How to get image file or backup file from Xavier NX or TX2 and copy that on eMMC of new Xavier NX or TX2?

reflash

Omid306 1 October 2, 2023, 7:20am

We have developed two Jetson Xavier NX and TX2 for AI applications. We want to develop them on similar boards and mass produce them. So we want to get image file or backup from them and copy to eMMC of new board with all developed programs and installed packages. We want to know this is the correct way to do it? If yes, how can I do that? and if not, what is the best way?

DaveYYY 2 October 2, 2023, 7:38am

Hi,

I suppose you are using JetPack 4.x since you have both Xavier NX and TX2, and if that's the case, please refer to `Linux_for_Tegra/README_Massflash.txt` on how to use Massflash.

Omid306 3 October 2, 2023, 12:17pm

Hi, thank you for your reply.

I do not want to fuse multiple Jetsons (like that text file). You suppose I want to get a image file or backup from Xavier (jetpack 5.1) or TX2 (jetpack 4.x) with dd command (or any better procedure you suggest). At first, how can I do that and then how can I use that?

I got an image file with this command:

ssh System_name@Device_IP "sudo -S dd if=/dev/mmcblk0p1 | gzip -1 -" | dd of=image.img status=progress bs=4M

But I could not use image.img file to make sure I have backed up correctly or not.

DaveYYY 4 October 2, 2023, 12:23pm

Hi,

I thought you are looking for something like massflash because you mentioned this:

> Omid306:
>
> We want to develop them on similar boards and mass produce them.

If you do not need massflash, then this command should also do it:

> sudo ./flash.sh -r -k APP -G <clone> <board> mmcblk0p1

It's the same on both JetPack 4 and JetPack 5.
https://docs.nvidia.com/jetson/archives/r35.4.1/DeveloperGuide/text/SD/FlashingSupport.html

linuxdev 5 October 2, 2023, 9:35pm

Added information you might want…

If you use `dd`, then it is imperfect because it requires the Jetson to be running. For example, temporary files, lock files, so on, will exist. `rsync` tends to be a bit better if you `rsync` to a loopback mounted `ext4` file on the host PC (this is how `rsync` would recreate a partition instead of just a subdirectory of content; `dd` directly produces a file of the exact size). However, cloning (as given by @DaveYYY) is superior.

A clone gives two copies of the rootfs/APP partition, all from the shut down state (no lock other temporary files being open). One copy/clone is a "raw" clone, which is a bit-for-bit exact replica of the rootfs. The other is a "sparse" clone, and is smaller; this clone is basically the size of the content in the partition, and as the content approaches a filled partition, the size of the sparse clone approaches the size of the raw clone. Thus, if you clone a full device, your host PC will need spare disk space in excess of twice the partition size. If the partition is 28 GB, then that means a full system uses up 56 GB of space; a typical base install would use about 35 GB. These are huge files and take a long time to move across devices, or copy, so on.

Both sparse and raw clones are useful for flashing. Sparse clones just take less time to flash. Sparse clones can be created from raw clones via a NULL filler pattern in the `mksparse` program. I tend to throw away my sparse clones though. The reason why is that raw clones are much more useful; the sparse clones are basically useful only for flash. Raw clones can be loopback mounted, read, content copied, modified, so on, from any Linux computer. One can even update the original device and use `rsync` over the network to quickly update the loopback mounted raw clone. When not in use I run `bzip2 -9` on the raw clone (this alone takes a very long time to do).

A clone is like `dd`, but it runs when the system is shut down and not mounted. Keep in mind that if the filesystem is corrupt, e.g., incorrectly shut down, then clones and `dd` will produce an exact replica which is also corrupt.

The non-rootfs content could be considered the equivalent of the BIOS plus boot content (Jetsons don't have a BIOS, they do the equivalent in software). Normal flashing of a Jetson is like also flashing the BIOS at the same time. The cloned content will be guaranteed compatible for that hardware model and that boot content, but not necessarily compatible as versions change. So for example, if you clone from L4T R32.6 (you can see the L4T release via "`head -n 1 /etc/nv_tegra_release`") on an Xavier to another Xavier, and you use the R32.6 software to flash not only the clone but also the non-rootfs content, then it should always "just work". If you clone on R32.x and flash this to the wrong R32.x release (a few releases will be compatible), then as you get further from that original release, it becomes more likely your flash will work, but boot will fail. If you use a rootfs on R32.x with an R35.x, then it is guaranteed to fail. If you clone from a TX2 on R32.x to the same release on an Xavier, *but use the correct non-rootfs content*, then *maybe* it will work. This latter has a lot of failure possibilities.

