```

Perhaps they did not bother to stop me because they knew I would fail.

I know the file exists since, well, it *exists*, so why is it not found? Simple: Linux is kinda weird and if the binary you call depends on a library that's not found, then you'll get "File not found".



```
dracut:/# ldd /sysroot/bin/bash
linux-vdso.so.1 (0x00007e122b196000)
libreadline.so.8 => /usr/lib/libreadline.so.8 (0x00007e122b01a000)
libc.so.6 => /usr/lib/libc.so.6 (0x00007e122ae2e000)
libncursesw.so.6 => /usr/lib/libncursesw.so.6 (0x00007e122adbf000)
/lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2
(0x00007e122b198000)
```

However, these symlinks don't actually exist! Remember how earlier we noted that relative symlinks don't work? Well, that's come back to bite me. The Kernel is looking for files in `/sysroot` inside `/sysroot/sysroot`. Luckily, this is an easy enough fix: we just need to have `/sysroot` linked to `/sysroot/sysroot` without links:

```
dracut:/# mkdir /sysroot/sysroot
dracut:/# mount --rbind /sysroot /sysroot/sysroot
```

Now time to boot!

It took five minutes for Arch to rebuild the dynamic linker cache, another minute per systemd unit, and then, nothing. The startup halted in its tracks.

```
[ TIME ] Timed out waiting for device /dev/ttyS0.
[DEPEND] Dependency failed for Serial Getty on ttyS0.
```

Guess I have to increase the timeout and reboot. In `/etc/systemd/system/dev-ttyS0.device`, I put:

```
[Unit]
Description=Serial device ttyS0
DefaultDependencies=no
Before=sysinit.target
JobTimeoutSec=infinity
```

Luckily, it did not take infinite time to boot.

```

[ OK ] Listening on GnuPG cryptographic a... browsers) for /etc/pacman.d/gnupg.
[ OK ] Listening on GnuPG cryptographic a...estricted) for /etc/pacman.d/gnupg.
[ OK ] Listening on GnuPG cryptographic a...emulation) for /etc/pacman.d/gnupg.
[ OK ] Listening on GnuPG cryptographic a...rase cache for /etc/pacman.d/gnupg.
[ OK ] Listening on GnuPG public key mana...nt service for /etc/pacman.d/gnupg.
[ OK ] Listening on Hostname Service Socket.
[ OK ] Reached target Socket Units.
[ OK ] Reached target Basic System.
[ OK ] Started DNSCrypt-proxy client.
[ OK ] Reached target Host and Network Name Lookups.
Starting User Login Management...
Starting Permit User Sessions...
[ OK ] Finished Permit User Sessions.
[ OK ] Started Getty on tty1.
[ OK ] Started Serial Getty on ttyS0.
[ OK ] Reached target Login Prompts.
Starting D-Bus System Message Bus...
[ OK ] Started User Login Management.
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.

Arch Linux 6.9.6-arch1-1 (ttyS0)

archlinux login: root
login: timed out after 60 seconds
Arch Linux 6.9.6-arch1-1 (ttyS0)

archlinux login: █

```

I'm so close to victory I can *taste* it! I just have to increase *another* timeout. I set `LOGIN_TIMEOUT` to `0` in `/etc/login.defs` in Google Drive, and tried logging in again.

Thankfully, there's a cache, so subsequent file reads aren't nearly as slow.

```

[root@archlinux ~]# fastfetch
      .o+
     /ooo/
    +oooo:
   +oooooo:
  +oooooo+:
 /:-!+oooo+:
/++++/+++++++:
/+++++++:
/+++++oooooooooo+/
./ooooo+oooo+
.ooooo-`-`-/oooo+
-oooooo.   :oooooo.
:oooooo/   ooooo++
/oooooo/   +ooooo/-
/oooooo+/- -:/+oooo+
+so+:-     .-/oso:
++:..      -/+
      /

      root@archlinux
      -----
      OS: Arch Linux x86_64
      Host: KVM/QEMU Standard PC (i440FX + PIIX, 1996) (pc-i440fx-9.0)
      Kernel: Linux 6.9.6-arch1-1
      Uptime: 10 mins
      Shell: bash 5.2.26
      Display (QEMU Monitor): 1280x800 @ 75Hz
      Terminal: vt220
      CPU: QEMU Virtual version 2.5+ @ 2.92 GHz
      GPU: Unknown Device 1111 (VGA compatible)
      Memory: 492.92 MiB / 3.82 GiB (13%)
      Swap: Disabled
      Disk (/): 2.71 GiB / 15.00 GiB (18%) - fuse.google-drive-ocamlfuse
      Local IP (ens3): 10.0.2.15/24 *
      Locale: C.UTF-8

