Do-it-Yourself Debian Remix OS

This tutorial explores a method by which a motivated user can generate a personally customized version of a live boot OS based on the Debian Gnu/Linux operating system.

This tutorial assumes the potential user is already familiar with the concept of a live boot operating system, especially for some version of Linux. It also assumes (eventually) a certain fluency with Linux operations and terms. Live boot OSes allow a user to try out (or use everyday) an alternative operating system without making any unwanted permanent commitments or installations to a host computer.

Contents

1. Why a personal Remix is possible and useful
2. Support for remix OS building by the official OS vendors
3. Why Debian Mate
4. The nature of the Debian live-build process
5. The awkward recent history of the live-build project
6. The basic method to make your custom Debian Remix OS
7. Detailed procedure to make your custom Debian Remix OS
8. Screenshots of the Remix build process

If you want a computer magazine's opinion of why a live boot Linux OS might be useful, see:

> https://www.computerworld.com/article/2519158/linux/5-ways-to-use-bootable-linux-live-discs.html

There are many pre-built live boot OS images available for download from the internet. The most easily found have been designed with a broad audience in mind, and include many popular features and applications. Most users of live boot Linux OSes use one of these official variants.

However, some of us are more adventurous (or picky) and want to try alternatives which work essentially the same way, but with different sets of features or applications which fulfill more personal interests. These are the potential users **and builders** of custom redistributions, otherwise known as **remix** operating systems.

The author (GLF) personally maintains several personal Debian Remix projects, that is, live boot OS images currently based on the stable (stretch) distribution of Debian. The remix uses primarily the Debian binary repositories and the officially supported live-build package to build an ISO image which, when copied to DVD-R+ or USB stick, can be booted to run a live OS without permanent commitment or installation to a computer's hard drive.

> (GLF Programming Environment download site)

Others have come up with alternate methods of producing custom live boot OS remixes, for Debian and for other Gnu/Linux variants. The author has tried a few of these methods in the past, and has not tried some others which might be available. The author does not pretend that this tutorial is the **only way** or the **best way** to produce such remixes. This tutorial merely attempts to provide instruction to anyone interested in a particular method proven to work by the author, by which **any** interested user can build such a remix on his/her own.

(go back to top)

Why a personal Remix OS is possible and useful

The concept of a free remix OS relies upon the underlying OS being inherently free software. One may not legally take a proprietary commercial OS (such as Microsoft Windows), copy and remix it, then distribute working copies to others. But with most Linux distributions it would be natural to do so, even if perhaps technically difficult for a user to assemble.

> https://en.wikipedia.org/wiki/Freely_redistributable_software

Remixing an OS is different from rebuilding an OS from scratch, that is, compiling all components from source code (that's pretty hard, though it can be done -- see http://www.linuxfromscratch.org/). A remix OS starts with an available free distribution (usually based on BSD Unix or Gnu/Linux) which provides an official working live boot release image (ISO file) which can be transferred to a bootable medium such as DVD-R+ or USB stick. The remix builder adds/subtracts software and data to/from such an image, and then repackages it into another image file for redistribution, personally or to others.

Such remixes primarily lean on the same established software package repositories supported by the main OS vendors to install working binaries into the remix. They may optionally include some code compiled from personal or externally provided source code, or custom configuration and other data specific to the remix.

The concept of a personal remix OS is attractive to any enthusiast of alternate operating systems who likes some aspects of current official offerings, but not others, or who has special need for certain preinstalled toolsets or perhaps unusual or less popular applications.

(go back to top)

Support for remix OS building by the official OS vendors

At one time or another, the various well-known Gnu/Linux distributions such as Ubuntu and Fedora have included utility packages or provided how-to support information on building custom remixes of their base releases. Some of this support has waned, projects have stalled or grown obsolete, and some OS vendors have become sensitive to trademark issues.

Examples:

> https://fedoraproject.org/wiki/Remix
> https://help.ubuntu.com/community/LiveCDCustomization
> https://docs.kali.org/development/live-build-a-custom-kali-iso
> http://debian-live.alioth.debian.org/live-manual/unstable/manual/html/live-manual.en.html
> https://en.wikipedia.org/wiki/Remastersys (obsolete)

As of this writing, Fedora supports the concept of a "Fedora Remix", and currently provides packages to assist building one. Kali Linux also seems to provide official support for remixes. The support looks good, but the author has no experience with either or Fedora or Kali, or derivatives.

The author's first attempts at remixing OS distribution was with Ubuntu (circa Ubuntu 9) and later (reacting to changes in the Ubuntu UI) with Linux Mint Mate. Ubuntu has evolved a mixed record of support for remixes. Ubuntu has repeatedly included, then pulled or made obsolete such packages, or its website advice has gone stale or rendered unworkable by design changes to the OS. More recently Canonical, Inc., the parent of Ubuntu, has taken a harsh stance toward unofficial remixes, requiring the stripping from any nonofficial distribution of any Ubuntu-named content, including logos and certain applications, even so far as to require recompilation from source **ALL** distributed binaries. Linux Mint, a derivative of Ubuntu, completely canceled support for, and discourages remixes.

The author has found the Debian OS platform to be a more hospitable starting point for building a custom remix OS which meets his personal needs. The Debian OS project (www.debian.org) officially supports the concept of "Debian Derivatives" (https://wiki.debian.org/Derivatives), and encourages (but does not require) their inclusion into their official "census" and QA process. That would be appropriate for a large-scale remix project aiming for mass appeal, but is probably overkill for a typical enthusiast who just wants to try a personal knock-off to serve a local need. So long as a remix creator does not try to pass off his product as an official Debian release, it is probably safe.

(go back to top)

Why Debian Mate

The author's impetus for developing a standalone OS derives from an ongoing desire to make accessible to clients the long-term maintenance of C/C++ programs he has written for others. C/C++ source code, while generally portable in nature, leans upon an assumed set of compiler tools and libraries which might be difficult for someone else to find and install into an integrated whole support structure capable of rebuilding a given project from source code.

The author first came up with a comprehensive, installable package of compilers and utilities to allow a user to build programs on a Windows computer. That addressed the need for Windows users. When the author eventually ported to Linux, he found a rich set of equivalent tools readily available, but the process for getting all of the needed ones in place was far from simple. Every time he tried a different OS, or new release version of an OS, he had to relearn the differences and reassemble the toolset. He wanted to make a portable, live bootable image which already contained everything needed, and which a potential user could build any of his programs, immediately and without any other commitments. As a bonus he wanted the live OS to be optionally installable, with all the live session apps and configurations, to a computer's hard drive as a permanent single or dual-boot OS.

The author tried various OS variants, Ubuntu, Mint, then finally Debian. Each time he had to adapt build scripts found on obscure or obsolete websites, and deal with changes to the underlying OS structure or required locations within the filesystem for key data. Finally, he found a reference to the "live-build" package supported by Debian, and immediately saw the advantages.

Changes in user interface design (e.g. Gnome changing from 2.0 to 3.0, and Ubuntu's adoption of Unity as its preferred UI) in the past few years have caused a variety of compatibility issues for older code developed by the author. Since Mate is a fork of Gnome 2.0, it was more immediately compatible with the author's codebase, originally ported under Gnome 2.0.

https://mate-desktop.org/

In particular, the Mate flavor of UI has matched the comfort level of the author for user interface and applicability to his personal needs, and has proven itself as an environment to compile and test run various C and C++ programs and libraries from various personal and open sources.

If the user is interested in the author's own remixes (GLFPE for Linux), introductory videos can be found by searching Youtube for "CustomOSMaker", or directly going to:

GLFPE Tutorial Series

The video descriptions include links to the download site.

(go back to top)

The nature of the Debian live-build process

The live-build process rebuilds an ISO image on the fly by reliably constructing a chroot environment (sort of a local protected "pretend" version of the parent OS) and downloading all packages required during the build, based on a relatively small set of configuration files prepared ahead of the build.

All packages downloaded during live-build are the latest available at the time, so it is easy to generate an upgraded disk meeting the same specs by simply rerunning the build at a later date. All packages which have been upgraded, including kernel packages, are used in the new build. The configuration scheme supports a variety of customization features, including provisions for installation package lists, installation of local packages (*.deb files) not found in the repositories, hook scripts to perform special scripts at key parts of the build, and data to override default data both inside the live boot OS filesystem and outside it on the produced DVD/USB medium. It is very versatile.

The Debian live-build process is not only the easiest process found by the author to build a live boot OS, it is by far the most reliable. The live-build package is (or at least HAS BEEN) the officially supported tool used by Debian itself to build its own official releases. Once you get a configuration fileset to meet your requirements and successfully build, the same configuration will usually work again and again over time. The ISO image includes an installer that JUST WORKS.

The optional installer, run from the live boot initial menu, has proven to work repeatedly on the author's test computers. However, the author still prefers to use live boot of his own remixes for everyday work, especially for secure web surfing and banking (no caches, cookies, or other memory left behind between boots).

(go back to top)

The awkward recent history of the live-build project

The author (GLF) first discovered the availability of "live-build" about November 2015. By sheer coincidence, that was almost exactly the time when events behind the scenes of the "Debian Live" project were coming to a head. It seems that the official Debian CD distribution maintainers were unhappy with the developer and maintainer of the "live-build" project (Daniel Baumann), and had chosen in November 2015 to "demote" his status from developer to maintainer, so that the Debian project could take over the lead on the project. Much rancor ensued, and Mr.Baumann quit the project officially, and shut down his servers for the Debian Live project, leaving only archives at the state of his final involvement at the time.

For a complete discussion of the awkward transition and its implications, see:

https://adrian15sgd.wordpress.com/2015/11/15/debian-live-ng/

The author of the above discussion is the developer of "Rescatux", and whose product depended on "live-build", so it is the point of view of a noncombatant but interested third party.

Up until that time, the Debian Live project was the official methodology for Debian to produce its live boot ISO images. It just so happened that the author of this tutorial (GLF) had stumbled across this final state of configurations at the same time, and proceeded to have much success producing personal remix images based on the initial configurations found in the archived live-mate-image package files, which were completely valid for the extant version of Debian 8 (jessie). The author was unaware at the time that the Debian project was officially moving away from Mr. Baumann's methodology and developing their own. Apparently by the time Debian 9 (stretch) was released, the official ISO images were being created by a replacement suite called "live-wrapper".

Unfortunately to this date the author has not found online any specific published process for producing duplicates of the Debian 9 ISO releases. The original Debian Mate (and other UI) configuration sets are still available in multiple places online and within the repositories, but are now anachronistic. The live-build project itself has continued to exist in the repository, but with a new maintainer (Raphael Hertzog). It has been officially deprecated as official release methodology, but NOT deprecated from existence. The project has continued to be supported as an alternate methodology, as other users continue to build ISOs with it. The version of live-build and live-config has changed enough that older configuration sets will not work without modifications, but if you follow the new rules (from a new manual), it

still works. It is uncertain whether the package will retain relevance beyond Debian 9, but so far development of live-build is currently continuing under the "testing" branch, so presumably it will also be found in the next version of Debian to be released.

The author continues to find live-build a suitable, reliable methodology for personal remixes, and has so far been unable to get live-wrapper to work. Therefore, the remainder of this presentation will assume the use of live-build, and describe suggested alterations to the (old) official Debian Mate configuration files to get a reasonably close facsimile of the official release to successfully build using that altered configuration set.

(go back to top)

The basic method to make your custom Debian Remix OS

This method "bootstraps" the new remix OS without actually permanently installing the originating OS to any host computer. if the build is successful, you use one live boot OS disk, a host computer, and an external temporary work disk to generate a new image file representing the remix live boot OS disk.

1) Select and acquire an official live boot Debian OS image and learn to use it.

2) Acquire and/or create a configuration set to specify how to build your remix.

3) Prepare a suitable work drive onto which you will build the support structure and the final ISO image.

4) Boot the live boot image, load the work drive, install the prerequisite software during the live session, and perform the build process (may take several hours).

5) Burn the resulting ISO image and test/use it.

(go back to top)

Detailed procedure to make your custom Debian Remix OS

Get an official release (and learn to use it)

You will need to acquire a compatible live boot OS and run it from a DVD or USB stick. The procedure can best be illustrated when starting with an OFFICIAL Debian release, but it can build an upgraded Remix using any CPU-compatible Debian (or Debian remix) using the same CPU type and OS version (currently Debian 9 stretch). Your remix can bootstrap its own upgrade. The following assumes the user starts by downloading the official Debian release, to demonstrate building up from first principles.

1. Using any suitable computer and operating system, go to https://www.debian.org. Choose and download the ISO image (https://www.debian.org/CD/live/) of the current release of a selected flavor of the live OS (currently cinnamon, gnome, kde, lxde, mate, and xfce). Different flavors use different GUI interfaces, each with advantages and disadvantages, Pick a UI which matches your preferences. Try several if you want. Since the author prefers Debian Mate (the basis of his personal remixes), that will be assumed for examples in the remainder of this tutorial.

   You need to match the CPU architecture of your intended computer. You can choose amd64 if you want to support 64-bit processing, or i386 (actually i686) for 32-bit processing. You can always do 32-bit processing on a 64-bit computer, but not the other way around. For the remainder of this tutorial, the author will assume the "686" flavor of i386 will be used.

The current i386 live build image at the time if this writing is available from:

https://cdimage.debian.org/debian-cd/current-live/i386/iso-hybrid/

and the author will choose to use the file:

debian-live-9.3.0-i386-mate.iso

to illustrate the methodology to follow.

Screenshots

2. Burn the ISO image to a DVD-R+ (it's too big for CD-R) or to a USB stick using utilities readily available for Windows or Linux.

Screenshots

3. Boot the live OS, WITHOUT INSTALLING IT. Your computer will need to allow you to do this, or all bets are off. You may need to disable Secure Boot, enter Legacy (BIOS) mode, something else, or perhaps nothing special, depending on your computer's brand and age. (You'll have to figure this one out for yourself).

In order to run the OS (and eventually the remix build process), you will need a PC compatible computer with either an AMD64 OR x686 processor and at least 2 Gb of RAM. You can run the "i386" version on an AMD64 CPU because the instruction set is compatible, but not the reverse. The official "i386" release of Debian stretch now actually requires at least a 686 CPU to run. Very few Linux distributions support even 486 CPUs anymore, none 386. There is partial support for 586 processors within Debian, but it tends to break due to applications assuming 686 machine instructions. It is possible that the **actual** required CPU type required will become 686-pae (extensions to 686 set) in the near future, but for now it is possible to reliably specify 686 "flavour" for the build. The author chooses to specify "686" in an attempt to make the ISO applicable to the widest variety of systems. Time marches on, and older machines eventually go unsupported by OS vendors and application developers. Sigh.

Screenshots

4. Try out the OS, and get comfortable with it. If you are new to Linux, you will need to figure out what you like and don't like before you can design your own remix. Try things out, figure out how to improve things when possible. In a live boot system, you can install many packages and try them out, until you run out of available memory. If anything goes wrong, don't panic. For live boot, everything goes away with the next reboot. If you don't already have it, install the "synaptic" package -- it will help you find the packages you need. Make a list of package names you want to add to your remix.

Screenshots

Set up a configuration file set

The rebuild procedure starts with a configuration file set. The live-build system includes some simple configuration building utilities to get an arbitrary base configuration started from scratch. The standard procedure to create a base configuration repeatedly invokes the "lb config" command with a variety of command-line arguments, or a VERY LONG "lb config" command with LOTS of arguments. See https://manpages.debian.org/stretch/live-build/lb_config.1.en.html.

Since the official Debian releases were originally based on live-build (before 2016), and Debian still retains those configuration sets as open software, the original compatible configuration files are readily available from the repositories, **THOUGH THEY ARE ACTUALLY OBSOLETE AS IS** (see above history of live-build). Because these filesets

once described the official release content specifications, they represent approximations to the current live image releases.

One may manually edit the basic configuration files and the folder contents, but any errors will probably cause failure of the build, so be careful (if all else fails, read the live-build manual: http://debian-live.alioth.debian.org/live-manual/unstable/manual/html/live-manual.en.html).

One may personally craft an intricate configuration set including special data files and hook scripts to override the defaults, delete unwanted packages included in the original build, and/or augment the packages available from normal sources with special build processes performed on the fly. Those methodologies have been incorporated into the author's own remixes comprising the GLFPE for Linux (GLF Programming Environment), and those configurations are available from the author as well as matching finished ISO images.

(GLFPE download site)

However the purpose of this tutorial is to show how to make a **personal** remix. If a user doesn't need to override default data and only wishes to augment the packages installed to suit a personal need, then it is much simpler to start with an already known working configuration fileset, especially one derived from a past official release, then edit the content as needed to prepare the specifications for a custom remix.

5. When you know what you want and are ready to try building a personal remix in a live boot session, download the live-image package for the base configuration of the release you are running (e.g. cinnamon, kde, mate). If you don't already have it, install the "synaptic" package -- it will help you find the packages you need. You will need:

live-build

and ONE of the following, matching the flavor you want to remix:

```
live-image-cinnamon-desktop
live-image-gnome-desktop
live-image-kde-desktop
live-image-lxde-desktop
live-image-mate-desktop
live-image-xfce-desktop
```

Install the live-build package (if not already installed).

Install the selected live-image--*desktop package. It will install (since it's live boot, temporarily) a number of files. Use the file manager to look under /usr/share/live/images/ for the name of your chosen desktop (e.g. mate-desktop). That folder contains an* incomplete* configuration set for the chosen UI flavor. Copy that folder to the home (~) folder.

Screenshots

6. Bring up a terminal, and change to that configuration set folder (e.g. ~/mate-desktop). The configuration is incomplete, because the developer of the official build expected to auto-configure some of the content using certain assumptions and built-in scripts. Execute the following command (part of live-build):

lb config

The script will auto-configure the tree and populate it with basic configuration files matching the selected flavor of desktop. Look under the configuration set folder for a subfolder "config". You should see a list of folders and six plaintext config files (binary, bootstrap, build, chroot, common, source). These contain overall specifications for the image build, including name, type of CPU, etc. You should change a few of the entries, at least to change the name of your remix distribution. Look for "build" file, and change the Name

entry from "live-image" to "live-image-remix001" or similar to identify the name of the eventual ISO file in a manner unique to your remix.

Screenshots

7. The configuration as it stands WILL NOT WORK as is, but it is a good starting point. To get to a minimum configuration which will be buildable and provide MOST of the base packages expected in the official Debian Mate release, you must alter some of the configuration files and folders as created above. The fact that the live-build project uses new rules and that the configuration set just recreated is obsolete (from 2015), means that some editing is required, or the build will fail (without meaningful helping error messages to help debug).

The author (GLF) has worked out by trial and error the minimal changes necessary to get a successful build which mostly matches the official Debian Mate release (some are not technically minimal, but author preferences):

"binary" file

Change the LB_ISO_VOLUME= parameter to "Debian32REMIX001" or similar name. There is a limit to the name length, so stay within the above example bounds. This is to keep from using Debian's original chosen name for your remix.

Change the LB_MEMTEST= parameter to "memtest86+"

"build" file

Change the Name: parameter to debian-live-remix-001. The number (001) in this should be consistent with the volume name edited above.

Optional -- change the Archive-Areas: parameter from main to main contrib non-free

That above change will allow bringing in packages from repositories other than Debian's own "free software only" repository (main) which by default is the only one enabled. This should not matter for a minimal demo remix, but for some later personal remixes, could be essential.

"chroot" file

Change the LB_LINUX_FLAVOURS= parameter from "686-pae" to "686". This is the author's (GLF) choice to make the remix have broader applicability, but you can choose "686-pae" if your opinion differs. Of course, if this is a 64-bit build, use "amd64" (and run on a 64-bit Debian image version).

auto folder

Delete all the files under the auto folder. Once you have performed the initial "lb config" (above), they have outlived their purpose and are no longer useful, and get in the way later.

package-lists folder

There are a number of list files created during autobuild. The only essential ones are "desktop.list.chroot" and "live.list.chroot". The "desktop.list.chroot" file is key, because the packages listed therein will be installed to the live boot image. The main line which determines the UI flavor selected is the line "task-mate-desktop". That package is actually a so-called metapackage, which specifies a list of other packages to install, and makes up the bulk of what constitutes the Mate UI flavor.

There are two files, which if left intact WILL KILL THE BUILD, and the messages won't really tell you why. Those files are "dkms.list.chroot_install" and "dkms.list.chroot_live".