If you need to clone from an R32.x TX2 to an R35.x Xavier, then the raw clone is still useful, but that clone should not be directly flashed to the Xavier. You could, it just wouldn't work. That clone (if raw) is still very very useful in your case. You can restore the TX2 with that clone at any time by flashing with the correct L4T release and the clone. You can also mount the clone on the host PC over loopback, and copy subsets, e.g., the home directory or "`/usr/local`" content (but "`/usr/local`" content is also probably invalid going from R32.x to R35.x; on the other hand, that content is likely valid if copied from a TX2 of the same L4T release to an Xavier). This latter is because R35.x does not support CUDA 10, which a TX2 uses, and requires CUDA 11 (or 12 via docker).

I highly recommend cloning and experimenting with loopback mount of the raw clone. If you clone via this command:
```
sudo ./flash.sh -r -k APP -G my_backup.img jetson-tx2
```

`mmcblk0p1`
…then you will get sparse clone "`my_backup.img`" and raw clone "`my_backup.img.raw`".

If you have "`my_backup.img.raw`", you can examine it like this on the host PC:

```
sudo mount -o loop ./my_backup.img.raw /mnt
cd /mnt
ls
cd -
sudo umount /mnt
```

You could replace "`Linux_for_Tegra/bootloader/system.img`" with either the raw or sparse clone, and then flash like this to another TX2 to get an exact copy of the rootfs while also installing the correct non-rootfs content (requires using the same L4T release):
`sudo ./flash.sh -r jetson-tx2 mmcblk0p1`
(notice the "`-r`" which says to "reuse" the image instead of generating one; this is sometimes used for non-flash operations as well since you don't always want a new image)

You could also `rsync` from the loopback mounted clone's "`/home`" to "`Linux_for_Tegra/rootfs/home/`" and you would magically get all *new* generated images to use that home. This content would be valid on any of the Jetson hardware, and across any of the L4T releases (mostly). Let's say you have `ssh` keys in "`~/.ssh/`", and you've *properly* added this to "`rootfs/`" via `rsync`…the default flash would already have those keys on every flash. Very useful because boot does not depend on "`/home`", it isn't part of the system, and so unless home itself contained something which is version dependent, then home via a clone will "just work" in "`rootfs/home/`".

Omid306 6 October 7, 2023, 7:56am

Hi,
Thank you for your replies. Both of them worked and were useful. I have two questions about them.

Q1: How can I find different board names and use them instead of "board"? (in this case I want to find name of TX2 and Xavier NX. I used "jetson-xavier-nx-devkit-emmc" for Xavier NX and worked for me but I am not sure that is correct. In addition I did not find any correct name for TX2)

Q2: Is there any way to do this without needing Linux_for_tegra (and

flash.sh file)? This means without knowing version of L4T.
Until now, I was using it in the following way:

- Getting version of L4T with “head -n 1 /etc/nv_tegra_release”.
- Creating the Jetson OS Image according L4T version with SDK manager and skipping to flash it.
- Using Linux_for_Tegra directory and flash.sh file.

---

linuxdev 7 October 7, 2023, 3:12pm

For the first question, if you go to the “`Linux_for_Tegra/`” directory, and run this command you’ll see the answer:
`ls -l jetson*.conf`

Notice that each is just a symbolic link to the real file. That symbolic link is just more human-readable. The files being linked are named after a combination of the module’s model along with the carrier board’s model (sometimes including variants of the revision). All of these files are human-readable, and often include some minor amount of data, but then include another file. It is sort of a poor man’s version of object-oriented file inheritance.

I should add that the SoC (“System on a Chip”) used in Jetsons often reuses a previous generation’s silicon for some small subset. For example, the serial UARTs might be the same silicon in a TX1 as in the TX2, so you might see inclusion of some subset file which seems to be from the wrong Jetson, but it isn’t wrong. The SoC itself has names like `t124`, `t210`, `t186`, `t194`, or `t234`. An older SoC won’t refer to a newer SoC file, but a newer SoC file might refer to the shared silicon of an older SoC.

Incidentally, the SoC names tend to be for a “series” of implementations. Long ago there was a Tegra 3, and the reference would have been to a t30, whereby the `0` was the first implementation. If you look at something like the `t186` (which is a TX2), then there might be several `t18x` SoCs, and the implementation in the Jetson is the `6` version. Perhaps there is hardware we’ve never heard of based on the same SoC, but with a different footprint in silicon, and mounted on a different module PCB. If you see references to a `t###`, then this is likely a finer division of describing a module because it is breaking it down to the SoC itself.

---

system Closed 8 October 21, 2023, 3:13pm

This topic was automatically closed 14 days after the last reply. New replies are no longer allowed.

Related Topics

Topic	Replies	Activity
Transition from Jetson Nano DevKit to Jetson TX2 NX reflash	2	March 1, 2022
Taking an image backup of the Jetson clone	3	February 18, 2022
Flashing the custom image on EMMC reflash	12	October 18, 2021
How to backup my rootfs clone	6	March 12, 2022
Create a backup image of the NX module flashed clone	4	August 29, 2021