dual-root

Release 3.0.0

Gene C

Oct 17, 2024

CONTENTS:

AKA Hot Spare Bootable Disk

CHAPTER

ONE

DUAL ROOT CAPABLE LINUX SYSTEM

1.1 AKA hot spare bootable root disk

1.2 NEW

October 17 2024:

- Performance improvements:

  There are times when the sync daemon was using too much cpu/io. This release improves on that.

- sync daemon now defaults to running with: nice = 15 ionice class = IDLE
- Rsync requests triggered by inotify are now placed on a queue and run in a separate thread. Queue is only run if sync_delay seconds have passsed since the last run and there is no currently running sync. Each queue is checked for pending sync requests periodicallyu (15 mins) and and pending requests are handled by the queue runner. On exit a final queue check is run. Each directory being monitored by inotify has a separate queue. The sync_delay determines the minimum time between queue runs and defaults to 300 seconds.
- These can all be changed in the sync-daemon.conf using variables: nice, ionice_class and ionice_value for the realtime/best effort classes. sync_delay - is the minimum number of seconds before any new queue run is done.

1.3 Goal

The goal is a system which is resilient to a failure of the root drive. In the event of a root disk failure, the alternate root disk will be automaticlly deployed. Ordinarily the UEFI bios will boot the alternate disk without any user interaction should the one of the 2 disks become unavailable.

If for some reason, that doesn't happen, then one can always use the bios boot menu, the one that lets user choose which drive to boot, to choose which root to boot; under normal circumstances either drive can be booted from this menu.

Since the system can function normally regardless of which disk is booted, then even after a disk failure, once the system is rebooted, things will continue to function completely normally with all services operational.

There are different ways to achieve this but they all share one key aspect which is having two drives with each drive having its own <*esp*>.

We outline two approaches here, the first, and preferable, choice works well when doing a fresh install and the second approach works when adding to an existing system where the goal is to keep machine running and not start over with new install. Second approach may also be chosen if using 1 nvme and 1 spinner for the 2 disks. Using raid in this case will likely lose much of the nvme speed advantage.

There are other possibilities but some may be risky and hacky in nature. We advocate systems that are robust, clean and transparent.

The methods outlined here will have some partitions which need to be kept synchronized, at a minimum this includes the <esp>. Other partitions will either be protected by RAID (RAID-1 or higher) or will be kept synchronized. Synchronizing is best limited to those areas which are stable rather than things that constantly changing, such as mail or databases. These *dynamic* areas should, if at all possible, be protected by RAID.

Since the requirement is to be able to boot either disk, then neither disk used for booting purposes can have any hard dependency on the other disk. That means each disk must have its own *<esp>* and provide the same key partitions holding the operating system.

The 2 approaches outlined here both use:

- 2 disks
- each disk has its own <esp> partition.
- <esp>s are kept in sync with each other.
- no constraints on disk other than they each have sufficient capacity.
- using systemd boot
  - Assume refind or grub would work too.

We provide *dual-boot-tool* and systemd services to make it as straightforward as possible to implement. The tool can identify the currently booted <esp>, bind mount it to */boot* and perform various sync operations either one off or as a daemon using inotify to detect changes in the filessystem.

1.4 First Approach Summary[1] :

2 disks each with 2 required partitions: <esp> and root. The 2 <esp>s are autonaticaly syncronized, and the 2 root partitions are joined using raid1

- Best Approach when possible[2]
- Best suited :
  - fresh installs
- sync list:
  - <esp>s
- kernel and initrd on <esp>
- Both <esp>s use same loader configs
- Everything else is mirrored using btrfs raid1
- With SSD, best not to mix SSD and spinner for RAID (lose speed benefit)
- included systemd service
  - mounts currently booted <esp> onto /boot
- included systemd service
  - identifies booted <esp> and sync's other <esp>s from current <esp>
- new install takes longer
  - use backup of existing root drive to minimize downtime

[1] As discussed on Arch General Mail List with thanks to Óscar Amor for the basic idea.
[2] See Lennart Poettering's Blog "Linux Boot Partitions"[4]
[4] https://0pointer.net/blog/

For those who prefer to keep their kernels on a linux filesystem, it is easy enough to use a separate /boot partition of type XBOODLDR. These would need to be added to sync config to be included.

Aside: I have not tested this, but it may be possible to bind the 2 boot partitions with a btrfs raid1. In the usual case, the loader finds the XBOOTLDR partition by looking on the same disk as the esp. With raid spanning the 2 disks, each of type XBOOTLDR it may or may not work - so untested. It is certainly simpler to leave them all on the <esp>

1.5 Second Approach Summary:

2 disks each with <esp> and root patition(s). The 2 <esp>s are kept synchroniuzed and each directory on the "root" partition is also kept synchronized.