The latter file does contain three lines:

```
open-vm-tools
open-vm-tools-desktop
open-vm-tools-dkms
```

which may be optionally copied (appended) into the list "desktoop.list.chroot". If you try a personalized remix with you own extra packages, you will also append your personally selected package names to the text in that same file.

Delete both the "dkms.list.chroot_install" and "dkms.list.chroot_live" files. If you don't, the build **will not work**.

bootloaders folder

There are four subfolders under this -- delete them all, or the build will not work.

hooks folder

This folder points to various scripts which run at particular points during the build, but they are incorporated as links. None of these hooks are essential to a minimal build, and all the links can can be deleted entirely.

If you choose to leave the hooks in place, delete the links, then go to the original installation files (/usr/share/live /images/mate-desktop/config/hooks) and copy the actual hook files back into the locations where the links were deleted. Note that what is under "live" and "normal" folders is important to the newer version of live-build found in Debian 9.

By copying actual files instead of links, you can maintain the hooks within a permanent archived configuration set, and possibly modify some or all of them for a special purpose. You might later wish to insert your own hook scripts into the mix to accomplish special tasks during the later build of your more personalized remix. Whatever else, leave the "live" and "normal" folders intact. The current version of live-build looks in both of those folders, and at least one hook script **must** be in each, even if the script does nothing important (an arcane fact learned the hard way).

The above changes to the configuration file set should go most of the way to providing a match to the official Debian Mate release configuration. The author (GLF) has found by experiment that a large list of other packages are included into the current official Debian 9 release that will **not** be installed by this minimal method (apparently they were not in the Debian 8 release). Most of those additional packages seem to be associated with support for languages other than English, and can be considered optional.

However, the author has found by experiment that appending a complete package inventory list extracted from the original Debian 9 release to "desktop.list.chroot" will fill any functional gaps, and come very close to replicating the official distribution (complete with original full multi-language support).

You **cannot completely replicate** the official Debian live distribution using this method, because the official live distribution is no longer built (since 2016) using "live-build", but instead with "live-wrapper". The specifics of configuring such an equivalent "live-wrapper" image are not published. However, the ISO image produced using live-build and the configuration set prepared as described above should work just as well for a personal Debian Remix. All the software installed this way into the remix is still official Debian software, or supported contributions to Debian "contrib" or "non-free" repositories.

Screenshots

8. Here's where YOU can make a difference and customize your remix by adding additional packages. Look for the subfolder of "config" named "package-lists". Look under it for "desktop.list.chroot". Bring it up in a text editor. You will see a list (example for mate-desktop):

task-mate-desktop

task-print-server

plymouth

plymouth-x11

gstreamer1.0-vaapi

i965-va-driver

Optional customization:

If you have established a list of packages you want installed, which you know aren't in the official release, type them in at the end of this list. Be sure the package names are correctly spelled. Any error will stop the build cold. The names are the same ones you will find when looking up packages in Synaptic. When the list is complete, save the file.

You can make other changes to other configuration files (arcane procedures documented elsewhere by the author), but adding packages in "package-lists" is the minimum action you can take to make a REMIX, a live boot OS DIFFERENT from the original.

Screenshots

9. SAVE YOUR CONFIGURATION! use the file archiver utility to compress the configuration set folder to a .tar.gz file (that format retains all Linux permission flags). Name the output compressed file with a name to uniquely identify your remix. Be sure to save the .tar.gz file to another drive, such as a USB stick (remember the home folder will disappear on reboot).

Screenshots

10. You can reboot the computer if you wish, because the configuration set you have made and the live-build package (and its dependencies) are all you should need to start a rebuild later.

Screenshots

Build the Remix ISO

For the build itself (GLF method) you will need an empty USB stick of at least 32 Gb capacity. This procedure avoids using internal hard drives to prevent accidental damage to the computer's installed OS. You will need good broadband internet access UNINTERRUPTED throughout this procedure. The build process is only reliable with a good internet connection. It tends to break easily on any error, which includes being unable to download a needed file.

11. Live boot the Debian OS you wish to use to build the remix. If you are building a 32-bit OS, be sure to run a i386/i686 version, Likewise for amd64 -- the build computer must run the same CPU type as the build. Since this is a fresh live boot, you will need to reinstall "live-build". If you are using the example Debian Mate OS, then the author recommends also installing "caja-gksu" and "caja-open-terminal". Those will come in handy later. If you have not yet prepared a work drive formatted as EXT4 (see below), you

should also install "gnome-disk-utility". All of these packages take up minimal "disk" space (actually RAM in live mode), so you should have plenty of room available to perform the build later. None of these temporary live installations affect what gets installed into the built image, though the author considers including them into "desktop.list.chroot" a good idea.

Screenshots

12. The empty 32 Gb USB drive needs to be reformatted as a Linux EXT4 disk. This can be done by one of several methods. The author recommends using the "gnome-disk-utility" package to format and reformat drives. You can use GParted, but it creates drives in root mode, which end up with root permissions on the drive, and that complicates things. Presumably it can be done with fdisk, but the author doesn't do that, and won't presume to tell you how. Format the drive to EXT4 format and give it a name (label). The drive needs to be a Linux format because of certain permission settings you will need to set later.

Screenshots

13. Unpack the before-created compressed configuration set to this new Linux drive. You will execute the build from inside the configuration set folder in a command window. Make sure you have at least 18 Gb free on the drive, or the build might fail by running out of disk space.

Screenshots

14. You need to perform a special procedure to remount the new Linux EXT4 work drive so that special access permissions are granted to the drive. First, determine the drive name of the new Linux work partition. Look the drive up using the graphical gnome-disk-utility, or use a command window to run:

```
sudo fdisk l
```

to list all drives. You need to identify the Linux work drive device name (e.g. /dev/sdd1). **GET THIS RIGHT!** The next step will use this drive name to remount the drive.

The author has provided in the GLFPE website archives a bash script (remountforlb.sh) to do this, but you can alternatively do it manually.

If you use the author's script, unpack or copy the script file "remountforlb.sh" to the home (~) folder (it must have execute flag set). Change to the home folder (~) in a terminal window, and assume for this example the Linux drive is /dev/sdd1. Type in:

```
./remountforlb.sh /dev/sdd1
```

You must perform the procedure from the home folder, not just from the work drive, or it will not complete properly. As the drive remounts and the script file "disappears" from its original mount point, the script must keep going to finish the remount, but that won't happen if you run the script from the same drive.

Alternatively, perform the equivalent process manually. Change to the home folder (~) in a terminal window, and assume for this example the Linux drive is /dev/sdd1:

```
sudo umount /dev/sdd1
sudo umount /home/$USER/livebuild
sudo rmdir /home/$USER/livebuild

sudo mkdir /home/$USER/livebuild

sudo chown -R $USER /home/$USER/livebuild
sudo chmod a+rwx /home/$USER/livebuild
```

```
sudo mount -t ext4 -o rw,nosuid,dev,exec,uid=$USER,gid=$USER,flush /dev/sdd1
/home/$USER/livebuild
```

This should unmount the drive, create a new folder under home folder named "livebuild", then mount the disk again under that folder, with new permissions necessary for the build to succeed (dev,exec). The normal file manager auto-mount does not meet the same permission requirements, hence the remount.

Screenshots

15. Ensure you have a reasonably fast working broadband connection (preferably wired) before you start the build. It will take several hours and any break in connection can kill the build.

Start the build process. Change to the configuration set folder in a terminal. If you use the command:

```
ls
```

you should see at least the following folders:

```
auto config local
```

When ready, type:

```
sudo lb build
```

and wait for 2-4 hours, depending on the speed of your computer.

The process is somewhat fragile in that any error will kill the build, and you will have to start over -- it doesn't usually recover from errors. On the other hand, if the connection is good and there are no configuration errors, it almost always works. If it does, you will have a new ISO at the end. If it dies early, diagnose the error and try again.

Screenshots

Aftermath

16. The build process consumes about 15 Gb within the configuration set folder, and many of the files will be set to root ownership. You will need to bring up a root access (Administrator) file manager or superuser command window to manage permissions on the ISO file. Change the ISO file's ownership back to the normal user, and move it back to the home folder. After that, you are free to delete all remaining content under the configuration set folder to free up the space.

Screenshots

17. Test the new ISO by burning it to DVD-R+ or by using an appropriate utility to copy it to a USB stick, just like you did with the original Debian ISO image. Linux utilities for making bootable media may be installed from the repositories (the author includes Brasero and MultibootUSB in his remixes). If you wish to use a Windows burning utility for burning to DVD or USB, first copy the ISO file to another FAT32 USB stick, since Windows cannot deal with Linux-formatted EXT4 disks.

Screenshots

18. Scan the new distribution for installed applications to produce an inventory. You can use this inventory to verify that intended applications or libraries were actually installed, or to compare the list to prior inventories of other live boot disks.

The author has provided in the GLFPE website archives a bash script (5_package_inventory.sh) to do this, but you can alternatively do it manually.

If you use the author's script, unpack or copy the script file "5_package_inventory.sh" to the home (~) folder (it must have execute flag set). Change to the home folder (~) in a terminal window. Type in:

```
./5_package_inventory.sh
```

Alternatively, manually perform the equivalent commands:

```
dpkg-query -W > filesystem.packages
```

```
dpkg-query -W -f='sudo apt-get -y install ${Package}\n' > package_sudo_install.script
```

You will find two inventory files in the home folder. One is a list of package names and version numbers, the other is a script fragmentwith apt-get commands to reinstall every package in the list.

Screenshots

(go back to top)

Screenshots of the Remix build process

All of the script files specified as part of the process below have been documented above in this tutorial, and are provided as complete script files on the author's GLF Programming Environment (GLFPE) website:

(GLF Programming Environment download site)

The process illustrated below follows the tutorial above, and the screenshots were derived from an actual successful performance of the described procedure.

1. You must start with a compatible Debian (Mate for this example) boot disk. If you have produced a compatible Remix before, you can use that to bootstrap its own upgrade, and skip to step 8 or 9. The way to fulfill this requirement when starting from scratch is to download the original Debian ISO Image drom www.debian.org and burn it to DVD or USB bootable media.

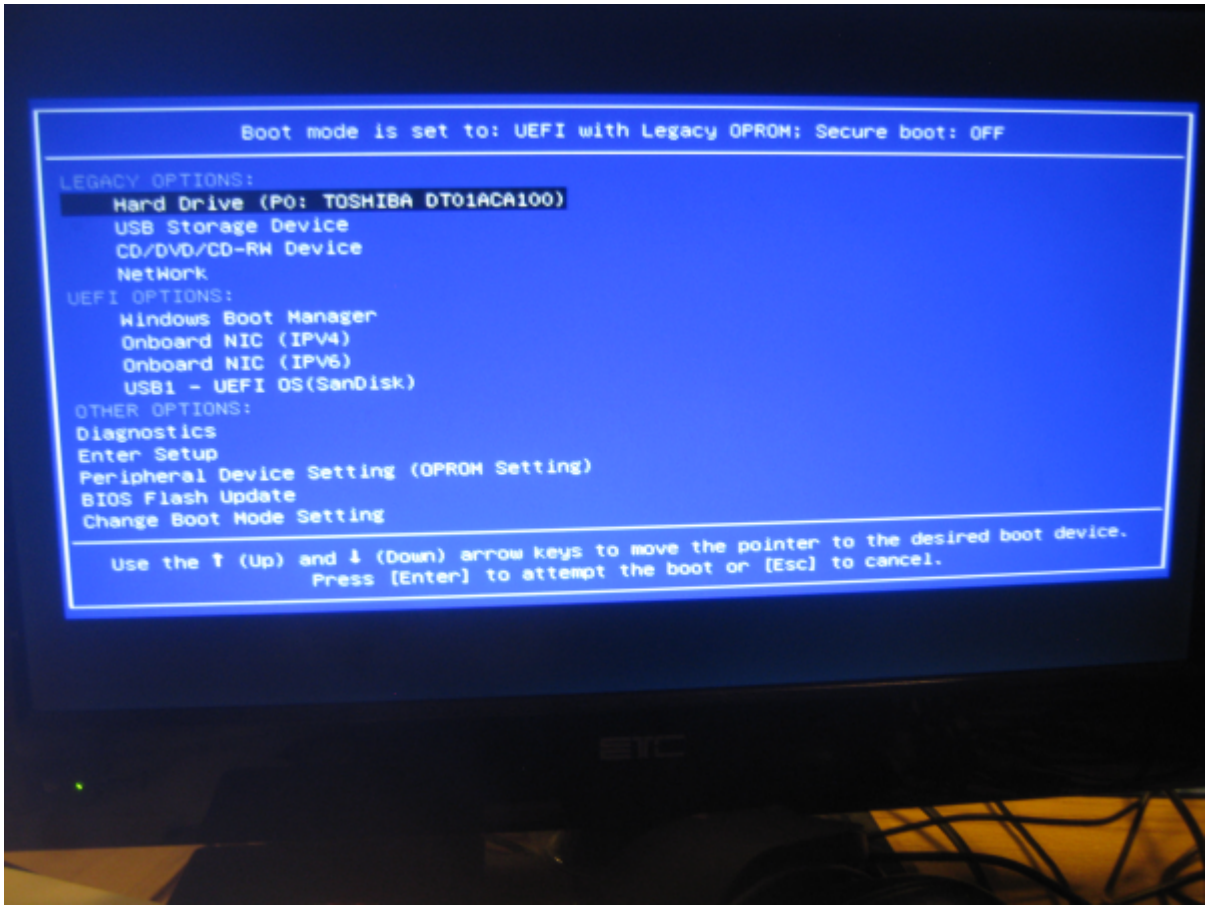
(go back to narrative)

2. Burn the ISO image for the current Debian (Mate assumed) live boot release. You can use any version of Linux or Windows to burn the ISO to DVD or USB using appropriate free utilities.

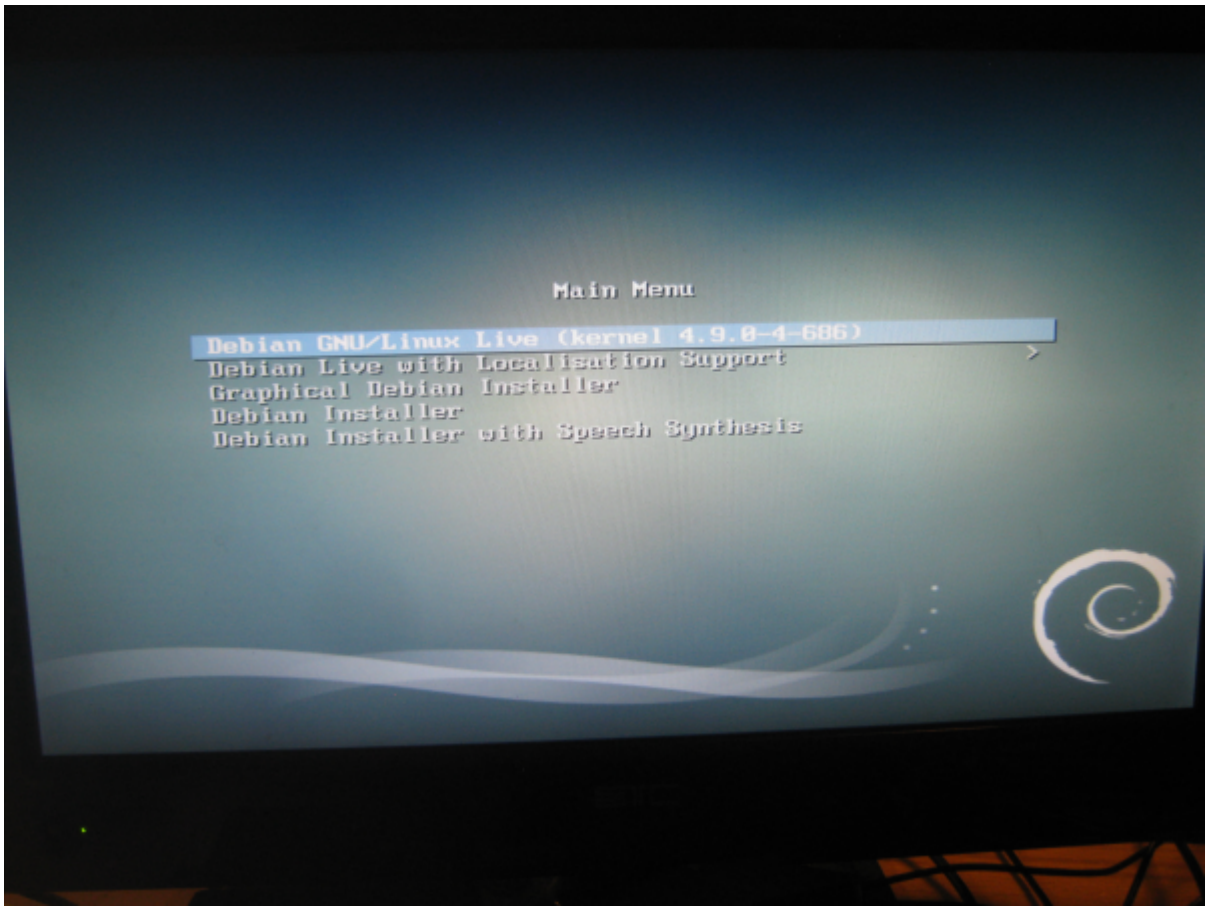
(go back to narrative)

3. Boot the live OS, WITHOUT INSTALLING IT.

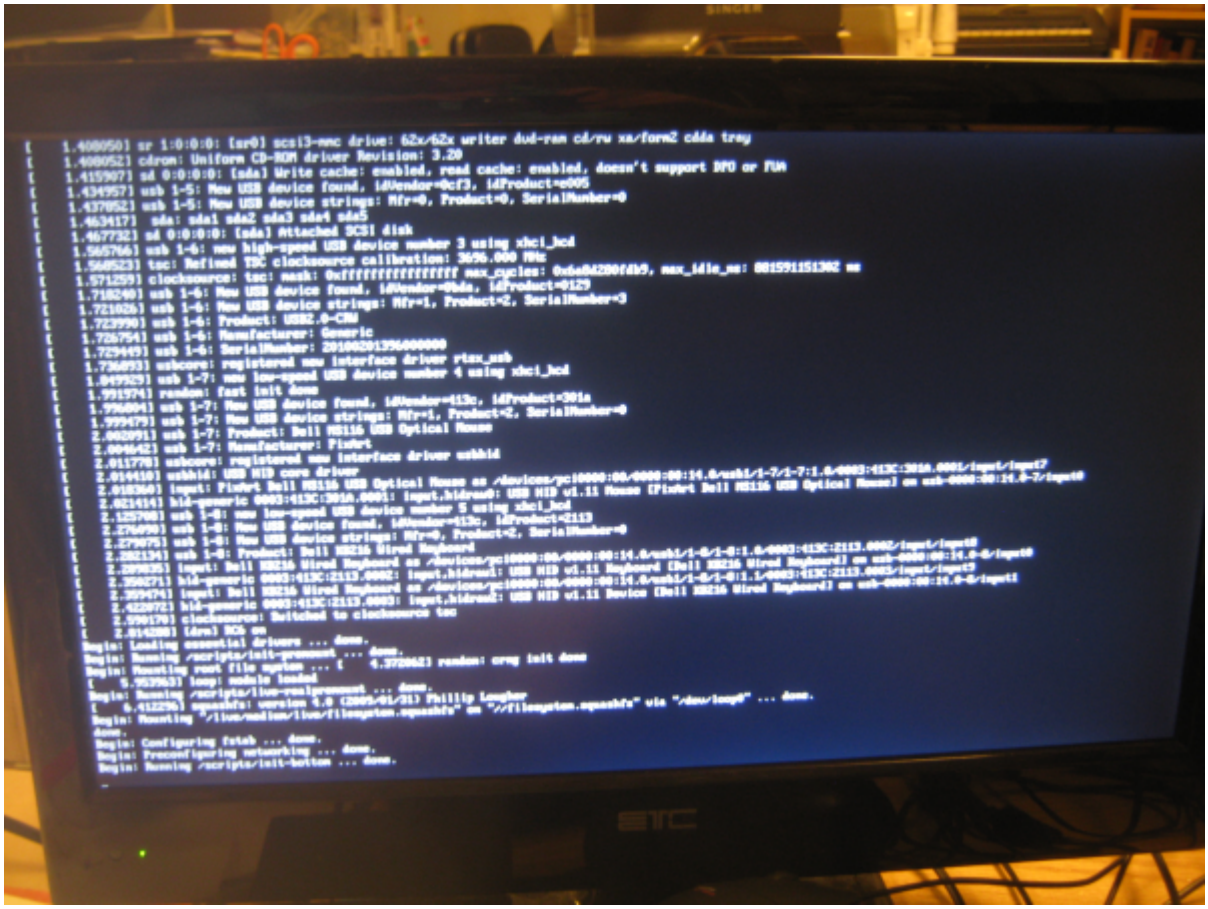
Insert the bootable DVD (or USB). Power up computer. Within seconds, repeatedly press specfied key to get boot menu.



