Notes on the antiX live system, as used by MX Linux

These notes on the antix Live system, as used by MX Linux, were created by forum member thomasl, and are reproduced here with his kind permission:

(Disclaimer: these notes are based on my understanding of MX persistence which may or may not be correct. So I am grateful for corrections and/or additional material from those in the know. There is also a series of videos by @dolphin_oracle which explain and show many of the details I am addressing with this and the next post.)

For me, one of MX Linux's best (and IMO underrated) features is the frugal install option (and the related persistence options). Why? Well, for a number of reasons actually and I will explore some in this and the following post.

First a frugal MX install can be reduced to as few as four files. Yes, that's right... a complete Linux installation in just four files: save those files to a backup disk (or two) or (securely encrypted) to the cloud and you have a full system backup. Even a full-featured install with all the bells and whistles (as described below) is only one file larger (there are also some small, machine-specific additional files but these are not required for a backup). How cool is that?

Second, those four (or five) files can reside on ANY filesystem. That means you can have a complete Linux system up and running on your Windows PC without the need to repartition your disk(s) or change the filesystem on any of the existing partitions. This is especially valuable if you want to "test the waters" with MX Linux. Or take this: I can copy my system (ie those four or five files) to a USB stick and install it with just a few CLI commands on almost any PC (with enough free space for those files). So deploying MX on a second (or third...) PC or a laptop is a breeze.

Let's take a closer look at those four files. On the one hand we have vmlinuz and initrd.gz which are the two basic files needed by Grub (or another suitable boot manager) to boot the system off the disk. vmlinuz is the Linux kernel and initrd.gz contains an initial ramdisk used to load and initialise the full system.

On the other hand we have linuxfs and rootfs. These two files are typically much bigger than vmlinuz and initrd.gz as they hold the "meat" of the system (my linuxfs file currently has a size of 1.87GB while my rootfs is always exactly 2GB). linuxfs contains a compressed read-only filesystem with the complete system, every single file that is needed to run MX Linux plus the user's files. In my case linuxfs is slightly below 2GB (with lz4 compression), but I am using a heavily customized system, with loads of stuff I don't want or don't need removed.

Now, a read-only filesystem has many advantages (for instance a virus/trojan can't compromise it) but of course it can't store any changes. That's where the other file, rootfs, comes in. This is one of two persistence files supported by MX; it contains a full Linux filesystem, though this is initially (almost) empty.

How does this persistence feature work? Well, the init code in initrd.gz basically does three things (it does a lot more but that's the principle as far as persistence is concerned): first it mounts linuxfs so that the contents can be accessed as if they were a normal disk partition (but, as mentioned, that's a read-only mount). Then it mounts the other file, rootfs, as read-write. Finally, it uses the rootfs mount to create a layer (an overlay in tech speak) above the linuxfs mount such that accesses to the filesystem first go through the rootfs layer before accessing the linuxfs layer. Now we have to separate two scenarios: a simple read access and a write access. First the reading: if an app wants to read a file then the overlay system intercepts that call and checks whether that file is already to be found in the rootfs layer. If so it's read from there and returned. If not, it's instead read from the linuxfs layer and returned. Writing is a different story though: every write access is intercepted and goes directly to the rootfs layer, ie that file or change is always stored in the rootfs filesystem. That's basically how MX persistence is implemented. (There are some slight complications like deleting files but the basic principle is that all changes to a running frugal install with root persistence enabled are intercepted and stored in the rootfs layer.)

Now there are two ways the rootfs file can be mounted. One is that it's mounted directly (ie as a loop device); this means that every change that is caught is written immediately to the rootfs file and so ends up on the hard disk. That mode is called static persistence. The other mode (called dynamic persistence) works by copying the complete contents of rootfs from the disk into RAM on boot and then mounting this RAM copy. Again, all write accesses and other changes are intercepted but this time they are stored in RAM only and not in the rootfs file on disk. This means that dynamic persistence is faster than static persistence (especially if the rootfs file is located on a slow USB stick), but it also means that all changes are stored in RAM only and can get lost, for instance if there's a power cut. Of course this also means that a chunk of RAM is not available for running the system and whether that is a problem depends on the total amount of RAM installed. In my workhorse PC I have 16GB and I've run into any trouble but a PC with just 4GB will probably not work very well with dynamic persistence.

Further, in this mode, the user has to make sure that changes are indeed written at some point to the rootfs file on disk if they are to be permanent. This can be done in two ways: explicitly by running a CLI utility called "persist-save" or by using the mx-remastercc utility (button "Save root persistence"). Or it can be done on shutdown, either manually or automatically. Again, see the mx-remastercc tool (button "Configure live persistence") or the CLI utility "persist-config". Either way, what happens is that all the accumulated changes held in RAM are then written out to the rootfs file on the hard disk in one go.