[root@archlinux ~]# mount
google-drive-ocamlfuse on / type fuse.google-drive-ocamlfuse (rw,nosuid,nodev,relatime,user_id=0,group_id=0)

```

Here I am, laurel crown perched upon my head, my chimera of Linux and Google Drive lurching around.

But I'm not satisfied yet. Nobody had stopped me because they *want* me to succeed. I have to take this further. I need this to work on *real hardware*.

## Now Do It On Real Hardware

Fortunately for me, I switched servers and now have an extra laptop with no storage just lying around! A wonderful victim<sup>3</sup> for my test!

There are a few changes I'll have to make:

1. Use the right ethernet driver and not the default `e1000`
2. Do not use a serial display
3. Change the network settings to match my house's network topology

All I need is the `r8169` driver for my ethernet port, and let's throw in a Powerline into the mix, because it's not going to impact the performance in any way that matters, and I don't have an ethernet cord that can reach my room.

I build the unified EFI file, throw it on a USB drive under `/BOOT/EFI`, and stick it in my old server. Despite my best attempts, I couldn't figure out what the modprobe directive is for the laptop's built-in keyboard, so I just modprobed `hid_usb` and used an external keyboard to set up networking.

```
Arch Linux 6.9.6-arch1-1 (tty1)
archlinux login: root
Password:
Last login: Sun Jun 30 17:34:45 on ttyS0
[root@archlinux ~]# fastfetch

      .o+
     /oo/
    +oooo:
   +oooooo:
  +ooooooo+:
 /:-:++oooo+:
/++++/+++++++:
/++++/+++++++:
/+++ooooooooooo/
.ooooosso+ooooosso+
.ooooosso-`ooosssso+
-ooooosso. :ssssso.
:ooooosso/  oosso+++
/ooooosso/  +sssooo/-
/ooooosso+/- -:/+oooo+
+ss+:`-      .-/+oso+
++:         .-/+

root@archlinux
-----
OS: Arch Linux x86_64
Host: Inspiron 5547 (A10)
Kernel: Linux 6.9.6-arch1-1
Uptime: 9 mins
Shell: bash 5.2.26
Display (AUO10ED): 1920x1080 @ 60Hz
Terminal: /dev/tty1
Terminal Font: UGA default kernel font
CPU: Intel(R) Core(TM) i5-4210U (4) @ 2.70 GHz
GPU: Intel Haswell-ULT Integrated Graphics Controller @ 1.00 GHz [Integrated]
Memory: 587.88 MiB / 11.57 GiB (5%)
Swap: Disabled
Disk (/): 2.71 GiB / 15.00 GiB (18%) - fuse.google-drive-ocamlfuse
Local IP (enp1s0): 192.168.2.251/24 *
Battery: 59% [AC Connected]
Locale: C.UTF-8

[root@archlinux ~]# mount
google-drive-ocamlfuse on / type fuse.google-drive-ocamlfuse (rw,nosuid,nodev,relatime,user_id=0,group_id=0)
```

This is my *magnum opus*. My Great Work. This is the mark I will leave on this planet long after I am gone: The Cloud Native Computer.

Nice thing is, I can just grab the screenshot<sup>4</sup> from Google Drive and put it here!

## Woe! Cloud Native Computer Be Upon Ye

Despite how silly this project is, there are a few less-silly uses I can think of, like booting Linux off of SSH, or perhaps booting Linux off of a Git repository and tracking every change in Git using gitfs. The possibilities are endless, despite the middling usefulness.

If there is anything I know about technology, it's that moving everything to The Cloud is the current trend. As such, I am prepared to commercialize this for any company wishing to leave their unreliable hardware storage behind and move entirely to The Cloud. Please [request a quote](#) if you are interested in True Cloud Native Computing.

Unfortunately, I don't know what to do next with this. Maybe I should install Nix?

---

Thoughts? Comments? Opinions? Feel free to share (relevant) ones with me! [Contact me here if you want.](#)

---

1. I understand mostly because I read [this Archwiki article](#). This section ends up being a wispy summarization. [↵](#)
2. I set `acknowledge_abuse=true`, and `root_folder=fuse-root`. [↵](#)
3. No computers were (physically) harmed in the making of this project. [↵](#)
4. I used [fbgrab](#) to take the screenshot [↵](#)