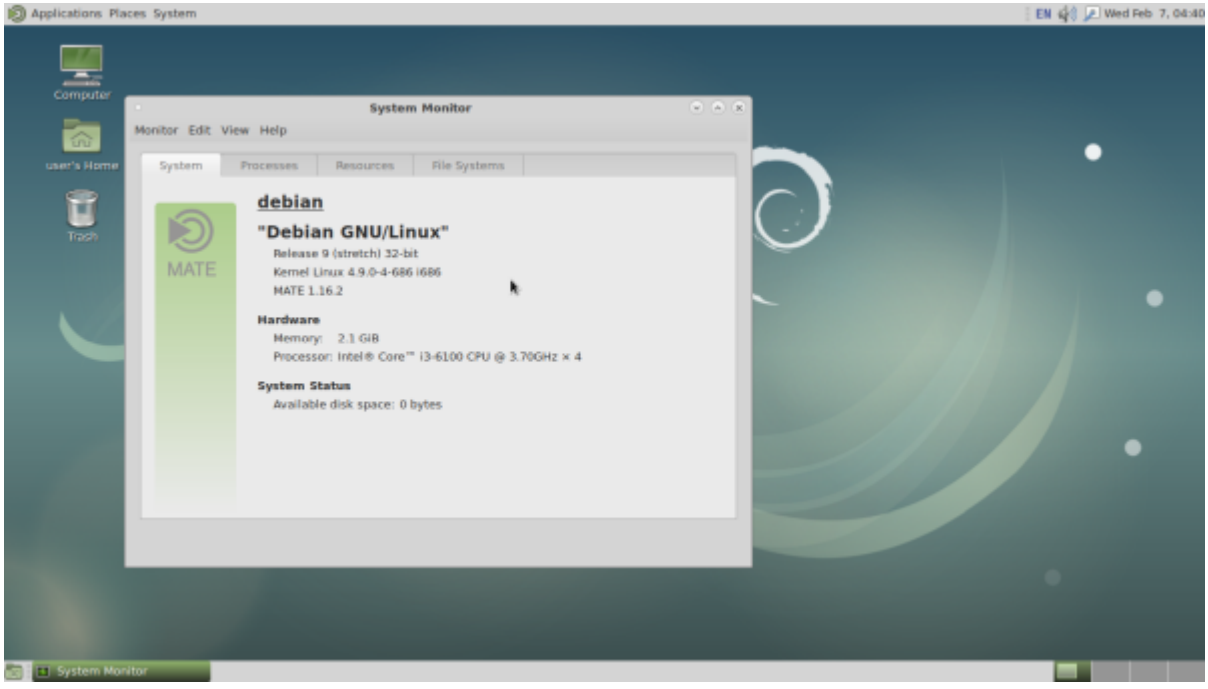
Main boot menu is displayed. Select normal (default) live boot mode.



Boot continues until GUI screen is displayed. Set location and time zone if desired (otherwise assumes UTC).



Once booted, verify the version of the Linux kernel and supported CPU by bringing up the Mate Sytem Monitor.



(go back to narrative)

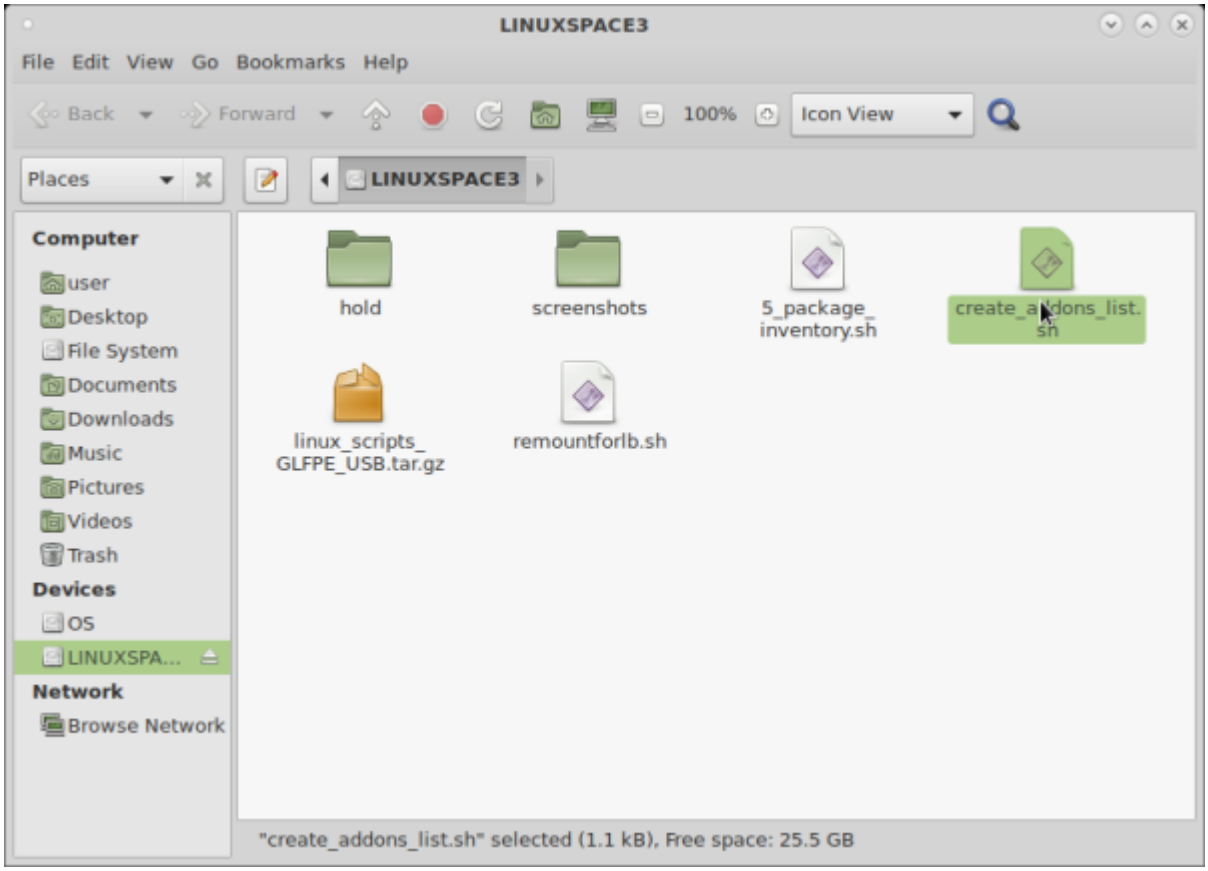
4. Try out the OS, and get comfortable with it.

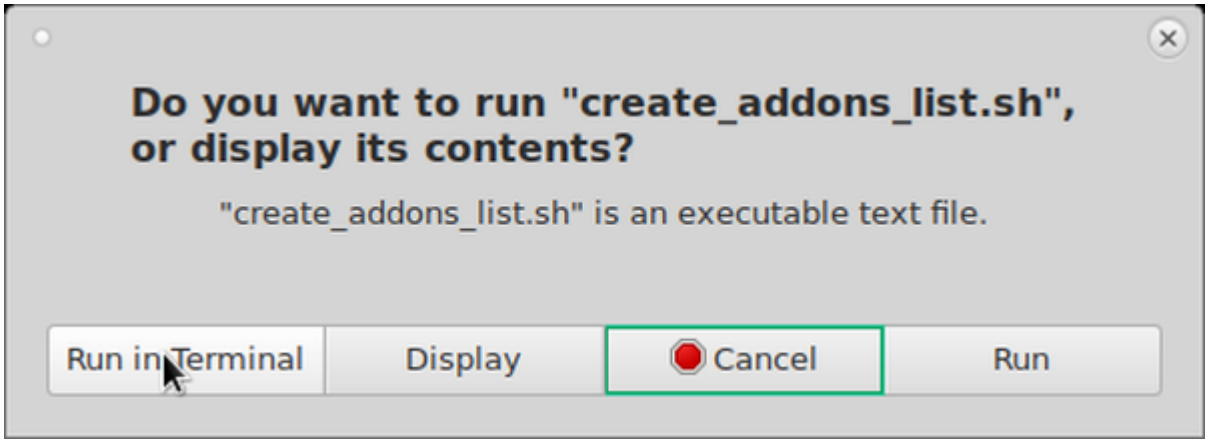
(go back to narrative)

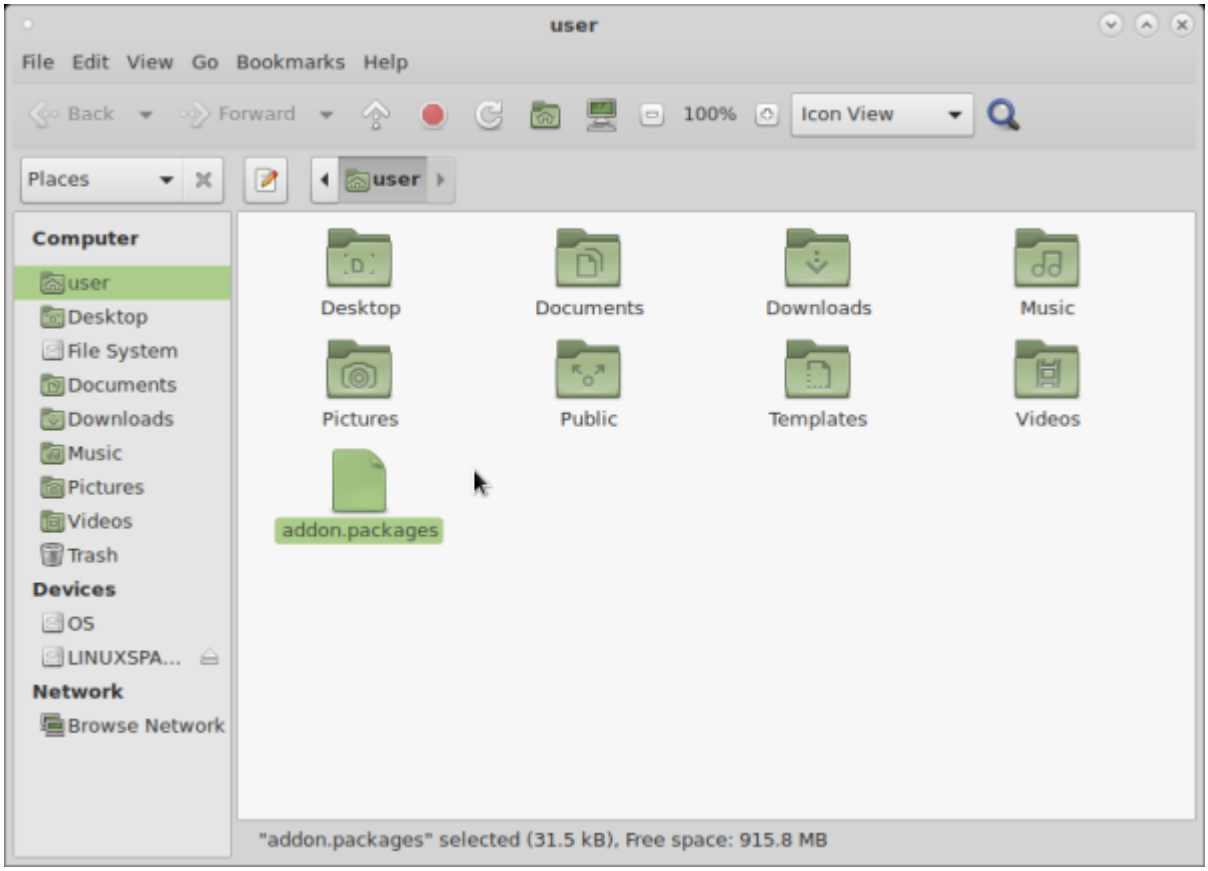
5. Download the live-image package to get the configuration fileset for the UI selected.

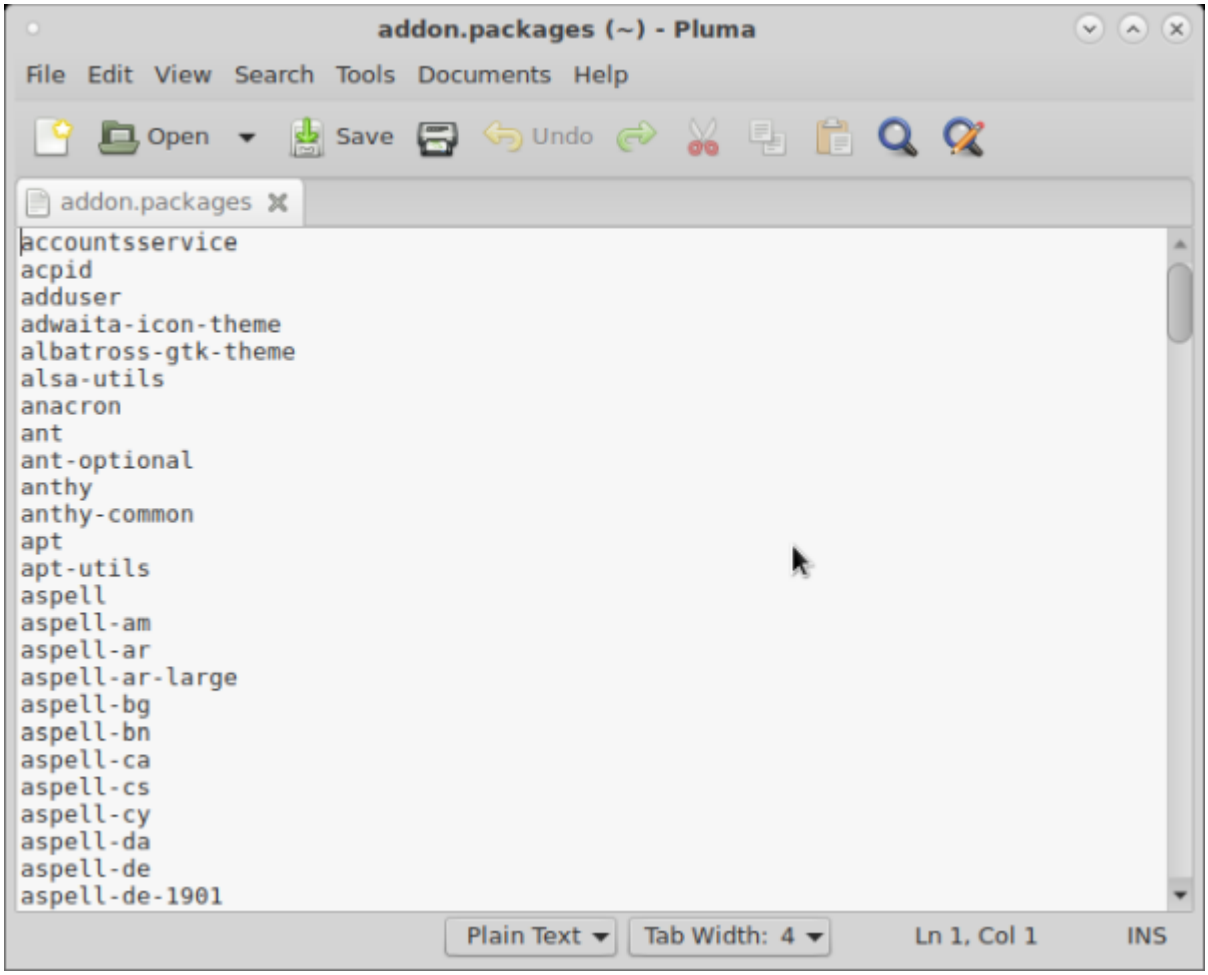
You will need a configuration file set. You may acquire one from prior work (the author provides examples in the archives of the GLFPE website), you can generate one from scratch by careful application of the rules found in the manual for "lb_config", or you can start with a prior known working set published by the original Debian Live project. The latter is what we will illustrate below. Because the Debian Live (Nov. 2015) configurations have been obsoleted by the current version of live-build, we will systematically modify the files and folders as shown below, into an experimentally proven working set. The author also provides a finished version of this configuration set (remix001_i386.tar.gz) on the GLFPE website to save time.

As a preliminary step, scan the existing OS for an inventory of installed package names. This illustrates use of a prepared script (create_addons_list.sh) to perform this task.