As already mentioned, dynamic persistence does require a PC with "enough" RAM to copy the contents of the rootfs file. OTOH, the actual size of the rootfs file in itself is not the real measure for the RAM required, as only the space used by files is allocated. For instance, I have a 2GB rootfs file but it currently takes only 1.2GB of RAM.

The persistence options don't stop with rootfs. It is possible to create and mount a further persistence file, called homefs. This file only stores changes to the content of the /home directory but in other respects works similar to static root persistence. This means the homefs file is mounted as read-write and all writes to any file in /home (or any other change in /home) will be intercepted and immediately stored in the homefs file on disk. Home persistence is always static, in other words.

So far, so good... but how do rootfs and homefs interact with each other? Or put differently, where do changes ultimately end up? We have the six following permutations:
1. No persistence: no changes are saved anywhere
2. static rootfs only: changes (including those in /home) are saved in rootfs on disk as they happen
3. dynamic rootfs only: changes (including those in /home) are saved in a ram copy of rootfs as they happen and have to be saved to disk either explicitly or on shutdown
4. homefs only: changes in /home are saved in homefs on disk as they happen; other changes are not saved
5. static rootfs and homefs: changes (excluding those in /home) are saved in rootfs as they happen; changes to /home are saved in homefs on disk as they happen
6. dynamic rootfs and homefs: changes (excluding those in /home) are saved in a ram copy of rootfs as they happen and have to be saved either explicitly or on shutdown; changes to /home are saved in homefs on disk as they happen

So let's look into what happens when we create (or change) two different files, one in /usr and one in the user's home directory for those six modes:
sudo touch /etc/testfile; touch ~/testfile
1. Neither of these files is saved.
2. Both files end up in the rootfs file on disk.
3. Both files end up in the rootfs copy in RAM and have to be saved, either explicitly or when shutting down. If this is done, both files end up in the rootfs file on disk.
4. ~/testfile is saved in the homefs file on disk. /etc/testfile is not saved.
5. ~/testfile is saved in the homefs file on disk. /etc/testfile is saved in the rootfs file on disk.
6. ~/testfile is saved in the homefs file on disk. /etc/testfile is saved in the rootfs copy in RAM and has to be saved, either explicitly or when shutting down. If this is done, the file ends up in the rootfs file on disk.

For 99% of the time I use only two of those options. My "normal" mode of operation is mode 6, so that changes to /home are saved immediately and all other changes are stored in the RAM copy of rootfs. However, I NEVER save any of those changes to disk manually or on shutdown, so when I boot into mode 6, the rootfs file on disk never changes between boots. This opens interesting possibilities, ie for testing stuff or test-installing some software. It also means that a virus or some other miscreant can't really damage the contents of rootfs file on disk. This setup will even survive a "sudo rm -rf /usr" or similar such CLI catastrophes.

If and when I want to change something outside of /home (ie do a system update, install an app etc) I first save the current version of the rootfs file to a backup medium, then (re-)boot into mode 5, do all the changes I have to do (which go to the rootfs file on the disk immediately) and finally reboot back into mode 6. If all went well, I just continue but in case of a problem I can simply copy the backup of rootfs I created before I began updating/changing the system. I don't do this very often, about once or twice a month.

(You may ask yourself why I do not do my changes while running dynamic persistence (ie mode 6) and then simply call "persist-save". Well, I've experienced reproducible problems with saving file changes while running dynamic persistence and so I have stopped using this feature (see https://forum.mxlinux.org/viewtopic.php?t=56823 for details). It is quite possible that this problem has been resolved in newer versions of MX but I have never checked this as my current setup is rock solid and stable. YMMV.)

Sometimes I just want to change a single file (say in /etc) or simply add a couple of files. In this case I won't necessarily go to the trouble of rebooting into mode 5. Instead I simply mount the rootfs file ("sudo mount -o loop /live/boot-dev/<YOUR-MXLINUX-DIR>/rootfs /mnt/rootfs"), change the file(s) I want to change and unmount /mnt/rootfs. For instance, if I wanted to change /etc/environment, I would edit /etc/environment as root, mount the rootfs file as shown above, then copy /etc/environment to /mnt/rootfs/upper/etc/environment as root and unmount (however, this only makes sense with dynamic rootfs persistence, NOT with static rootfs persistence). I rarely do this though.

This setup has proved to be rock-solid (in use for more than 4 years, starting with MX18) and the small inconvenience of having to boot into static persistence occasionally for updates and installs is, for me, well worth the added stability and peace of mind. But of course, MX Linux frugal installations and their persistence options are so flexible that this is just one of many possible ways to do it and others may find a completely different set of options much better suited to their needs.