- best suited :
  - upgrade existing system with minimal changes
  - using 1 SSD + 1 spinner, and keeping primary boot from the fast SSD
- sync list:
  - <esp>, boot, root, usr and possibly var
  - dynamic areas (e.g. var) should preferably be on a RAID array. Especially if there are things like mail or databses running. What I do is keep these on separate RAID-6 and bind mount them into var
- Short Downtime only to install 2nd disk. Configure while running normally.
- Each disk has its own root UUID and thus different boot loader config UUIDs - so these need to be excluded from sync. Likewise there are now 1 fstab on each drive, and this too needs to be excluded.

We use Archlinux but the distro shouldn't play any significant role in dual root setup. We find the Arch rolling release distro convenient and robust.

One of the beautiful things about linux is that, more often than not, there is more than one way to do things. And here are two ways :)

CHAPTER

TWO

FIRST APPROACH - DETAILS

Each of the two disks to be used needs its own <esp> and root partitions. The currently booted <esp> will be mounted as /boot. Actually the <esp>'s are all mounted as /efi0, /efi1, etc. And whichever is currently booted is then bind mounted to /boot.

Make the <esp> partitions each the same size - 1 - 2 GB provides plenty of room for multiple kernels. While btrfs raid mirror doesn't require equal sized partitions, if the disks are different sizes, then there will be unused space. Ignore it or make the 2 roots the same size, and create an extra partition on the larger one. That extra partition will not be part of the raid1 obviously.

Can also be a swap partition if desired, but it plays no direct role here.

If converting an existing setup, then backup everything either to another disk, external or internal or over the network to another computer. Otherwise we assume starting with fresh install.

This has one tricky part to sort out, which is that we have one root but 2 esp partitions. After the machine boots we will mount both <esp> partitions, and we need to know which one was used to boot so that we can sync it to the other one. We'll explain how to do that in a robust way a little later.

2.1 Partition sizing:

For example, if we use 2 GB <esp> partition and the root partition be rest of disk. In this example the <esp> as on sda1 / sdb1, swap partitions are sda2 / sdb2 and the root partitions are on sda3 / sdb3.

We are now ready to put filesystems on the disks. First format the <esp> partitions:

```
mkfs.vfat -n EFI0 /dev/sda1
mkfs.vfat -n EFI1 /dev/sdb1
```

Each gets its own swap in this example:

```
mkswap -L swap0 /dev/sda2
mkswap -L swap1 /dev/sdb2
```

And then the root filesystems:

```
mkfs.btrfs -L root -m raid1 -d raid1 /dev/sda2 /dev/sdb2
```

In this example the first disk is larger than the second, so we use the extra space to create a *data* partition.

Lets look at what we have and identify the UUIDs we'll need as well:

```
# lsblk -f
lsblk -f
```

(continues on next page)

(continued from previous page)

```
NAME   FSTYPE FSVER LABEL UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
sda
├─sda1 vfat   FAT32 EFI0  6B7E-A837
├─sda2 swap   1     swap0 285c7969-f137-4b3e-b89e-fabe81e44eb1
├─sda3 btrfs        root  a8426465-b755-429d-9604-9c77c2838fda
└─sda4 ext4   1.0   data0 315025e3-26a7-4d3e-a3af-cfb8f7cea339
sdb
├─sdb1 vfat   FAT32 EFI1  6C48-1623
├─sdb2 swap   1     swap1 3651f9e6-85a1-464d-ac70-74d3d085f577
└─sdb3 btrfs        root  a8426465-b755-429d-9604-9c77c2838fda
```

To continue we'll use temporary mounts:

```
mkdir -p /mnt/root
mount UUID=a8426465-b755-429d-9604-9c77c2838fda /mnt/root

cd /mnt/root
mkdir -p boot data dev efi etc home mnt opt proc root run srv sys usr var tmp

mkdir /mnt/root/efi0 /mnt/root/efi1
mount /dev/sda1 /mnt/root/efi0
mount /dev/sdb1 /mnt/root/efi1
mount --bind /mnt/root/efi0 /mnt/root/boot
```

At this point either use arch-chroot and install as usual or rsync from an appropriate backup. With this set up the efi is then bind mounted onto /boot. For our example we bind mount efi0 onto /boot.