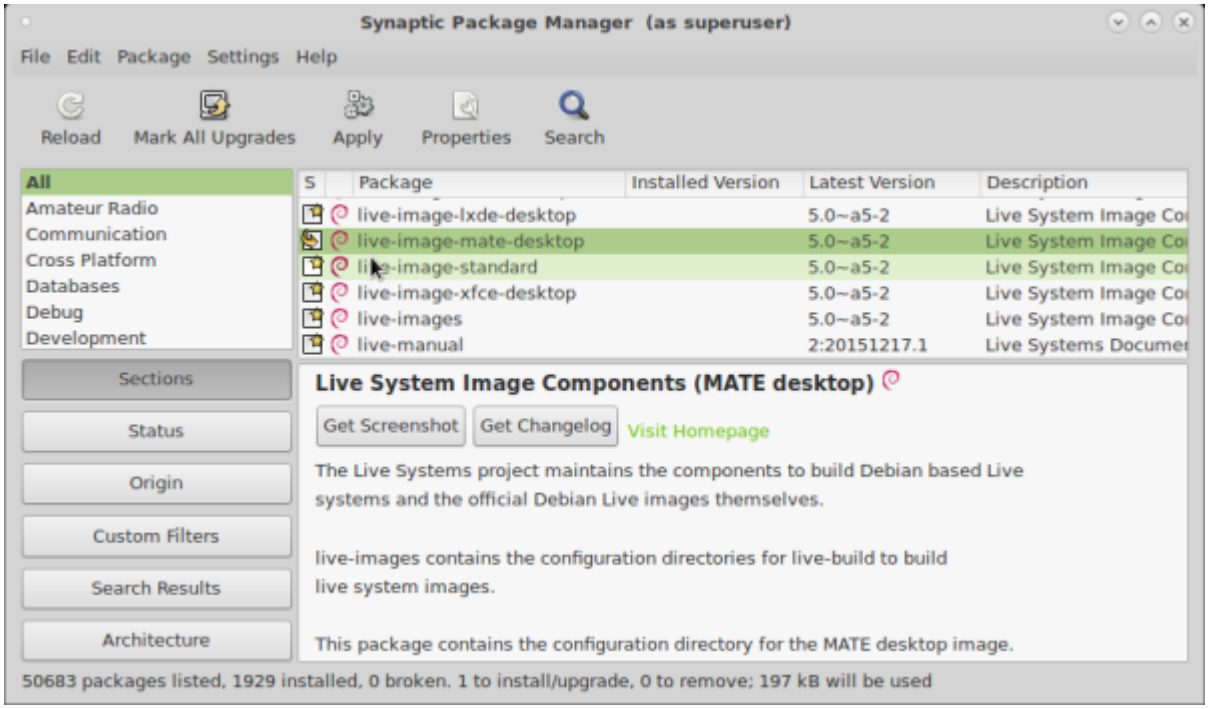


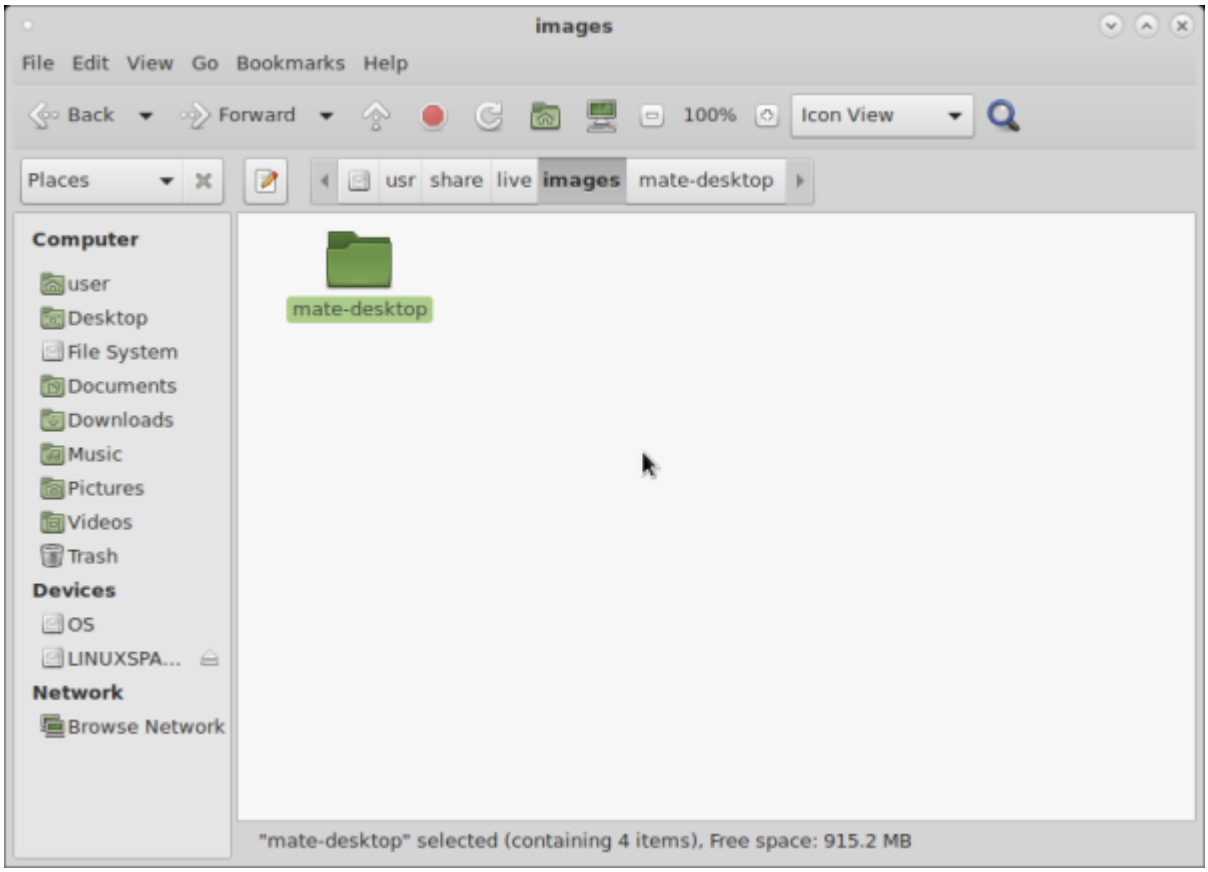




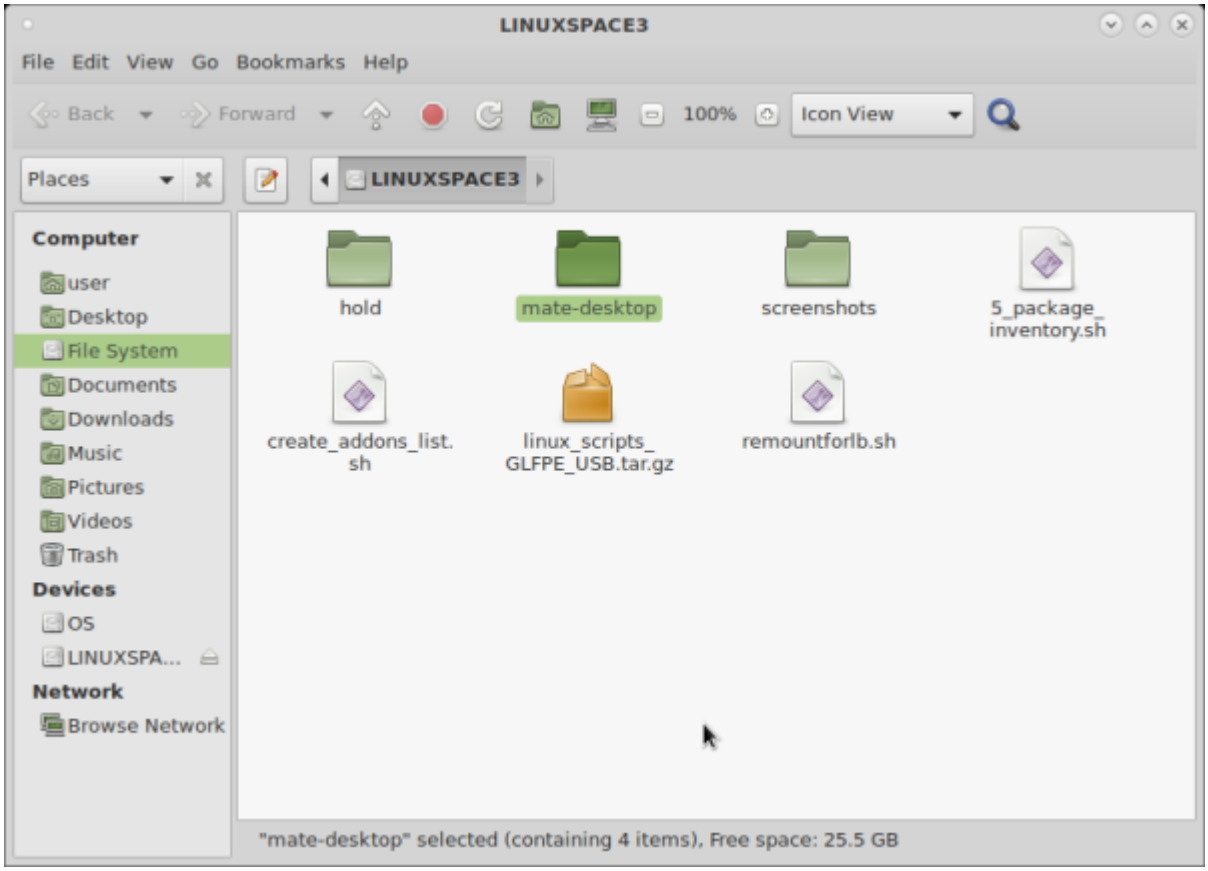


Use Synaptic to download the (2015 version) Mate base configuration fileset (live-image-mate-desktop).

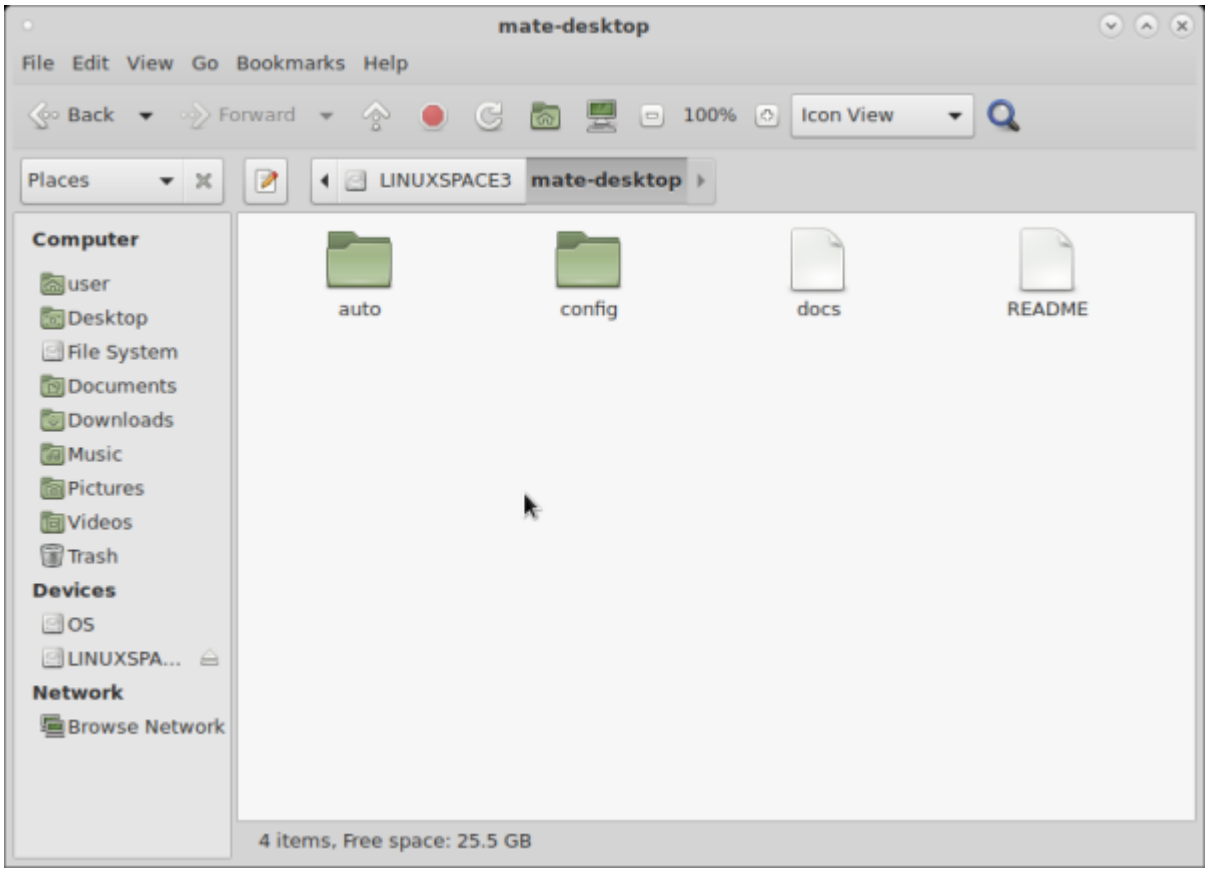




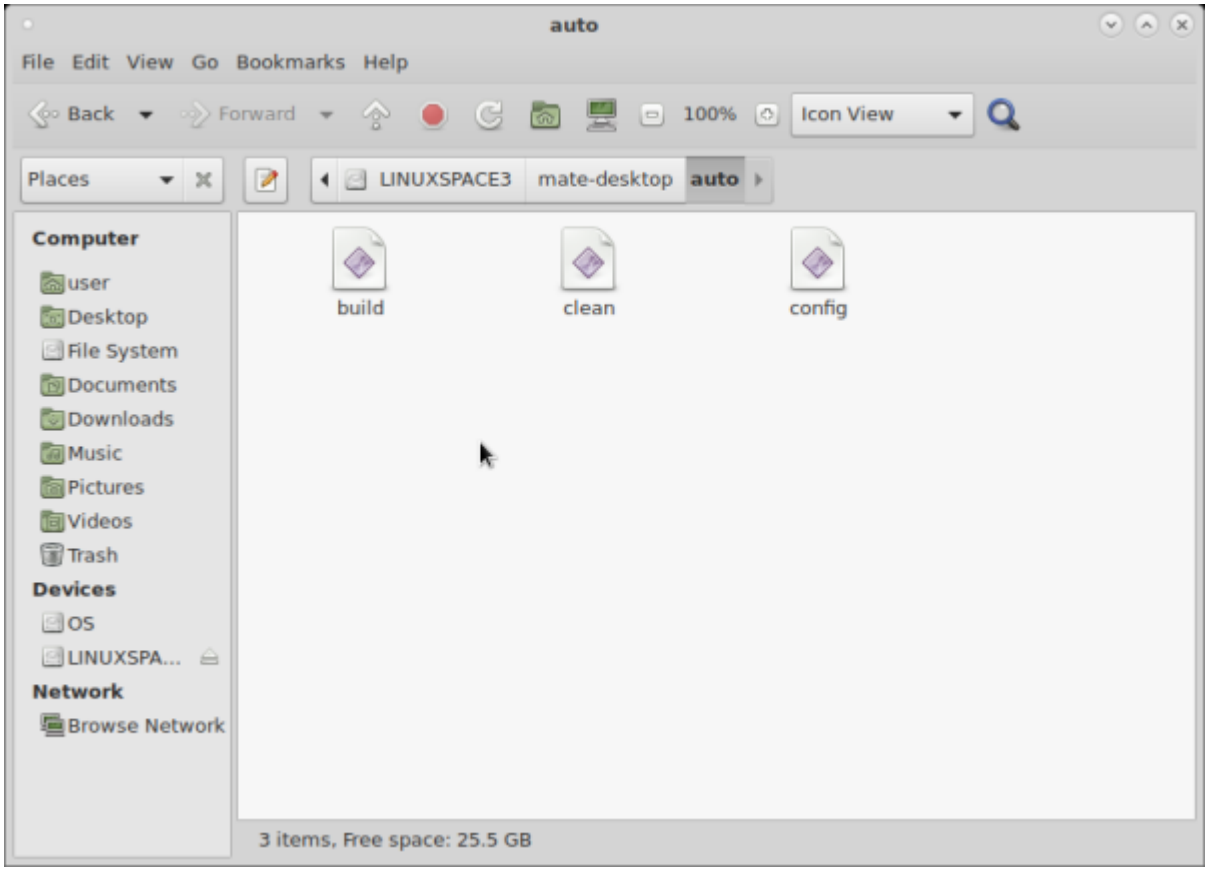
Drag-drop copy the mate-desktop folder from its installed location under /usr/share/live/images to the home folder or a suitable USB work drive.



If you look under the mate-desktop folder you should see...

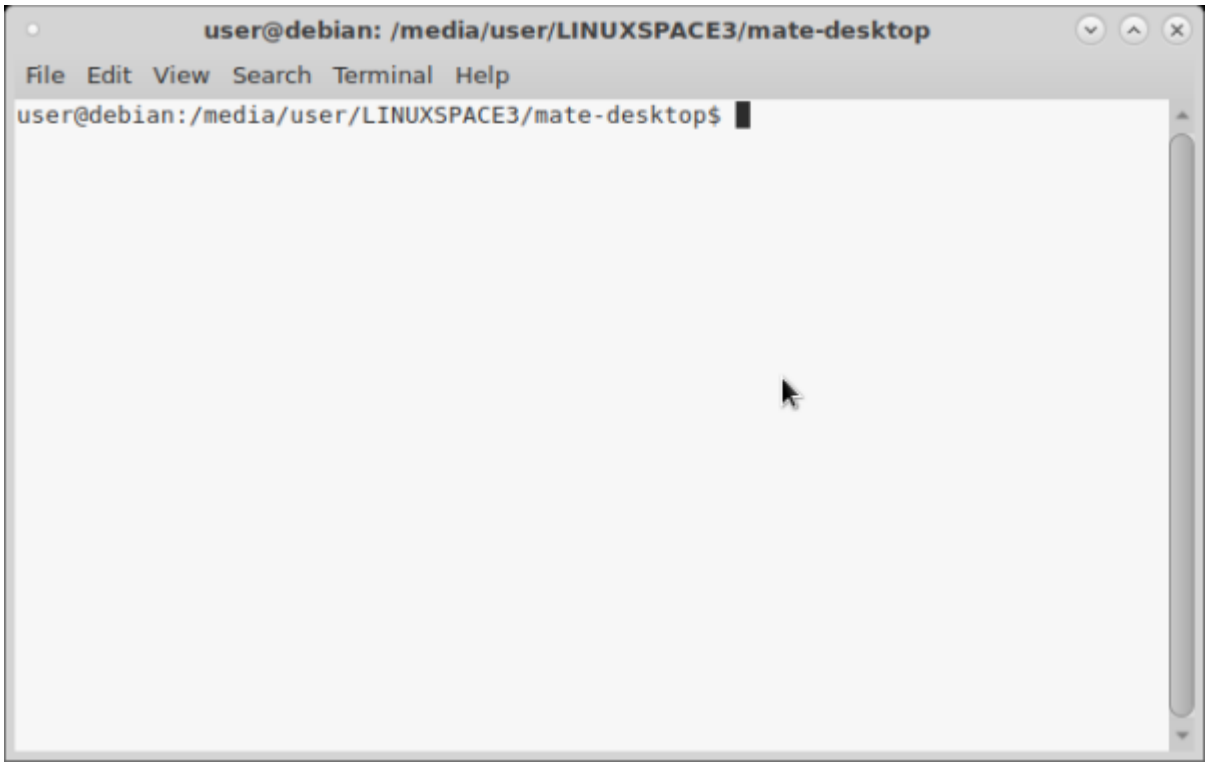


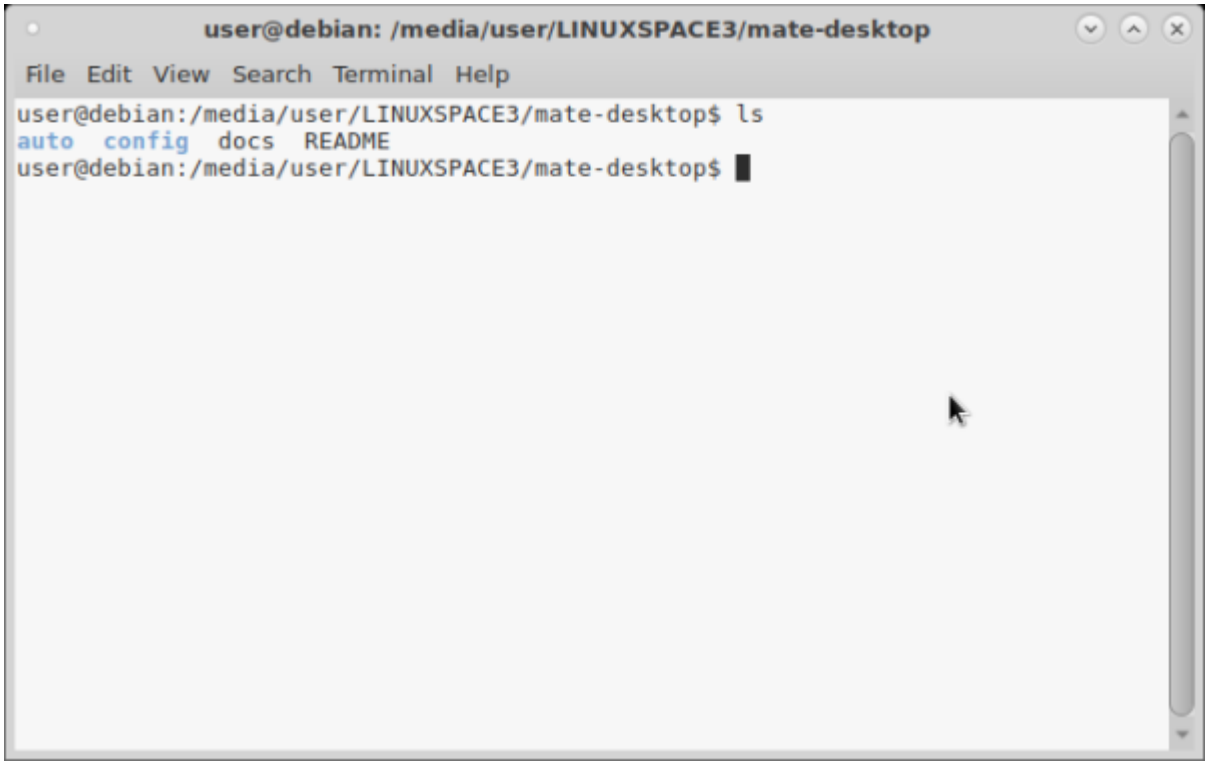
...and further under the "auto" subfolder...