So what if, after many weeks or a few months, the rootfs file has accumulated so many changes that there's not much free space left? For my 2GB-sized rootfs file reaching the point where a little above 1.6GB is used normally takes around four to six months and the remedy is quick and easy. I simply create a full system snapshot with the mx-snapshot tool (using its "Preserving accounts" option). A snapshot is actually an ISO file with all the files required to boot the full system and it contains, in directory /antiX/, a brand-new linuxfs file which has all the changes accumulated in the rootfs file added. So I basically open that ISO file, copy the new linuxfs (and initrd.gz if it has changed) into my MX boot directory to replace the old linuxfs and delete the old rootfs file (because it is not needed anymore). Then I boot into mode 5, the MX init logic recognises that there is currently no rootfs file and lets me create a new, empty one. As mentioned, my rootfs file is always 2GB in size, although the actual RAM required for dynamic persistence will be less as only the used space in the rootfs file needs space in RAM. The homefs file remains all the time untouched and will continue to work with the new linuxfs and rootfs files.

There is of course more to creating and using snapshots than this short sketch shows but that is probably a good candidate for a separate post.

One nice persistence gimmick deals with differences between rootfs and homefs. As described above if you work with both root and home persistence (ie mode 5 or 6 in above post) then all changes in /home end up in homefs, while all other changes end up in rootfs (if static or saved). So if you add a file to /home/$USER then this new file will end up in homefs but it will not be saved in rootfs's version of /home/$USER. And why should it... it's in homefs, after all. However, if the system is booted into modes 2 or 3 (ie with home persistence disabled) then all those files and changes in /home which have been accumulating in the homefs file will not be present as home persistence is not active. But why would anyone want to boot into mode 2 or 3? Well, very rarely, when I want to test something really, REALLY iffy, I want to boot into a system that's totally safe from ANY changes. Mode 6 doesn't fit the bill as it will prevent changes to rootfs but not to /home as these are saved in the homefs file as they occur. Yet, a completely isolated system can be achieved by booting into mode 3, where root persistence is dynamic and home persistence is off. But... if home persistence is off then all changes saved in homefs are not visible! That's not what I want.

Fortunately, there is a way to transfer all those accumulated /home changes stored in the homefs file back into the rootfs file; this nicely shows the flexibility of the MX persistence system. It is not a trivial task but once mastered, it works very well. Basically, once every blue moon I boot into mode 5 (ie static root persistence and home persistence). In this mode the homefs file is mounted at /home. By contrast, the overlay for the rootfs file is mounted at /live/aufs and changes to the rest of the system will immediately end up in rootfs.
So with a suitable synchronisation programme (ie FreeFileSync or similar) I can now simply update the /live/aufs/home/$USER directory (which lives in rootfs) with the contents of my /home/$USER directory. Say I have a file /home/$USER/testfile that is not yet in rootfs: if it is copied from /home/$USER/testfile to /live/aufs/home/$USER/testfile, it's now also available in rootfs. And so on.
The great advantage, after this synchronisation from homefs to rootfs has been done, is that I can then boot into mode 3 (ie dynamic root persistence and NO home persistence at all) and have all files, including everything in homefs, at my disposal. Now no changes, either in /home or the rest of the system, are ever saved to disk (as long as they are not saved by the user). A system booted in this way will survive anything, even "sudo rm -rf /".

To round this up, here's one final trick that may be worth knowing. It is entirely possible to have more than one set of persistence files ("set" here means either one or both of the persistence files) and to boot into two, three or more completely different systems. This can be done with the help of the "pdir=" boot option. Basically, this option tells the persistence routines where to look for the rootfs and/or homefs file(s). So you could have one MX directory with your "normal" system files (ie vmlinuz, initrd.gz, linuxfs and the persistence files) and then another directory (or even two or more) with a completely different set of persistence files (and just those are required, not any of the other files). This comes in handy if you want to test-drive some software for an extended period of time or if you want to install something but not have it in your "normal" system.

For instance, to create a second system that has GIMP installed, you could create a directory, say GIMP, below the directory that holds your frugal MX installation. Then you'd copy the current rootfs and/or homefs file(s) into the GIMP directory (probably a good idea to do that not while booted into the frugal install itself but with a live USB version as the homefs file changes all the time). Next you would boot into the "GIMP" system by creating a new boot entry by adding "pdir=<YOUR-MXLINUX-DIR>/GIMP" to your boot options. Then you install GIMP (which lands in the rootfs file in <YOUR-MXLINUX-DIR>/GIMP) and whenever you want to work with the GIMP version you simply boot with this rootfs file. Your main system remains totally untouched.