We will always mount both <esp> partitions under /efi0 and /efi1. In addition we bind mount one of them onto /boot for convenience. The goal is to have the currently booted <esp> bind mounted onto /boot - which is the standard place for kernels and initrds to be installed.

If you're pulling from a backup then regenerate all initrds to be sure they are consistent with the current set up. Don't skip this step :)

Make sure the systemd-loader entries, located in /mnt/root/boot/efi/loader/entries have the correct option root line. In our example the load entry for arch kernel would be:

```
title   Linux Arch
linux   /vmlinuz-linux
initrd  /initramfs-linux.img
initrd  /intel-ucode.img
options root="UUID=a8426465-b755-429d-9604-9c77c2838fda" rootfstype=btrfs rw audit=0
```

As you can see the root UUID is that of the btrfs one shown above.

We now use systemd's bootctl to install both <esp>s:

```
bootctl --efi-boot-option-description='Linux esp 1' --esp-path /mnt/root/efi1 install
bootctl --efi-boot-option-description='Linux esp 0' --esp-path /mnt/root/efi0 install
```

The second line could just as well be:

```
bootctl --esp-path /mnt/root/boot install
```

Doing it in this order makes the boot order efi0 then efi1.

Now run bootctl to check everything looks good and also use *efibootmgr* to check the boot order:

```
bootctl --esp-path /mnt/root/efi0 status
bootctl --esp-path /mnt/root/efi1 status
efibootmgr
```

We still need to adjust the new /mnt/root/etc/fstab. In this fstab we will mount both efi partitions. Later we will set up a mechanism to bind mount whichever <esp> was used to boot the machine to /boot.

Adjust the /mnt/root/fstab to mount each <esp> under /efi0 andf /efi1 And mount the btrfs root onto /. You can get the mounts to use by:

```
cd /mnt/root
genfstab -U .
```

In our case fstab looks like

```
# /dev/sda3 UUID=a8426465-b755-429d-9604-9c77c2838fda LABEL=root
UUID=a8426465-b755-429d-9604-9c77c2838fda / btrfs rw,relatime,ssd,discard=async,space_
↪cache=v2,subvolid=5,subvol=/  0 0

# /dev/sda1 UUID=6B7E-A837 LABEL=EFI0
UUID=6B7E-A837 /efi0 vfat rw,relatime,fmask=0022,dmask=0022,codepage=437,
↪iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0

# /dev/sdb1 UUID=6C48-1623 LABEL=EFI1
UUID=6C48-1623 /efi1 vfat  rw,relatime,fmask=0022,dmask=0022,codepage=437,
↪iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro    0 0
```

Delete the mount of /boot - we dont want or need this. We will come back to this shortly after and show how to automatically have the right currently booted <esp> bind mounted to /boot.

You can update the system before booting (provided /boot is still bind mounted of course) and it would be good to install the dual-root-tool script and bind-mount-efi.service file provided here. For Arch users you can also install the aur package.

Before we boot let's regenerate the initrds - this will of course only work provided the active efi is still bind mounted onto /boot as per above. Sorry to be repetitive but its important to avoid mistakes.

All being well you should be able to boot the system now or if you prefer you can do the next step which adds the automatic bind mount of the currently booted esp onto /boot. This is desribed in next section.

Ths tool will handle mounting */boot* as well syncing the alternate efi partitions. Handling this in a robust and safe way, was the most tricky part of the exercise!

2.2 Note on swap.

While its generally better to use a dedicated partition for swap, if there is sufficient memory that swap will not really be used much, then it may be simpler to use a swap file kept on the root raid filesystem. This also has advantage that the fstab is now references a file which is same regardless which <esp> was booted.

2.3 Mounting /boot

This was a little challenging to do properly. I had really hoped *bootctl -p* would provide a reliable way to detect which <esp> was used for current boot, but that didn't seem to be the case. So, instead I wrote the *dual-boot-tool* script. It identifies which <esp> was used to boot the system and can bind mount that <esp> onto /boot.

We also provide a systemd service unit to make this all work smoothly[5].

What needed is install the *dual-root-tool* script. The simplest way is run the installer with destination directory set to / (or install the dual-root package):

```
* ./scripts/do-install /
```

Also see Install.rst file for more info. Script installs the tool in */usr/bin/dual-root-tool* and the bind-mount-efi.service file into */usr/lib/systemd/system*.

Next add a mount option to both the efi0 and efi1 mount lines in */etc/fstab* (NB or /mnt/root/etc/fstab if you have not booted machine yet).