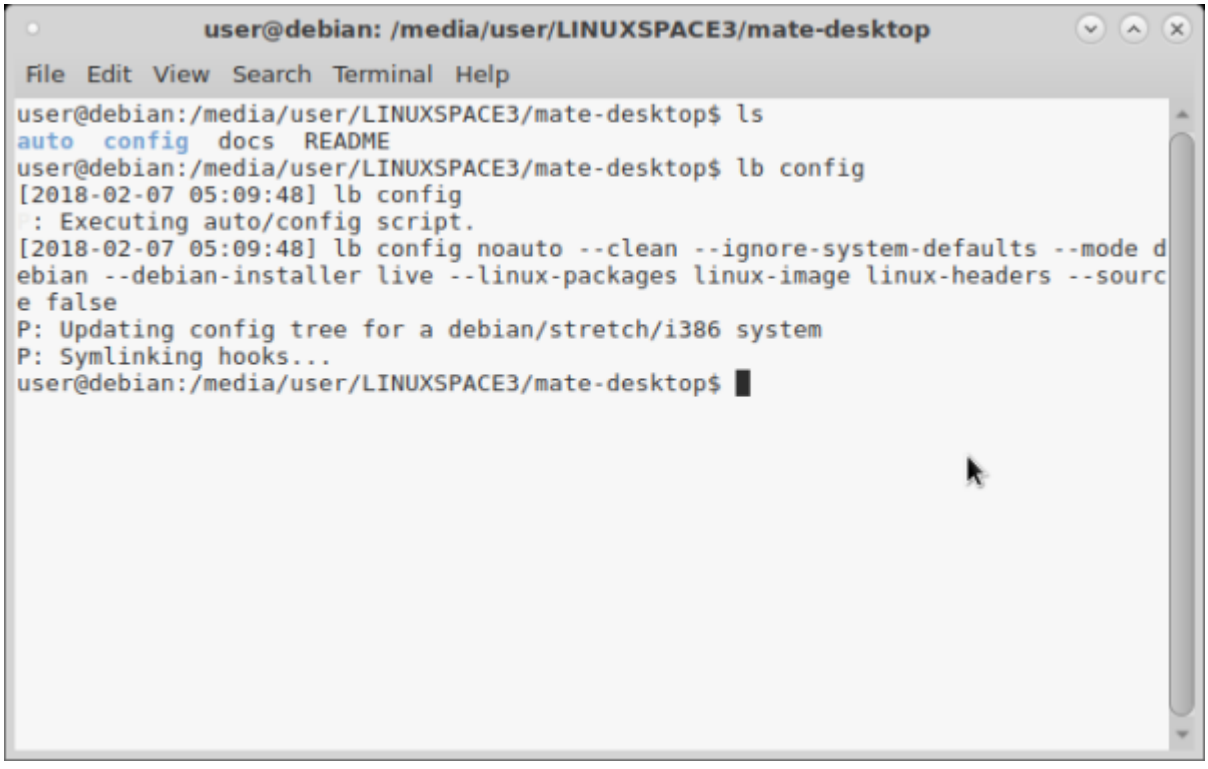
(go back to narrative)

6. Bring up a terminal into the "mate-desktop" folder, and build a basic configuration.

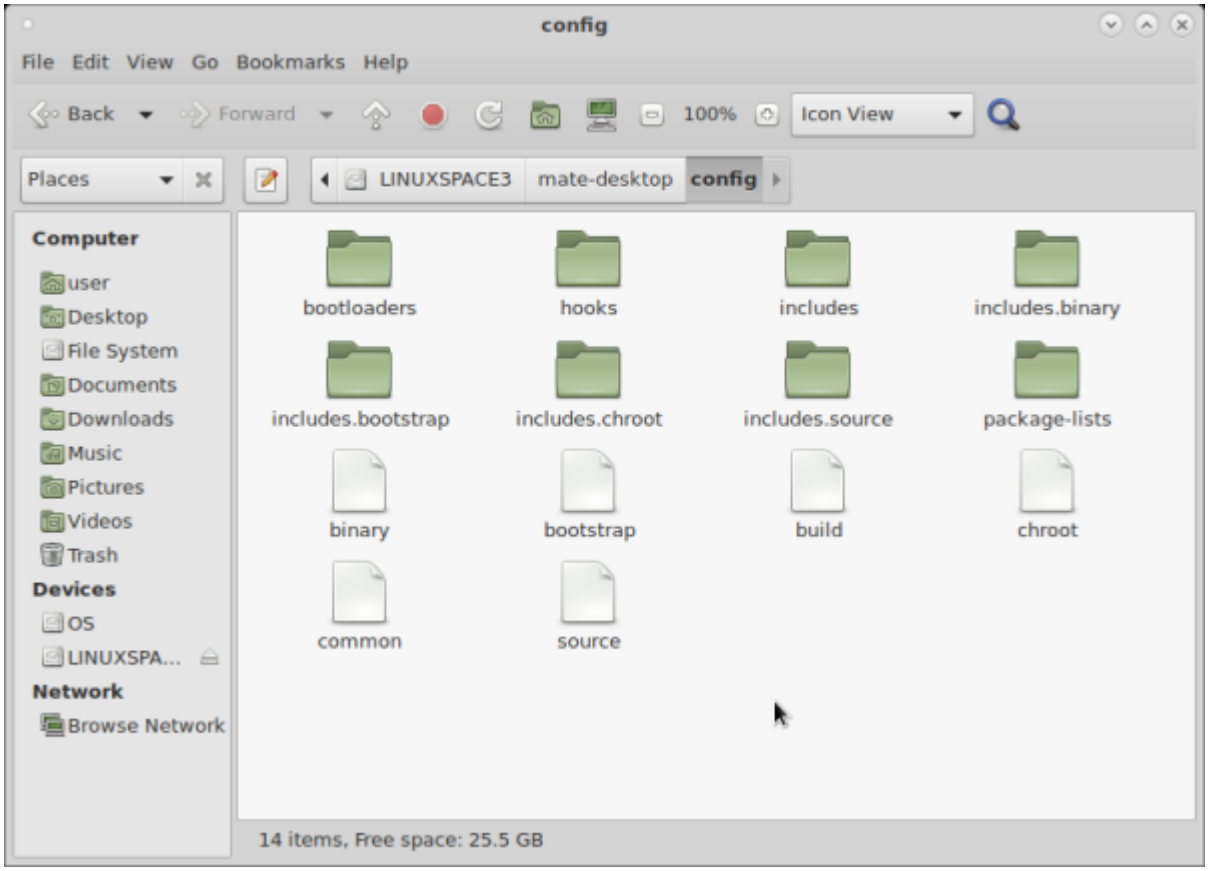




Execute "lb config". It will populate the initially bare configuration set with some basic confuration files matching the requirements (in 2015 anyway) of the Debian Mate official build.



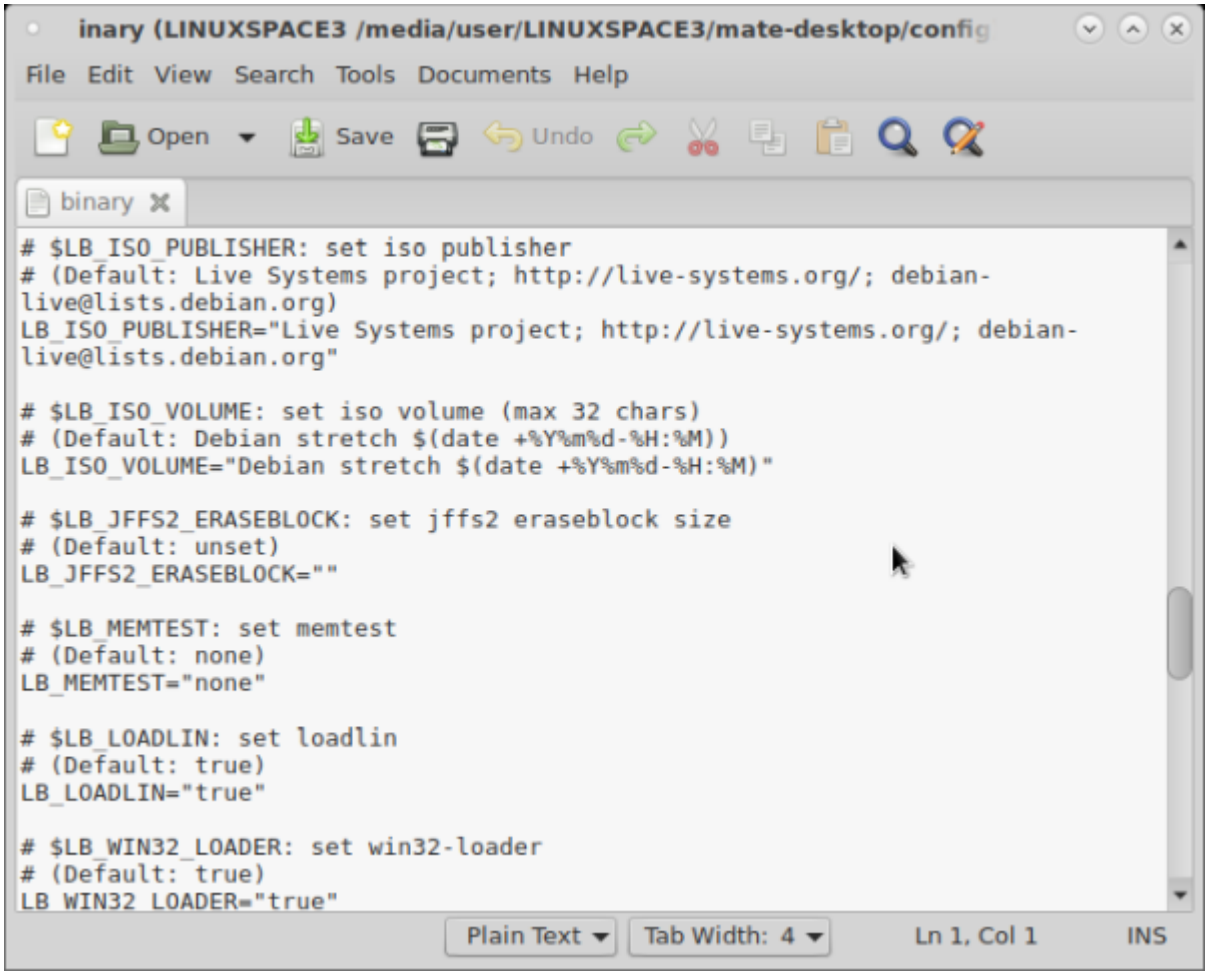
Looking insde "config" you will now see some new files.



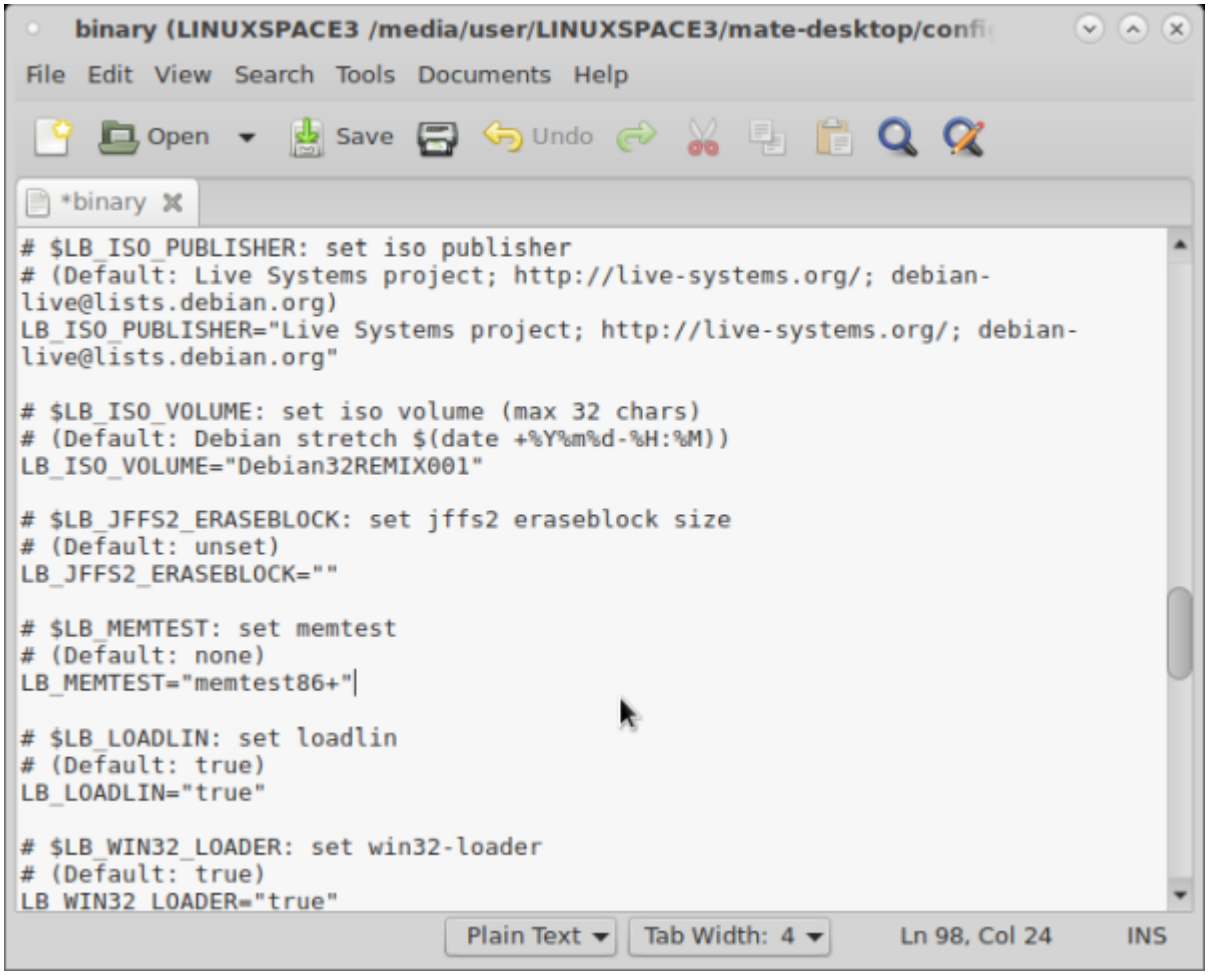
(go back to narrative)

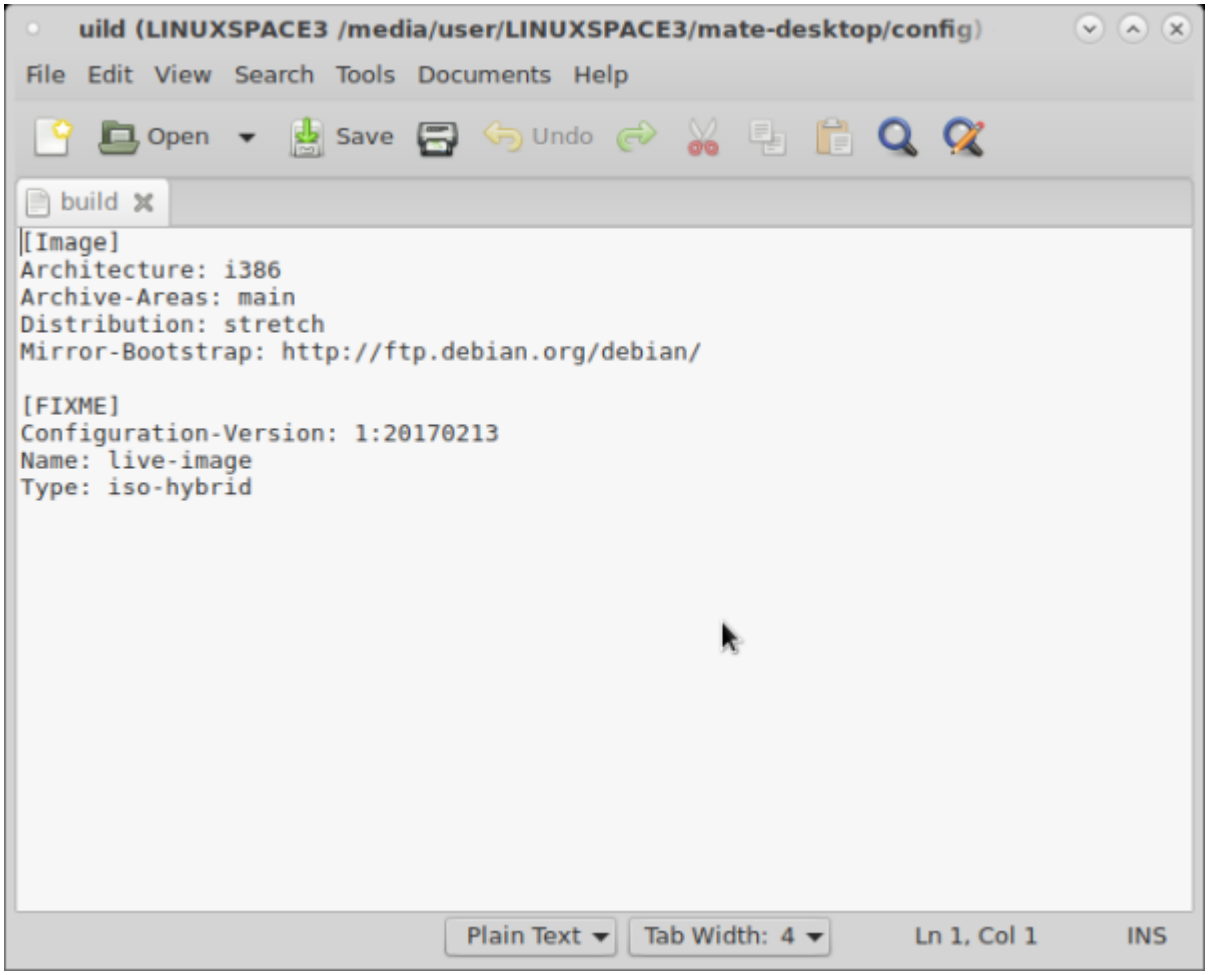
7. Manually edit the configuration to make it work.

Use the text editor "Pluma" to edit certain new confg files.

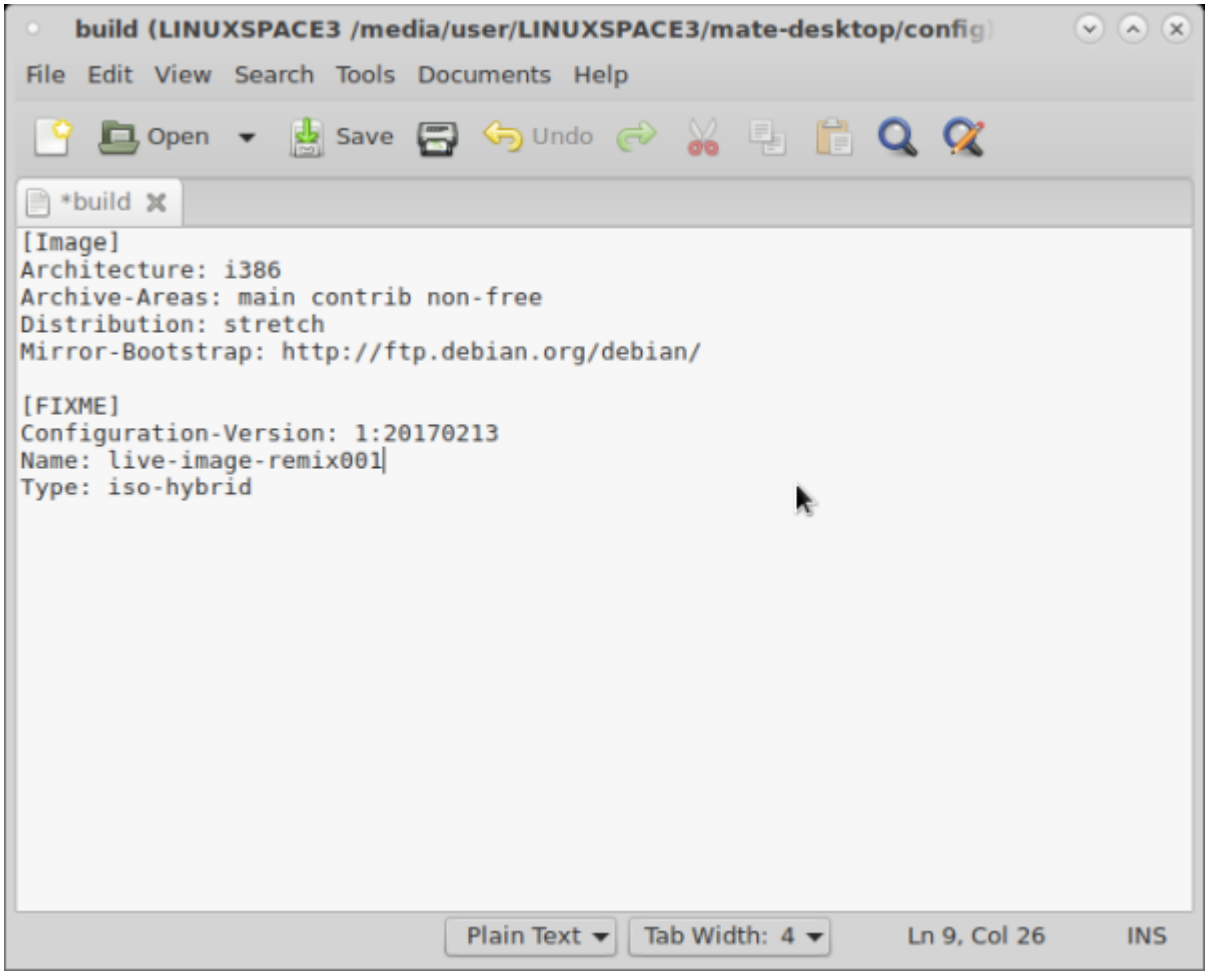


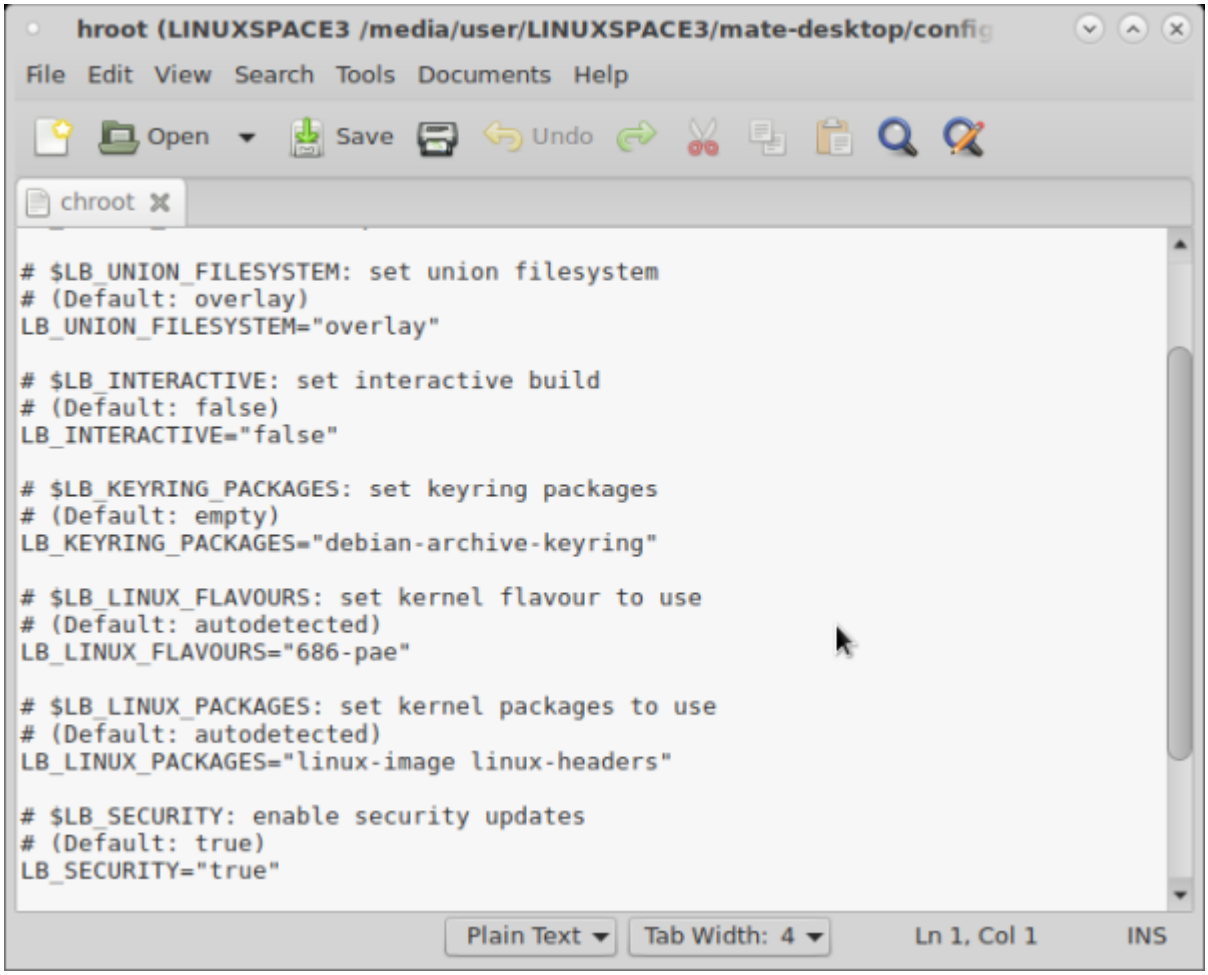
In the "binary" file, edit the ISO volume name of the remix and add a memory test to the boot menu.


binary (LINUXSPACE3 /media/user/LINUXSPACE3/mate-desktop/confi
File Edit View Search Tools Documents Help
Open
Save
Undo
*binary
# $LB_ISO_PUBLISHER: set iso publisher
# (Default: Live Systems project; http://live-systems.org/; debian-live@lists.debian.org)
LB_ISO_PUBLISHER="Live Systems project; http://live-systems.org/; debian-live@lists.debian.org"
# $LB_ISO_VOLUME: set iso volume (max 32 chars)
# (Default: Debian stretch $(date +%Y%m%d-%H:%M))
LB_ISO_VOLUME="Debian32REMIX001"
# $LB_JFFS2_ERASEBLOCK: set jffs2 eraseblock size
# (Default: unset)
LB_JFFS2_ERASEBLOCK=""
# $LB_MEMTEST: set memtest
# (Default: none)
LB_MEMTEST="memtest86+"
# $LB_LOADLIN: set loadlin
# (Default: true)
LB_LOADLIN="true"
# $LB_WIN32_LOADER: set win32-loader
# (Default: true)
LB_WIN32_LOADER="true"
Plain Text
Tab Width: 4
Ln 98, Col 24
INS

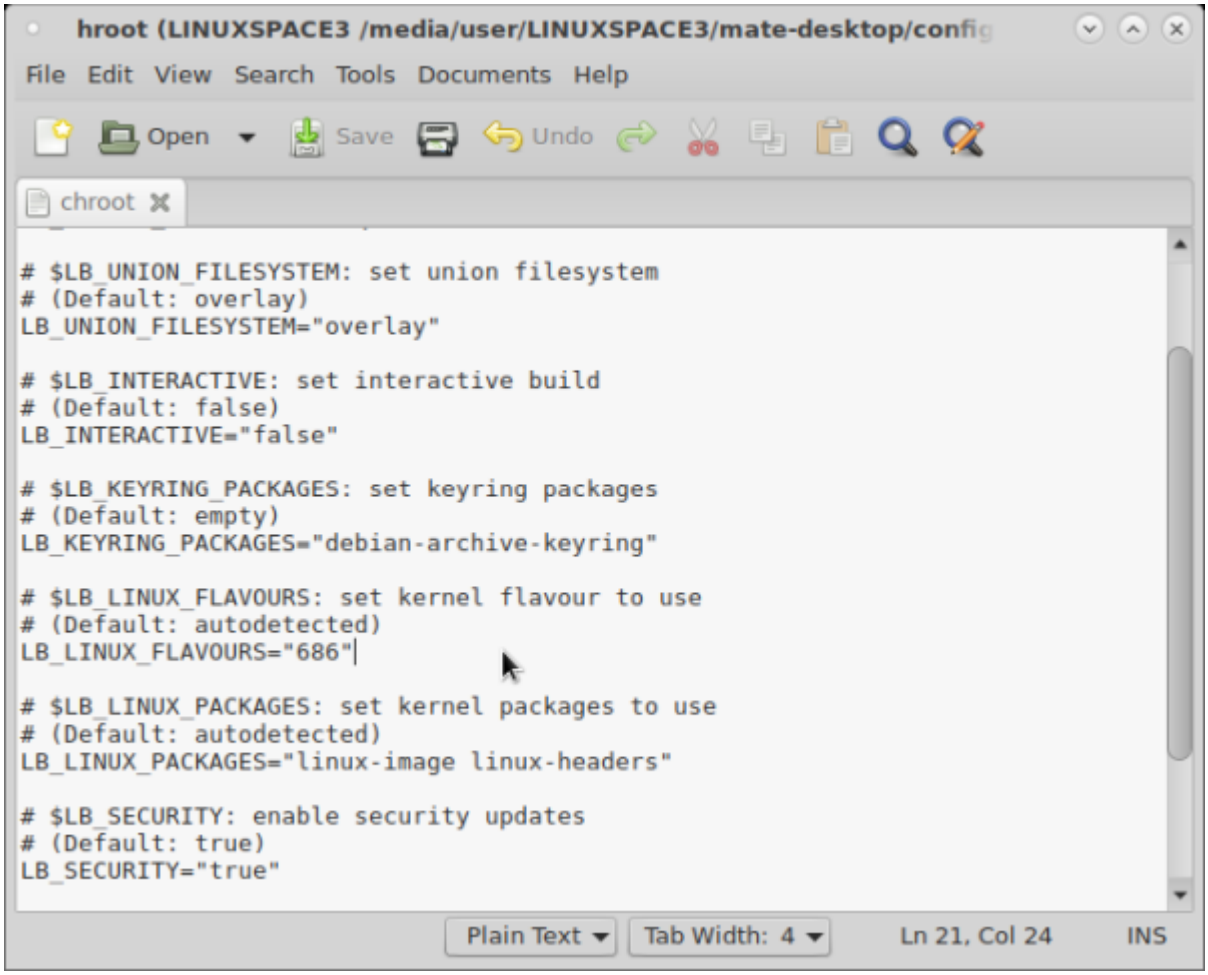


In the "build" file, edit the (partial) ISO file name of the remix and add additional repository types "contrib" and "non-free".

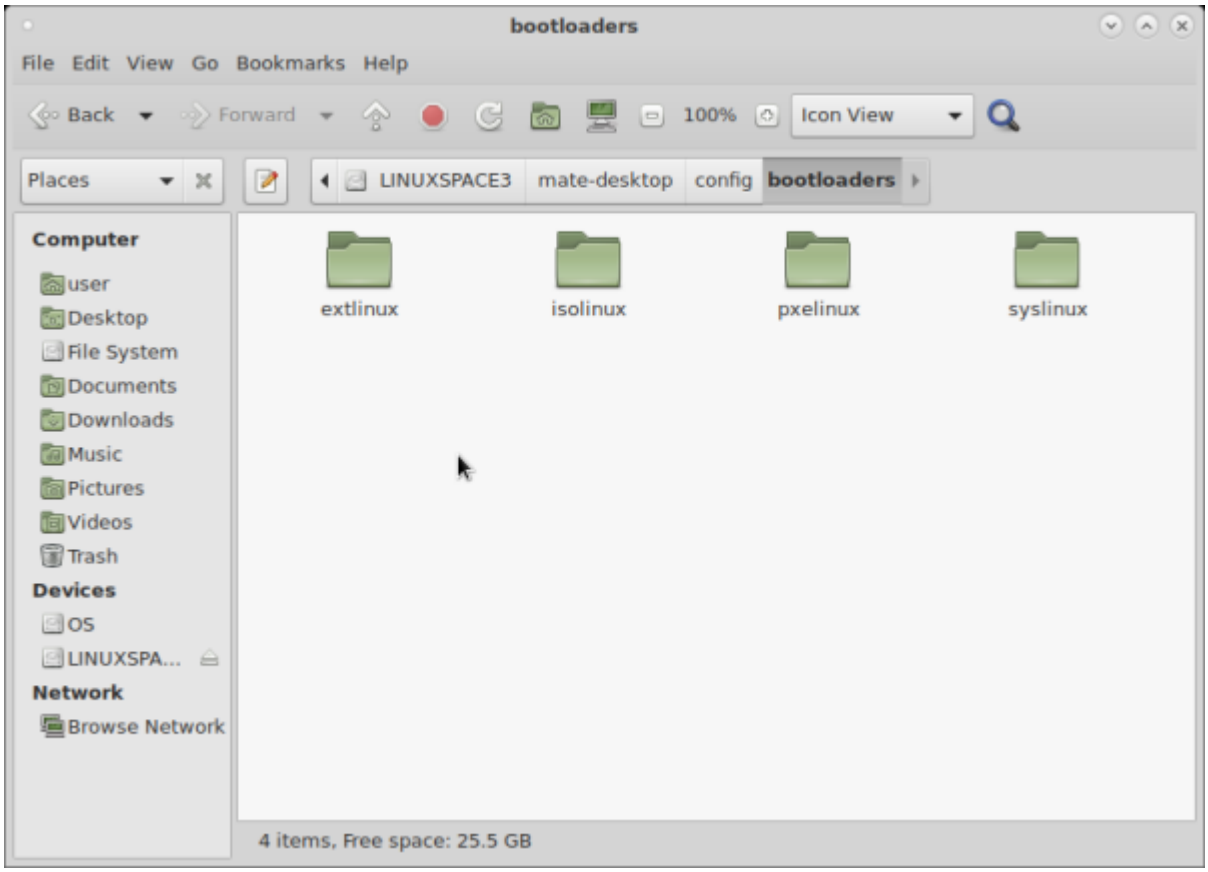

build (LINUXSPACE3 /media/user/LINUXSPACE3/mate-desktop/config)
File Edit View Search Tools Documents Help
Open
Save
Undo
*build
[Image]
Architecture: i386
Archive-Areas: main contrib non-free
Distribution: stretch
Mirror-Bootstrap: http://ftp.debian.org/debian/
[FIXME]
Configuration-Version: 1:20170213
Name: live-image-remix001
Type: iso-hybrid
Plain Text
Tab Width: 4
Ln 9, Col 26
INS



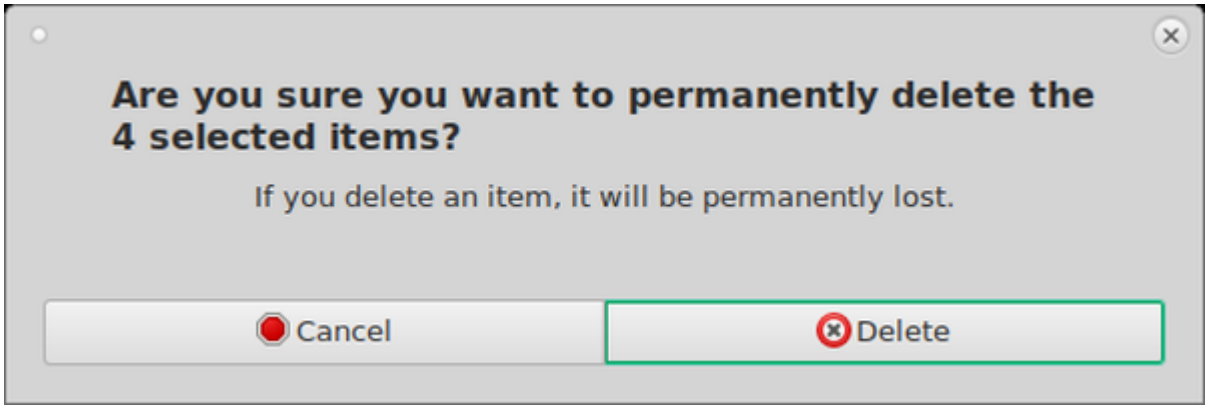
In the "chroot" file, if the specification for LB_LINUX_FLAVOURS is "686-pae", edit that to "686". that change will allow the built image to run on more (older) systems.

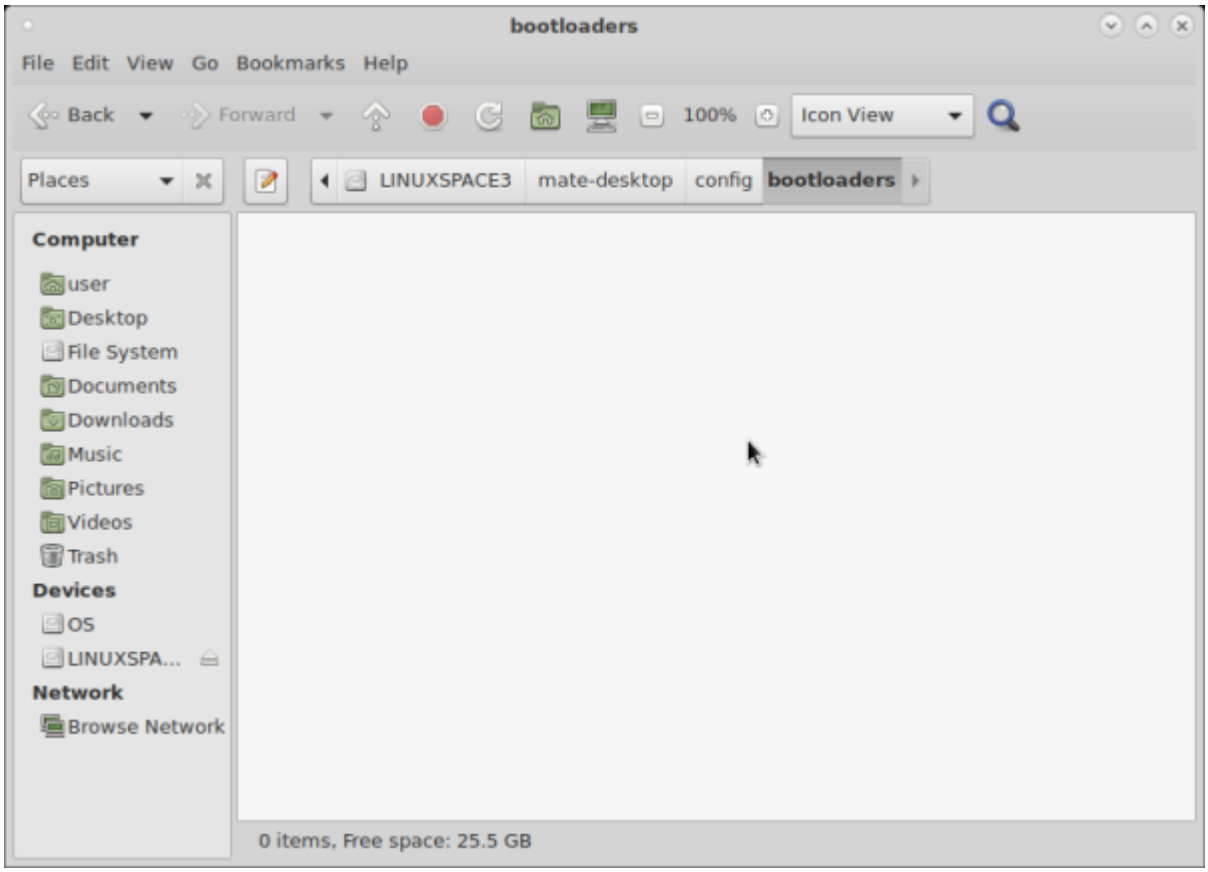


Under config/bootloaders, you should see...

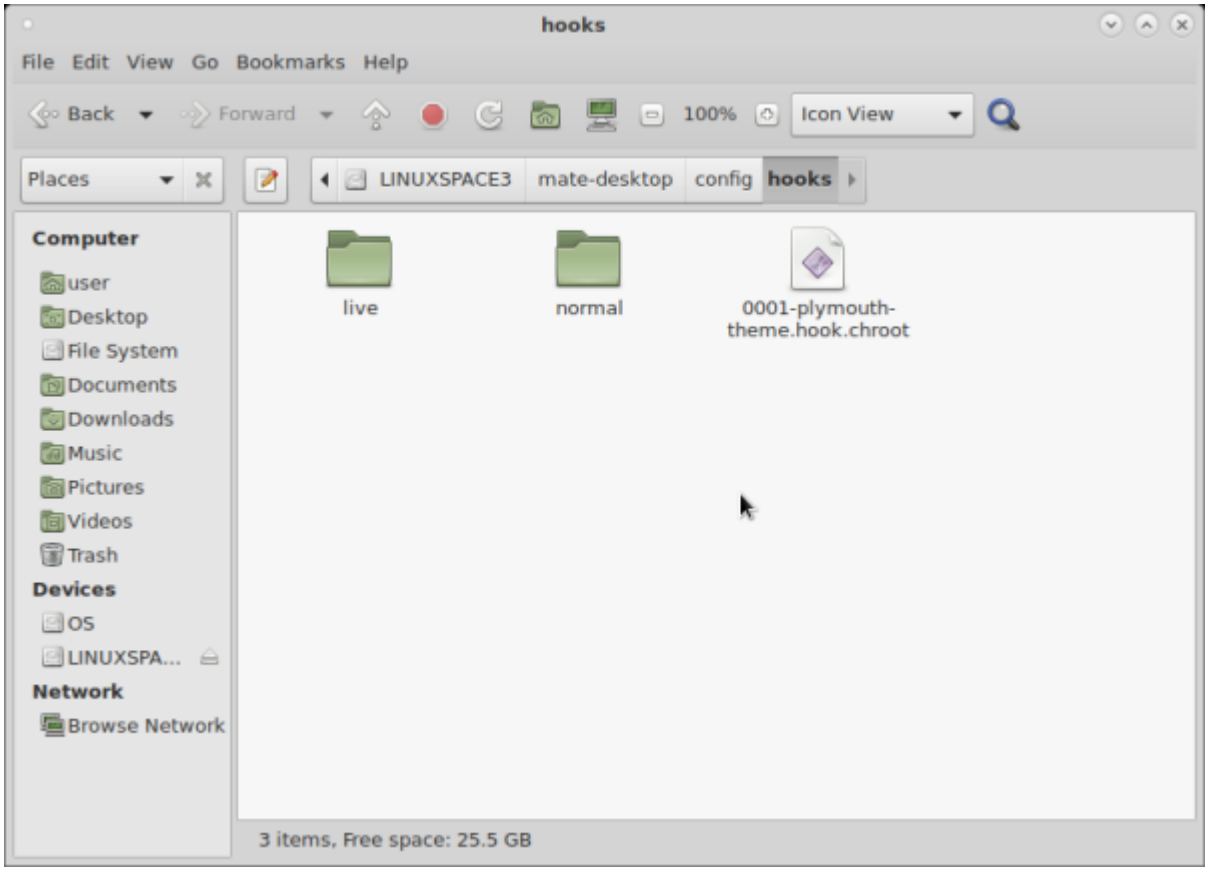


Delete all those folders -- they cause the build to fail if left intact (the original 2015 specs are wrong for the current build system).