In my example, the efi0 line gets additional option: x-systemd.before=bind-mount-efi.service. And the same for efi1 naturally:

```
UUID=6B7E-A837 /efi0 vfat rw,relatime,fmask=0022,dmask=0022,codepage=437,
↪iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro,x-systemd.before=bind-mount-
↪efi.service 0 0
```

This will ensure both */efi0* and */efi1* are mounted before the *bind-mount-efi* service, which uses *dual-root-tool -b* to determine which 2 <esp> was used to boot the system. Armed with that information, then the active <esp> will be mounted to */boot*.

2.4 dual-root-tool

Couple of notes on the *dual-root-tool* itself

This version is written in python, as I found doing it in bash unpleasant and I think far too complex for a bash script; though I am sure there are folks more skilled than me that could make a bash version.

I think it might be a good idea to have a version of dual-boot-tool written in C++ or C at some point. That said, As of now, the python works, and besides, who doesn't have python installed these days!

The *bind-mount-efi.service* uses *dual-root-tool* to do all the real work.

If *dual-root-tool* is run with no arguments, it prints information about the currently booted <esp>. You should run this to confirm it does the right thing on your system(s).

It also supports a *-b* option to bind mount */boot* - this is what the *bind-mount-efi.service* uses.

Lasty it has a *-s* option to sync the active <esp> onto the alternate <esp>s. You want to run this using test mode via *-t* to see what it would do. For example:

```
dual-root-tool -st
dual-root-tool -bt
```

Now is a good time to reboot - all should work and you should have /boot bind mounted from the actively booted <esp>.

After booting both <esp>s are mounted : */efi0* and */efi1*.

Now let's check that tool is working, run it with no arguments:

```
dual-root-tool
```

All being well will print out the currently booted <esp>. And you can also check that it will bind mount /boot by running:

[5] Code on github and available as an Arch aur package. https://github.com/gene-git/dual-root https://aur.archlinux.org/packages/dual-root

```
dual-root-tool -b
```

You can also run it in test mode by adding *-t* option. Now that we have */boot* holding the 'actively booted' <esp>. We have overcome what we believe to be the trickiest part of making this work correctly.

Now enable the service with the usual incantation so that */boot* is mounted automatically:

```
systemctl start bind-mount-efi.service
systemctl enable bind-mount-efi.service
```

2.5 Syncing ESPs

Now that we know the active <esp> we are able to sync the other <esp> from that one.

You can use use the output of *dual-root-tool* with no arguments to identify the current booted esp - then use rsync to update the alternate <esp>. For example if the current booted <esp> is mounted on */efi0*, and the alternate is on */efi1*, then you can update the latter using:

```
rsync -v -axHAXt --exclude=/lost+found/ --delete /efi0/ /efi1/
```

This can also done by using the sync option of the dual-root-tool. Lets run it in test mode where is simply shows what would be done:

```
dual-root-tool -st
```

When ready you can remove the *-t* flag.

This can be run manually at anytime or by using a pacman hook (Arch Linux) triggered by changes to /boot. It can be run periodically from cront. The best way way is to use inotify - this requires inotify-tools be installed.

The to start the inotify based sync daemon simply run with *-sd * or *–syncd*:

```
dual-root-tool -sd
```

This will sync once, then start the inotify based daemon to sync any changes thereafter. The sync daemon monitors the currently booted <esp> mount, and whenever it gets a change event notification from inotify, it will sync to the alternate. You can run it in test mode *-t* - in this case it will print what it would do but doesn't do any copy - similar to the testing behavior when running *-s -t*.

In non-test mode you can touch a file and watch it appear in the alternate. The service unit file runs in quiet mode (*-q -sd*).

The systemd service unit is installed when using the scripts/do-install script into the usual */usr/lib/systemtd/system/dual-root-syncd.service* location.

To use the sync service, enable start as usual:

```
systemctl enable dual-root-syncd.service
systemctl start dual-root-syncd.service
```

This uses inotify to monitor */boot* for changes. Whenever a change event is detected, it then calls on rsync to update any alternate <esp> from the currently booted <esp>

2.6 How to Recover if 1 Disk Dies

First thing is machine will boot of the good disk - raid will be degraded but keep running. Replace the disk - add partitions to new disk - *bootctl install* onto the new disk's <esp>. And add the other partition back into the raid. Sync daemon will update the new <esp>.

Thats it - back in business!!