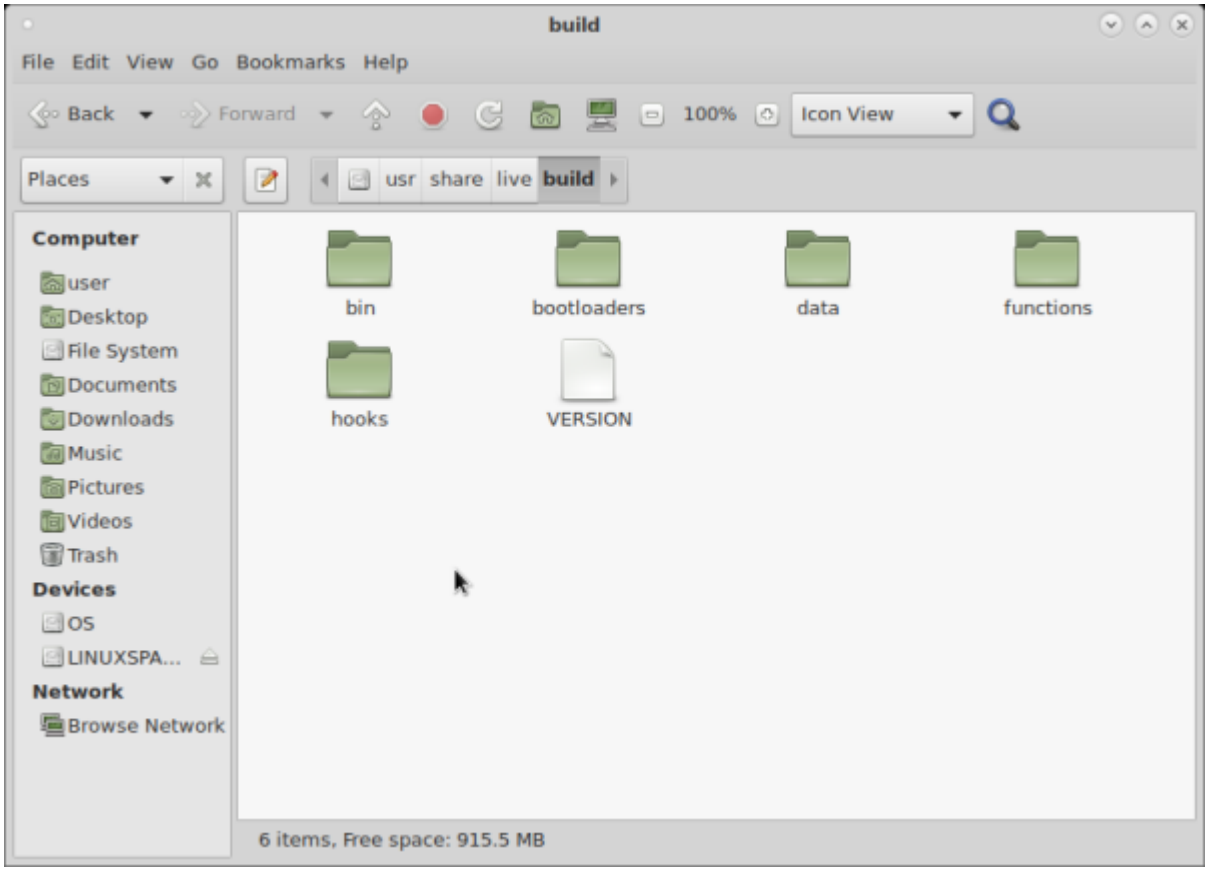


Under config/hooks, you will find links to hook script files installed into the filesystem with live-build.

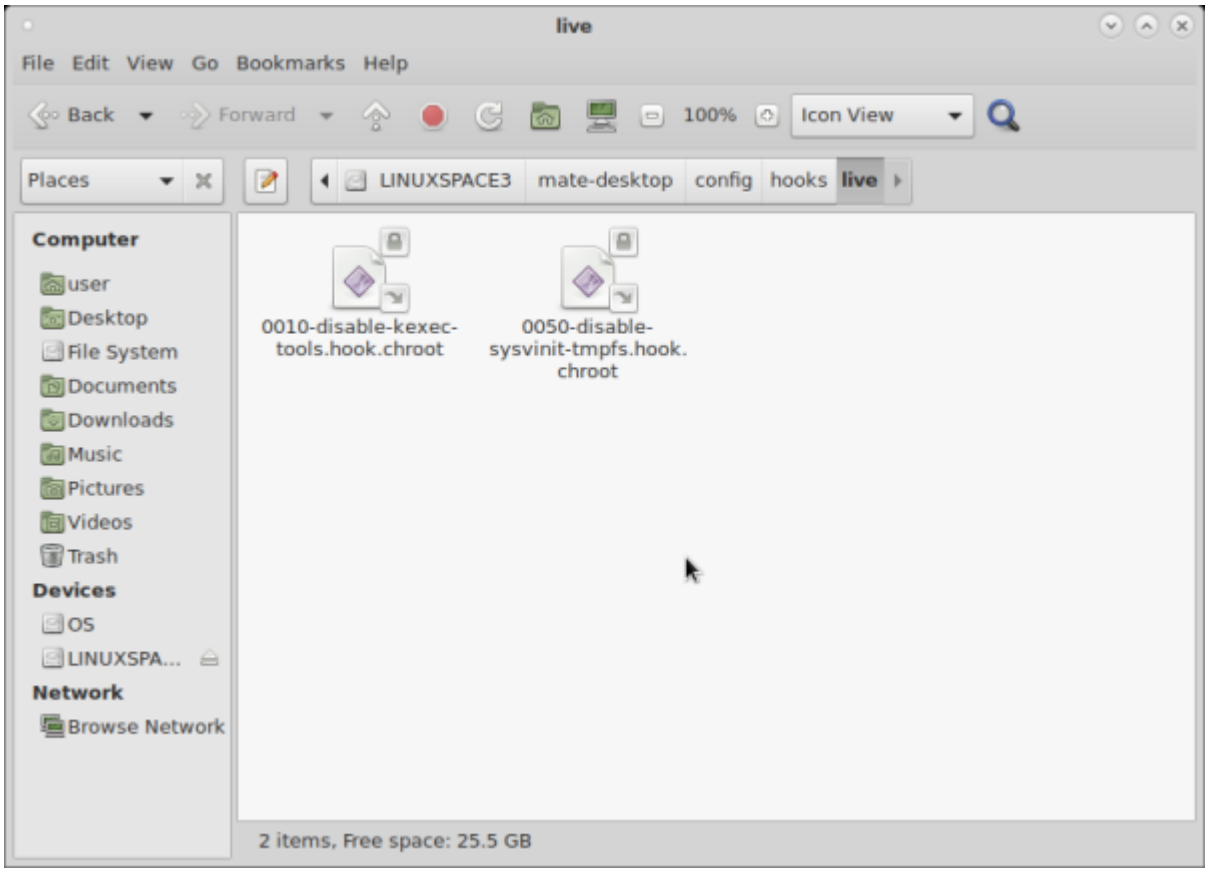


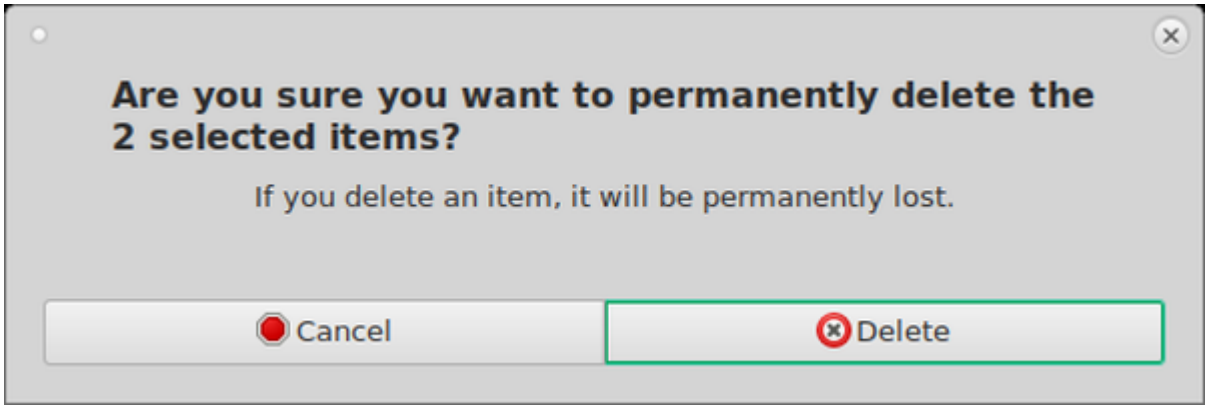
You can often get away without these hooks, so you can delete the links altogether. However, since you don't know everything they do, you may wish to keep them. if you do, the links can cause trouble later during the "chroot" phase of the build (the links are coded relative, and the chroot can't "see" higher than its own top level folder, so the search for each link comes up empty). It is better to replace the links with actual copies of the original files, found under /usr/share/live/build/hooks. Delete all the links first, then copy the "normal" folder contents and the "live" folder contents to the configuration set.

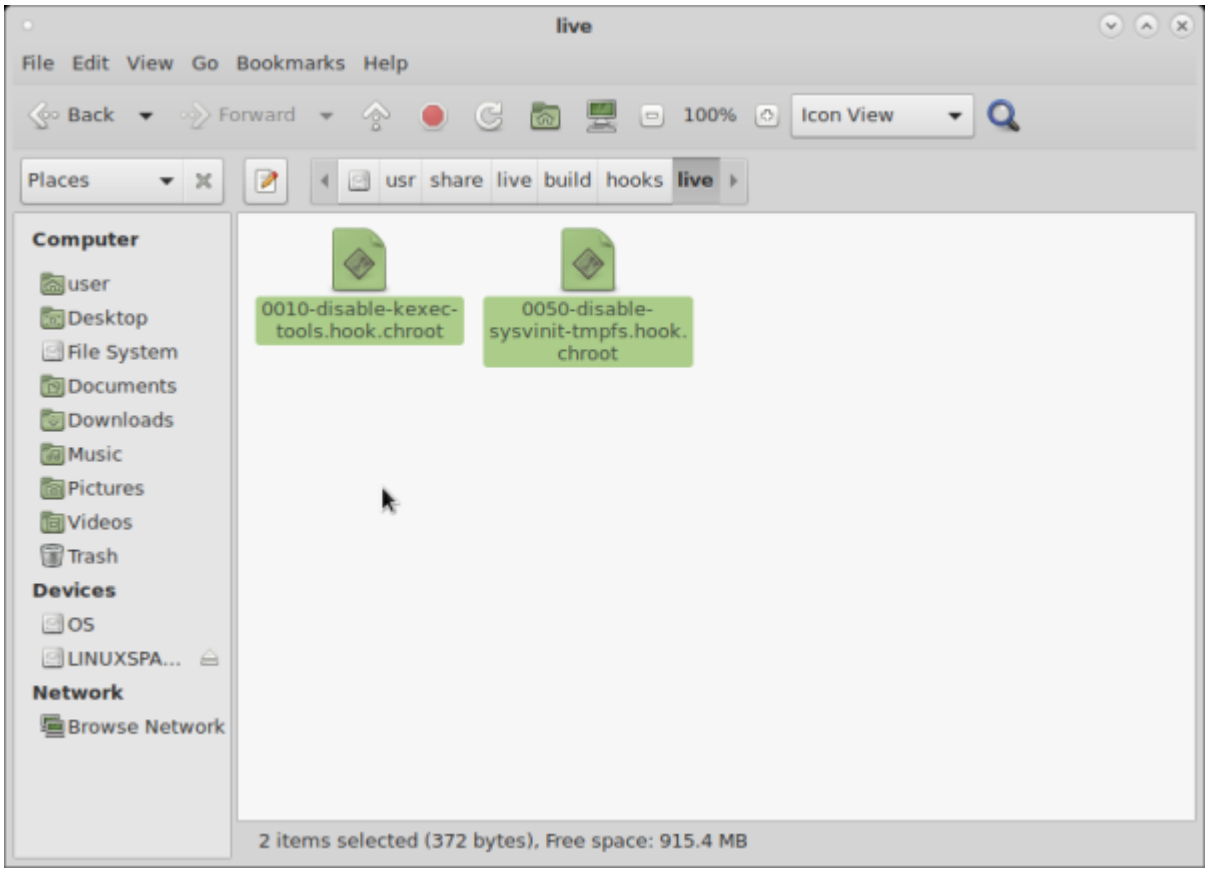
Locate the replacement hook files to which the links point.

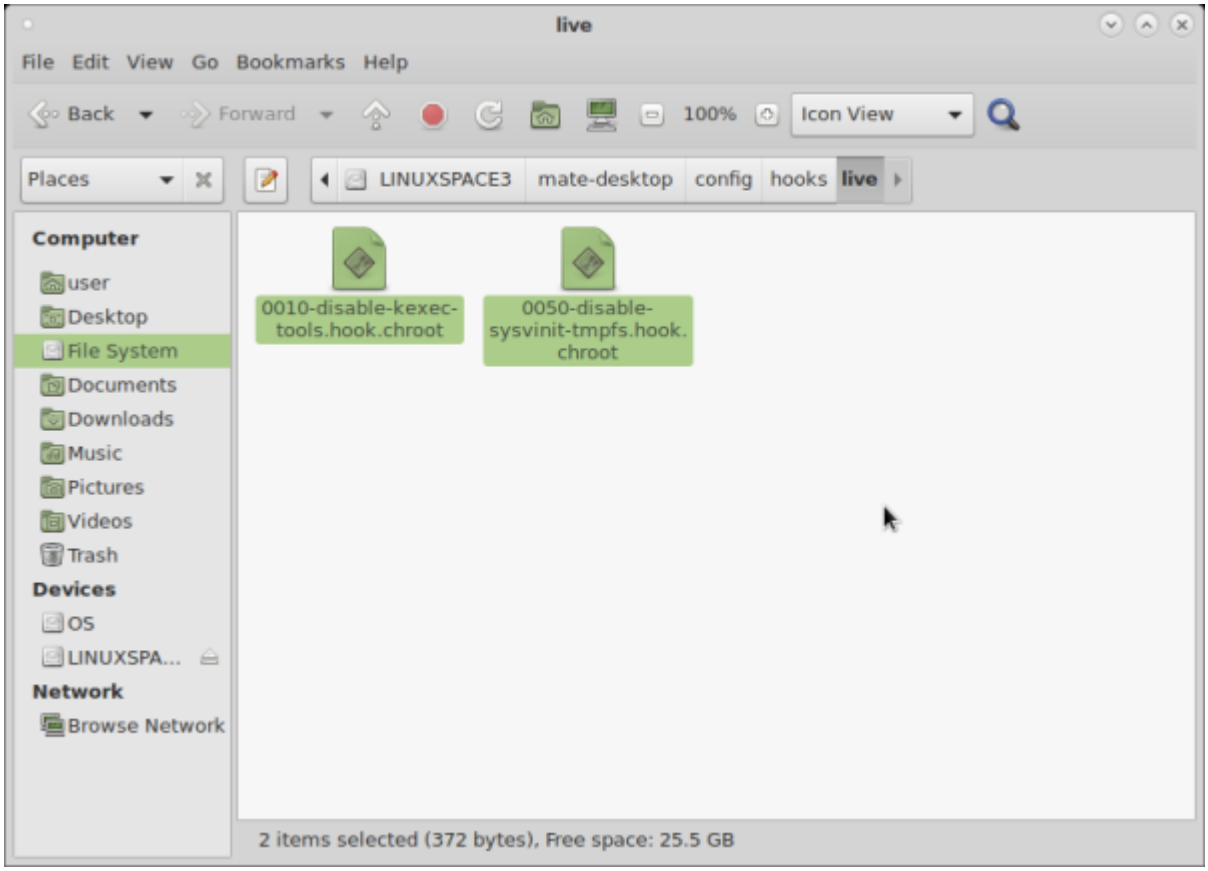


Delete the links under "live", then copy the replacements...