CHAPTER

THREE

SECOND APPROACH - DETAILS

For convenience, we partition each disk the same way. We choose the following standard set of partitions :

Table 1: Disk Partitions

Partition	Required	Approx Size	Comment
<esp>	yes	2 GB	FAT32, larger if no /boot
boot	no	4 GB	linux filesystem
root	yes	100 GB	
swap	no	16 GB	
home	yes	128 - 256 GB	Optional if on different disk
data	no	rest	Cache, RAID or mounted filesystem

The important partitons for the purpose at hand are the first 3 (esp, root and boot). Some schemes do not have a separate boot partition, but instead use a larger <esp> partition mounted on */boot* - that works for this pupose as well, with obvious adjustments. The most important thing is each disk has its own <esp> partition.

CHAPTER

FOUR

PREPARING THE ALTERNATE DISK

Clearly it doesn't matter whether the disks are SSD or spinners. For simplicity we'll assume the current booting disk is /dev/sda and the alternate is /dev/sdb. Adjust device names as needed.

4.1 Partitioning the disk

Use gdisk to make the 6 partitions as illustrated in *Table-1*. While there are obviously different choices one can make, each disk must have at a minimum an <*esp*> (EFI) and *root* partitions. Since we want to have the system be the same regardless which disk is used to boot the system, we want both disks to be similarly partitioned - at least for the key partitions (esp, boot, root).

Table 1: Sample Disk Partition

Partition	size	GPT Type	Label	Mount	Comment
1	2G	EF00	EFI	/efi	•
2	4G	EA00	boot	/boot	XBOOTLDR
3	100G	8300	root	/	•
4	16G	8200	swap	N/A	•
5	128G	8302	home	/home	•
6	rest	8300	data	/data	if mounted

Labels might also have a suffix indicating the disk number. For example, *root0* and *root1* Each mounts the other disk's partitions under */mnt/root1/xxx* to allow the non-booted disk to be kept in sync with the currently booted disk.

Partition 6 may or may not be mounted - for example it could be part of a raid array.

4.2 Put Filesystem on alternate disk

The starting point is a working system and the presence of the second disk to be used for the alternate root. For completeness, we'll quickly go over making appropriate filesystems. Again, the critical one is the <esp> which must be FAT32.

Now lets make filesystems on the alternate disk's partitions. We use ext4 for the linux partitions as its robust and well supported.

```
mkfs.vfat -n EFI2 /dev/sdb1
mkfs.ext4 -L boot2 /dev/sdb2
mkfs.ext4 -L root2 /dev/sdb3
mkfs.ext4 -L home2 /dev/sdb5
mkfs.ext4 -L data2 /dev/sdb6
mkswap -L swap2 /dev/sdb4
```

CHAPTER

FIVE

COPY CURRENT SYSTEM TO ALTERNATE

We'll make a copy of everything on the currently booted disk onto the alternate disk. Each disk has some things which are unique to the disk. The root drive is, by definition, unique and it's UUID is used for both booting and in its *fstab* to ensure things are mounted appropriately.

First we make a copy of everything relevant on the current disk - then we'll make the appropriate changes on the alternate to accomodate the different disk UUIDs.

While in spirit we are copying everything, we actually need to be a little more surgical. For example, we dont want to copy /dev, /sys, /proc or even tmpfs directores such as /tmp. Instead we copy only the things we actually need.

For example we might populate the alternate using:

```
mkdir -p /mnt/root1
mount /dev/sdb3 /mnt/root1
cd /mnt/root1
mkdir -p boot data dev efi etc home mnt opt proc root run srv sys usr var tmp
# if you have any NFS mount points add as needed

alt="/mnt/root1"
opt="-avxHAX --exclude=/lost+found/ --delete --info=progress"
rsync $opt /efi/EFI $alt/efi/
rsync $opt /boot/* $alt/boot/
rsync $opt /bin /lib /lib64 /usr $alt/
rsync $opt /root $alt/
rsync $opt /var $alt/
rsync $opt /etc $alt/
rsync $opt /data/* $alt/data/
rsync $opt /srv $alt/
rsync $opt /home $alt/
```

5.1 Modifications for different UUIDs

Now that the alternate disk has its own copy of the system, we need to make the appropriate modifications so booting and mounting reference the correct disk. If we didn't change it, they would all be referring to the first disk.

First lets fixup mounts.

5.2 Updating fstab

First lets edit the alternate disk's fstab - we'll also add a few lines to mount first (currently booted) disk under /mnt/root1.

Identify the UUIDs of the alternate disk using blkid or lsblk:

```
# lsblk -f /dev/sdb
NAME   FSTYPE FSVER LABEL UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
sdb
├─sdb1 vfat   FAT32 EFI   74B3-8D8F                                 2G     0% /efi
├─sdb2 ext4   1.0   boot  0436e342-856a-495e-bd07-5f0dab1525fe    3.3G     9% /boot
├─sdb3 ext4   1.0   root  385c796c-a046-4bcb-b0e6-bec6dd543faa   68.9G    24% /
├─ ...
```

Our focus is on <esp>, boot and root. If you're using /home or /data then record those as well.

Now edit **/mnt/root1/fstab** (NOT /etc/fstab!) and duplicate the existing 3 lines for /, /efi and /boot, Next change the UUID to be the ones from the alternate disk obtained above.

In same fstab, change the mount points for the other disk so they now all get mounted under */mnt/root1*:

- change / to */mnt/root1*
- change */efi* to */mnt/root1/efi*
- change */boot* to */mnt/root1/boot*

Of course, do same for any other mounted partitions (e.g. /home).

Lastly, edit the current disk's **/etc/fstab** and add mounts for the new alternate disk - now the alternate disk gets mounted under /mnt/root1.

One that's done, each fstab has mounts for the *other* disk on /mnt/root1, /mnt/root1/efi, /mnt/root1/boot etc.

5.3 Updating systemd-boot loader entries

The boot loader entries that are used by sd-boot each reference the root disk. We must now update those on the alternate disk to point to their own (alternate) disk.

Edit each entry in **/mnt/root1/boot/loader/entries/*** and change the kernel option line:

```
options root="UUID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" rw
```

to have the correct UUID found above - in our case this would be:

```
options root="UUID=385c796c-a046-4bcb-b0e6-bec6dd543faa" rw
```

Once they're all done we're almost ready - in the next section we'll install a boot loader.

5.4 systemd-boot install

All that's needed now is to install boot loader into the alternate <esp>. sd-boot makes this straightforward to do:

```
bootctl --esp-path /mnt/root1/efi --boot-path /mnt/root1/boot --efi-boot-option-
↪description="Linux Alt Boot Manager" install
```

We specify a descriptive name, so that any system boot menu will show a different name than the default used for the first disk. The name of either can be easily changed at any time.

This will also put the alternate disk first in the boot order - you can leave it or change it back to original disk - we'll discuss more below. First lets check to make sure things look good.

Check the current booted disk:

```
bootctl status
```

This should look same as always. Now let check the alternate disk:

```
bootctl --esp-path /mnt/root1/efi --boot-path /mnt/root1/boot status
```

This should look good. Please note sd-boot may issue or issues a warning which can safely be ignored.

bootctl compares the esp UUID with the UUID of the esp that was used to boot the current system. It warns if they differ. Well they should differ by design - we want 2 <esp> each with its own UUID. So this is a *good* thing. The warning will happen for whichever disk is NOT currently booted.

Its also a good idea to check the boot order saved in the efi variables:

```
efibootmgr
```

You should now see both Linux entries listed.

CHAPTER

SIX

TESTING AND TIDYING UP

At this point we are ready to test. There are a few non-essential convenience things that may be desirable.

We changed the boot desciption - we may also want to change the boot desctiption of the original disk's <esp> as well. If we have not rebooted, then the original disk <esp> is mounted on /efi:

```
bootctl --esp-path /efi --boot-path /boot \
        --efi-boot-option-description='01 Linux Alt' install
```

This will also make this disk the first in the boot order. Boot order can also be changed using *efibootmgr*. For this case we don't need to specify the esp or boot paths as they are the defeaults. Doing it this way makes it explicitly clear.

It may be useful to change the title of each loader entry - e.g.

```
[/mnt/root1]/boot/loader/entries/xxx.conf
```

Perhaps prefix the title with 01 or 02 depending which disk it is for.

Be careful with the loader entry file names. If name is changed then the /efi/loader/loader.conf, which references the filename in the *default* line, will need it's filename changed to match.

6.1 Keeping Disks In Sync

Finally, we need to keep the disks in sync. The simplest way to do this is use the dual-root-tool

For this use case you can turn off the autosync which handles the first approach (using /efi0 /efi1 etc). Copy the sample config */etc/dual-root/sync-daemon.conf*. It has comments. Turn off the code that handles approach 1 by setting *dualroot = false* then make a list of items to sync. By default, *dualroot* is true. Each item will be used with rsync, and are therefore in rsync format (careful with trailing slashes!). Each item has [source_dir, dest_dir(s), exclusions].

To confirm it will do what you want run it in test mode

```
dual-root-tool -sd -t
```

It will print what will happen. Once you're happy,m then enable and start the daemon:

```
systemctl enable  dual-root-syncd
systemctl start  dual-root-syncd
```

This is examnple sync daemon config

```
# rsync_opts =          # default: if unset "-axHAX"
dualroot = false        # default: true
nice = 15               # default: 15
```

(continues on next page)

(continued from previous page)

```
ionice_class = 3        # default: 3 (IDLE) see : man ioprio_set or man ionice
# ionice_value = 6      # only relevant for ionice_class realtime(1) and best-effort(2)
sync_delay = 60         # seconds to sleep after each rsync
sync = [
    ["/efi/EFI", "/mnt/root1/efi/"],
    ["/boot", "/mnt/root1/", ["/boot/loader"]],
    ["/bin", "/mnt/root1/"],
    ["/lib", "/mnt/root1/"],
    ["/lib64", "/mnt/root1/"],
    #["/var", "/mnt/root1/", ["/var/cache/pacman/pkg"]],
    ["/etc", "/mnt/root1/", ["/etc/fstab"]],
    ["/srv", "/mnt/root1/" ],
    ["/home", "/mnt/root1/" ],
    ["/opt", "/mnt/root1/" ],
    ]
```

Note the example has exclusions to exclude */etc/fstab* and */boot/loader* Note that the following rsync_options are always used in addition to above *rsync_opts*:

–no-specials –atimes –open-noatime –exclude=/lost+found/ –delete

CHAPTER

SEVEN

EPILOGUE

There is some discussion around dual root and some of the challenges using mdadm RAID1 on the arch general mail list[3].

This brings me to a couple of todo items:

Todo #1: Use same basic mechanism as Second Approach to do fast installs.
Build a tool to do fresh installs from a template root drive.

For an install, one can imagine doing pretty much same thing as the second approach, but instead do a fresh install from a template. Of course care needs to be taken to avoid any services that are unique to the template machine. One way to apprach this might be to take a workstation install (with no services like mail, databases, etc) and use sync script to create a template to install from.

May need a little tweaking but then the template could be rsync'ed over the local network (or from a USB drive). This should make it reasonably straightforward and fast to get things installed. Needs some scripting work and a good template machine to get the ball rolling.

[3] https://lists.archlinux.org/archives/list/arch-general@lists.archlinux.org/thread/KAMOXQTWQCPCC5KNFF6IOUSFPMNMLIIW/

CHAPTER

EIGHT

END NOTES

CHAPTER

NINE

LICENSE

- SPDX-License-Identifier: MIT
- Copyright (c) 2023 Gene C

CHAPTER

TEN

DUAL ROOT - INSTALL

10.1 Docs

To build pdf or html version of the readme:

```
cd docs
make html
make latexpdf
```

This will create _build/latex/dual-root.pdf and a set of html pages under _build/html sphinx (aka python-sphinx) must be installed

10.2 duel-root-tool

On arch simply build and install the package. The PKGBUILD is on AUR and under packaging directory.

Otherwise manually install dual-root-tool, run the installer script as root with the destination directory set to /

```
./scripts/do-install /
```

This installs into

```
/etc/dual-root
/usr/share/dual-root
/usr/share/licenses/dual-root/

/usr/bin/dual-root-tool
/usr/lib/systemd/system/bind-mount-efi.service
/usr/lib/systemd/system/dual-root-syncd.service
```

/usr/bin/dual-root is now a symlink to */etc/dual-root/dual-root-tool* This allows us to organize the code a little better.

As usual to activate the bind service:

```
systemctl enable bind-mount-efi.service
systemctl start bind-mount-efi.service
```

And for the inotify based sync daemon:

```
systemctl enable bind-mount-efi.service
systemctl start bind-mount-efi.service
```

Remember to ensure that the <esp>s get mounted before bind mount service. That is done using systemd mount option in fstab. Add option *x-systemd.before=bind-mount-efi.service* to each of the <esp> mount lines:

```
UUID=... /efi0 vfat rw,...,x-systemd.before=bind-mount-efi.service 0 0
UUID=... /efi1 vfat rw,...,x-systemd.before=bind-mount-efi.service 0 0
```

CHAPTER

ELEVEN

CHANGELOG

[3.0.0] —— 2024-10-17

```
Performance improvements:
 - sync daemon now defaults to running with: nice=15, ionice_class=IDLE
 - Inotify triggers now placed on sync queue which is run in separate thread and queue␣
↪run must wait sync
   delay seconds before being run again. Each dir monitored by inotify has its own queue.
 - Periodic queue check/flushes are also run every 15 mins - this is also run on exit to
   ensure all pending syncs are completed.
 - New variables in sync daemon config are available
   nice, ionice_class and ionice_value; see man ioprio_set
   defaults: nice=15, ionice_class=3 (IDLE)
   sync_delay = 300 which is the minimum time between queue runs
update Docs/Changelog.rst
```

[2.9.0] —— 2023-09-27

```
Reorganize Docs and migrate to rst
update CHANGELOG.md
```

[2.8.0] —— 2023-09-25

```
sync code : remove obvious duplicate elements from rsync options list.
  We dont catch short vs long options or combined option flags such as "-t" vs "-atx"
update CHANGELOG.md
```

[2.7.0] —— 2023-06-28

```
rsync options now adds --times --no-specials to rsync_opta
Sync once before starting the inotify based sync daemon
update README
update CHANGELOG.md
```

[2.6.0] —— 2023-06-27

```
new rsync additional options
        default base options: "-axHAX --no-specials" which can be changed in the config␣
↪file
        always added to base: "--atimes --open-noatime --exclude=/lost+found/ --delete""
update CHANGELOG.md
```

[2.5.0] —— 2023-06-27

```
Add new variable rsync_opts to config file.
  Defaults to "-axHAX --no-specials".
update CHANGELOG.md
```

[2.4.0] — 2023-06-26

```
use --no-specials rsync option for the sync daemon
More word smithing on readme
update CHANGELOG.md
```

[2.3.2] — 2023-06-26

```
minor readme tweak
update CHANGELOG.md
```

[2.3.1] — 2023-05-17

```
Simplify Arch PKGBUILD and more closely follow arch guidelines
update CHANGELOG.md
```

[2.3.0] — 2023-04-29

```
Fix typo when fixing previous typo ...
update CHANGELOG.md
```

[2.2.0] — 2023-04-29

```
Fix typo in error message
update CHANGELOG.md
```

[2.1.1] — 2023-04-26

```
For Arch mkpkg users Add _mkpkg_depends to PKGBUILD so rebuilds package when python is␣
↪updated
Add short note about swap file for approach 1.
update CHANGELOG.md
```

[2.1.0] — 2023-03-12

```
tidy / simplify inotify terminate() method.
readme tweaks
update CHANGELOG.md
```

[2.0.3] — 2023-03-10

```
Readme tweaks, systemd unit description improvements
update CHANGELOG.md
```

[2.0.2] — 2023-03-09

```
Doc wordsmithing
Wordsmithing README
update CHANGELOG.md
```

[2.0.1] — 2023-03-09

```
Tidy some coding comments
update CHANGELOG.md
```

[2.0.0] —— 2023-03-09

```
Tweak systemd service descriptions
update README with new syncd info
Re-write sync code
  New Sync and Inotify classes
New optional sync-daemon.conf allows specifying what to sync with list of :
    [source, destination(s), exclusion(s)]  - each in rsync compatible form
Can be used with Approach 2
Remove timeout=0 from select()
update CHANGELOG.md
```

[1.0.2] —— 2023-03-07

```
Forgot to add dual-root-syncd.service file - added
Remove inotify todo item - its done :)
update CHANGELOG.md
```

[1.0.1] —— 2023-03-07

```
Comment change in inotify code. Add couple lines on recovering from disk failure to docs
Add comment on recovering from disk failure
update CHANGELOG.md
```

[1.0.0] —— 2023-03-07

```
Release 1.0.0
Inotify sync option (dual-root-tool -sd) available
dual-root-syncd.service to start the sync daemon
update CHANGELOG.md
```

[0.9.1] —— 2023-03-07

```
update to 0.9.1
Refactor and tidy up code
update CHANGELOG.md
```

[0.9.0] —— 2023-03-07

```
Add -q quiet option to dual-root-tool
update Install.rst instructions
Install uses /etc/dual-root
tidy up installer
small doc edits
update CHANGELOG.md
```

[0.7.0] —— 2023-03-06

```
fix installer typo
update CHANGELOG.md
```

[0.6.0] —— 2023-03-06

```
Add sphinx docs - cd docs; make latexpdf; make html
update CHANGELOG.md
```

[0.5.0] —— 2023-03-06

```
tweak doc, update to 0.5.0
More edits for dual-root-tool
update CHANGELOG.md
```

[0.4.0] —— 2023-03-06

```
add more protective checks
update CHANGELOG.md
```

[0.3.0] —— 2023-03-06

```
Add sync and test mode
update CHANGELOG.md
```

[0.2.0] —— 2023-03-06

```
Add dual-root-tool and bind service
more doc updates
Initial commit
```

CHAPTER

TWELVE

MIT LICENSE

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search