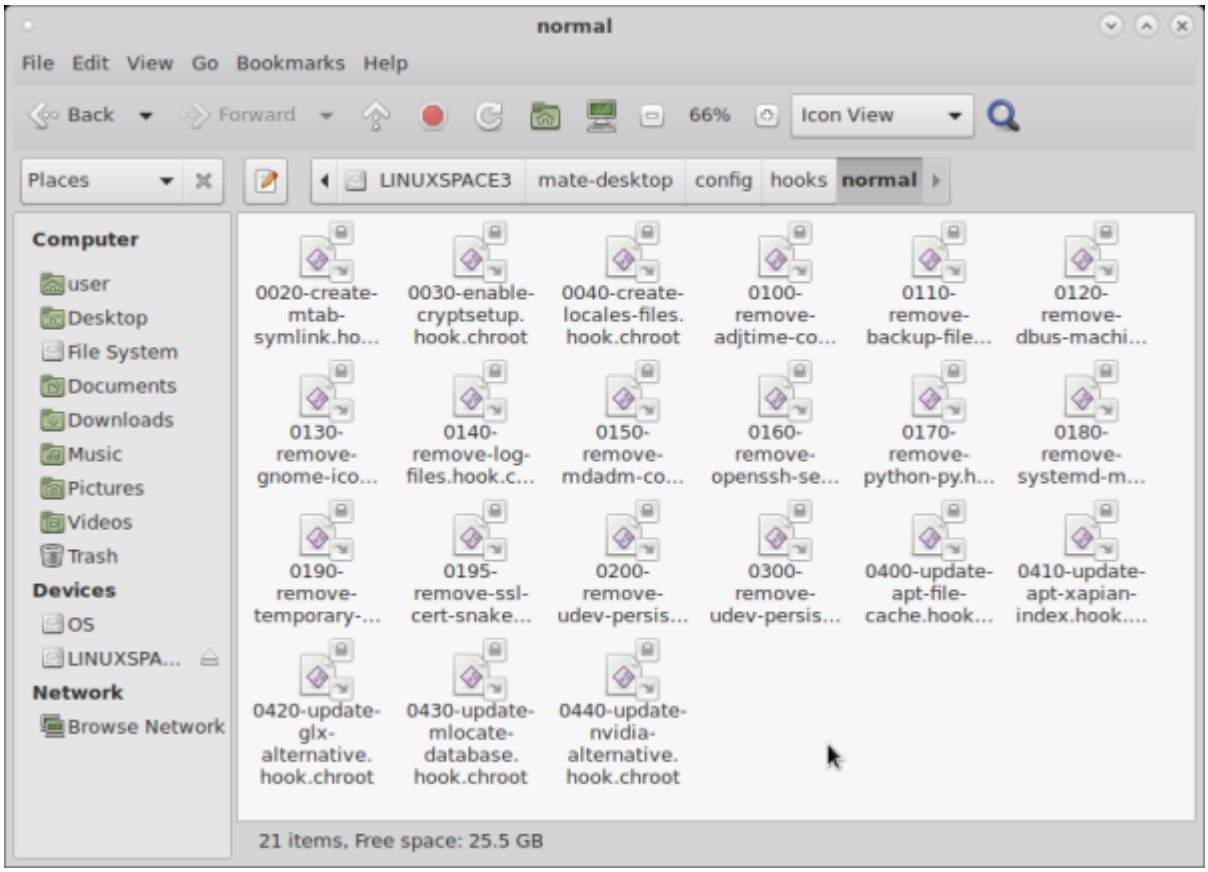


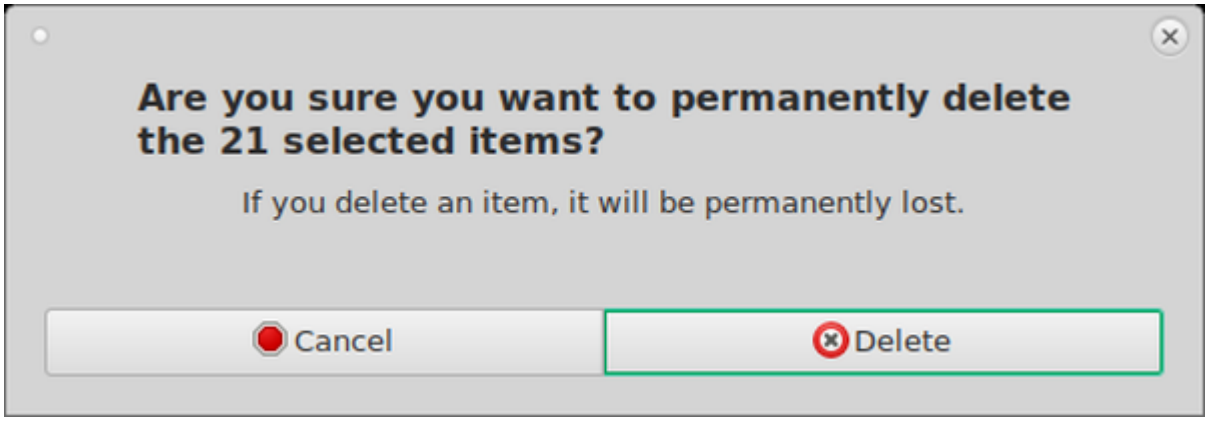

Are you sure you want to permanently delete the 2 selected items?
If you delete an item, it will be permanently lost.
Cancel
Delete

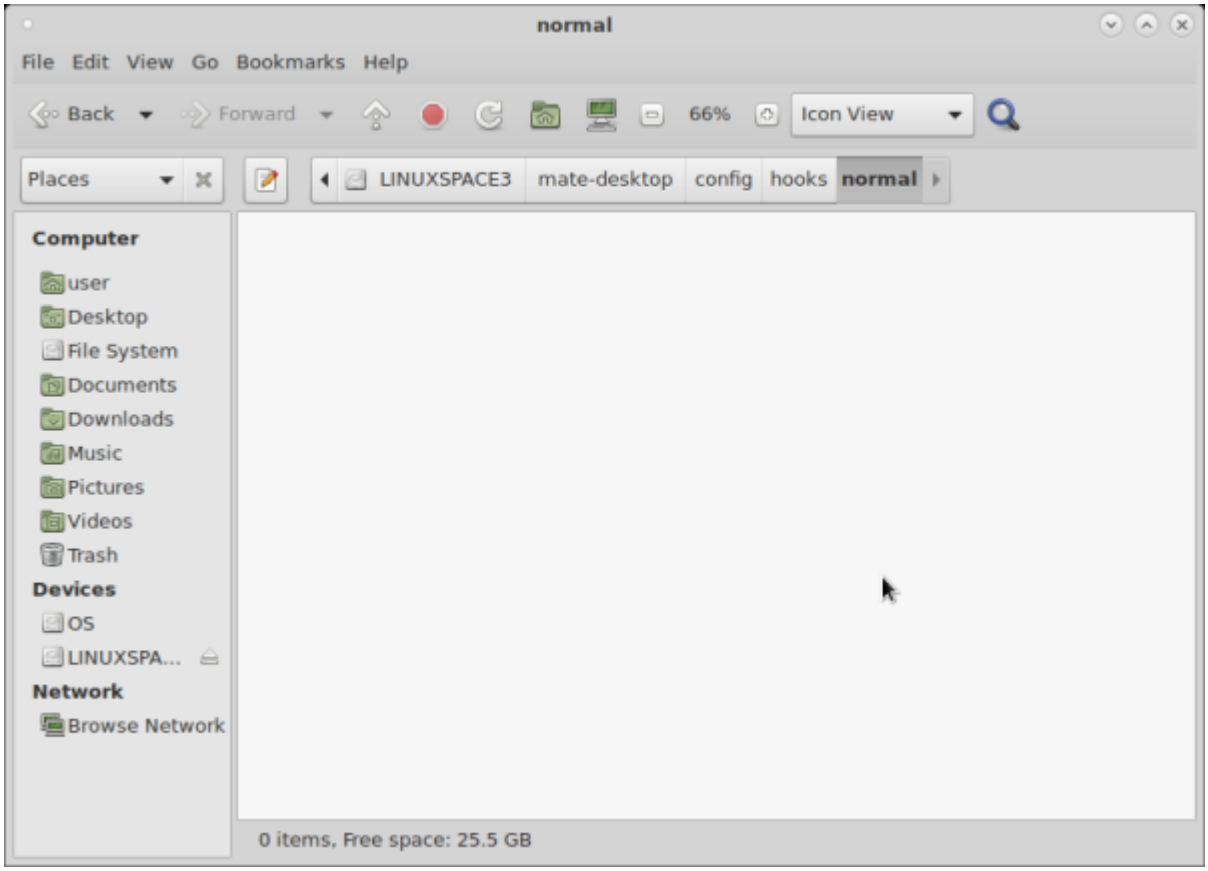

live
File Edit View Go Bookmarks Help
Back Forward 100% Icon View
Places
usr share live build hooks live
Computer
user
Desktop
File System
Documents
Downloads
Music
Pictures
Videos
Trash
Devices
OS
LINUXSPA...
Network
Browse Network
0010-disable-kexec-tools.hook.chroot
0050-disable-sysvinit-tmpfs.hook.chroot
2 items selected (372 bytes), Free space: 915.4 MB

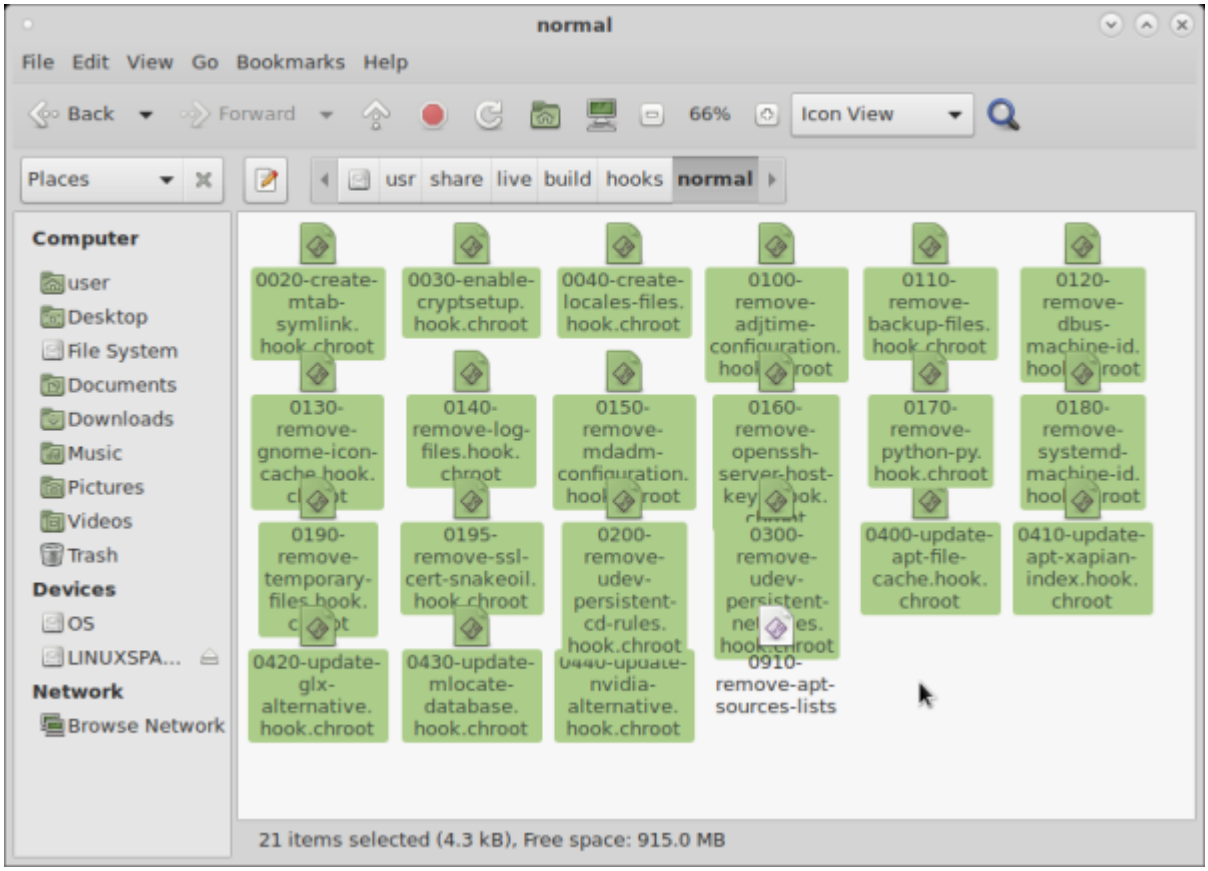


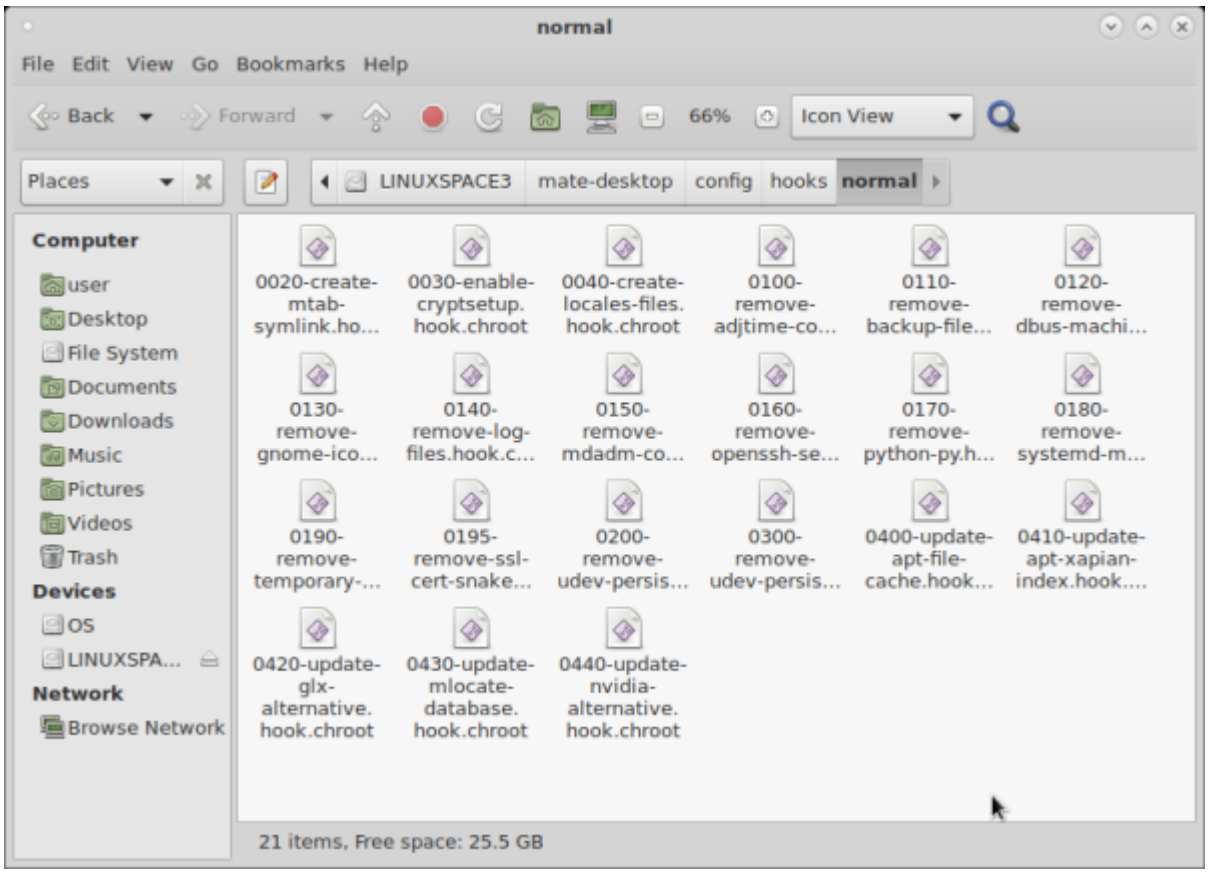
Do the same for links under "normal", then copy their replacements...



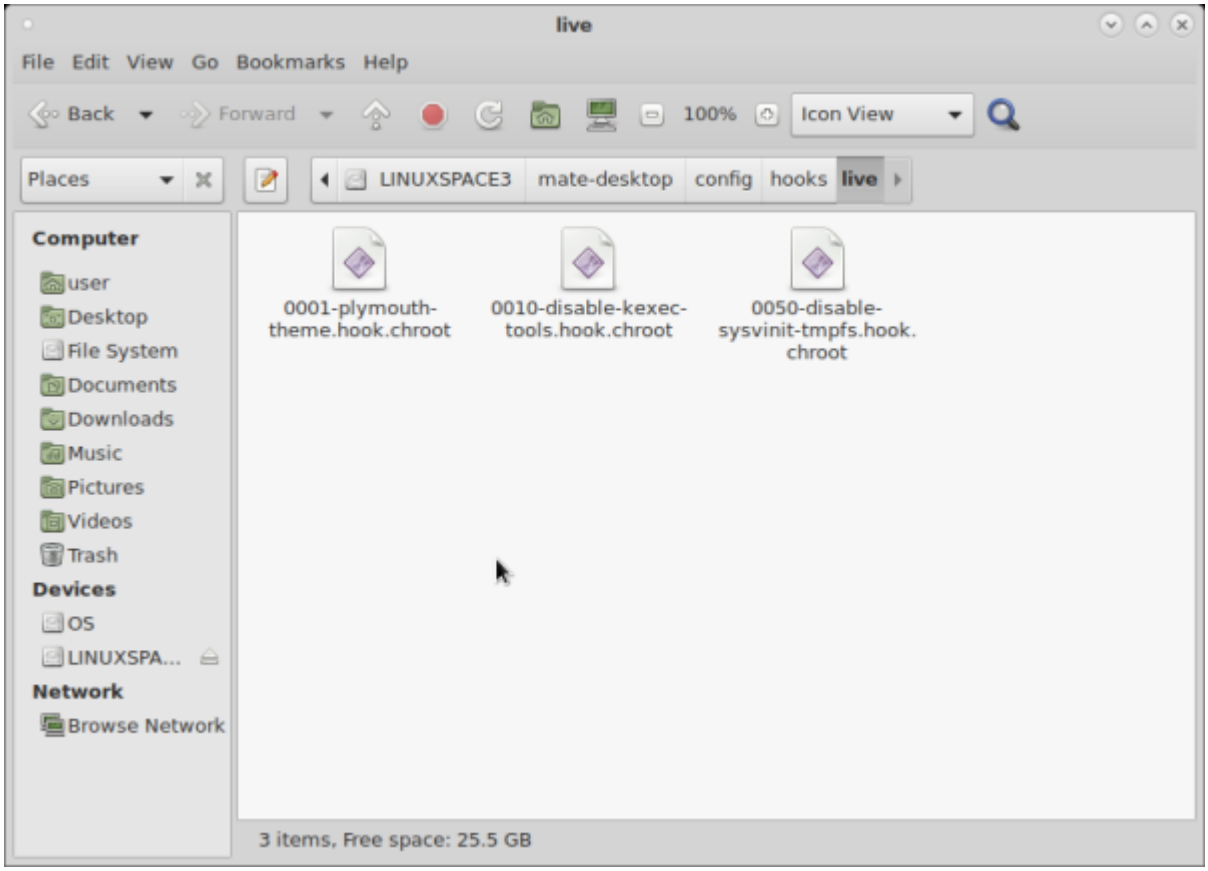

Are you sure you want to permanently delete the 21 selected items?
If you delete an item, it will be permanently lost.
Cancel
Delete


normal
File Edit View Go Bookmarks Help
Back Forward 66% Icon View
Places
LINUXSPACE3 mate-desktop config hooks normal
Computer
user
Desktop
File System
Documents
Downloads
Music
Pictures
Videos
Trash
Devices
OS
LINUXSPA...
Network
Browse Network
0 items, Free space: 25.5 GB

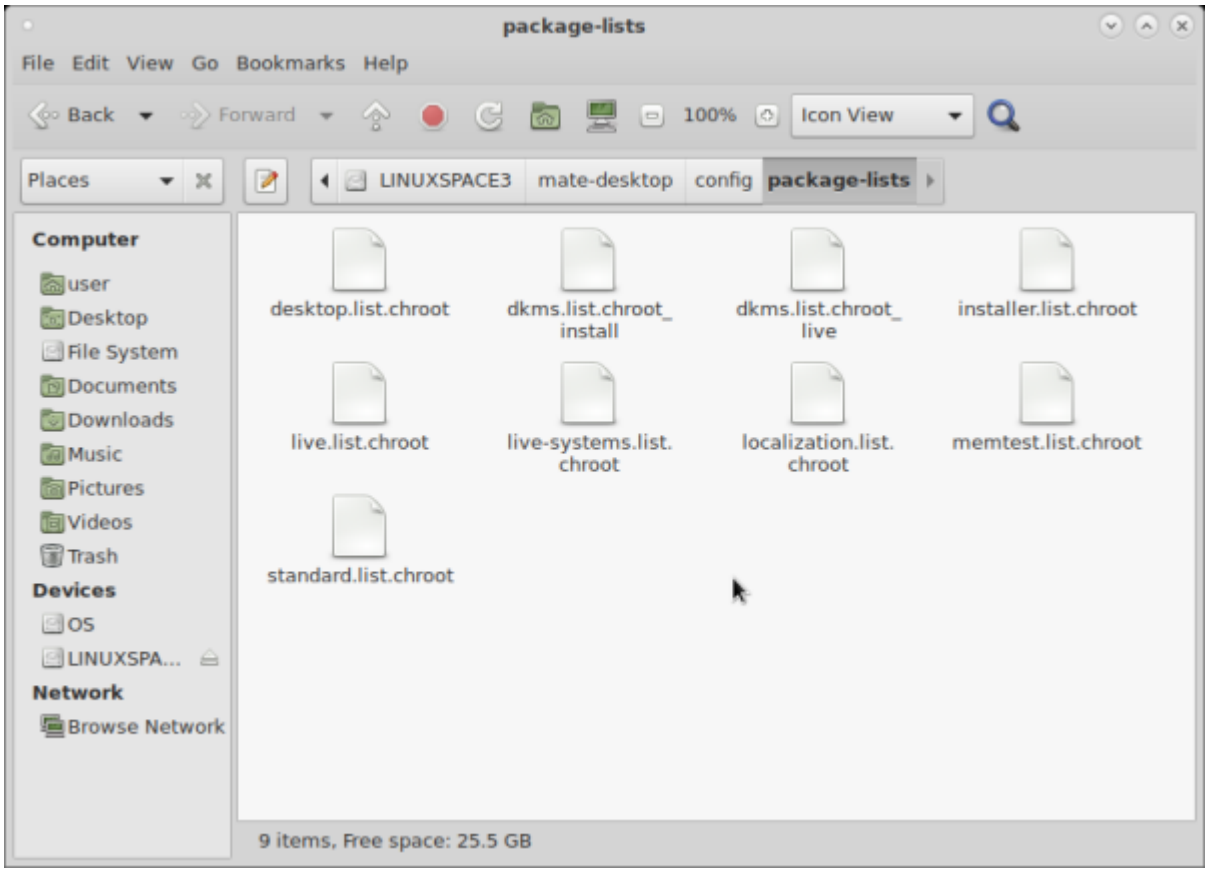




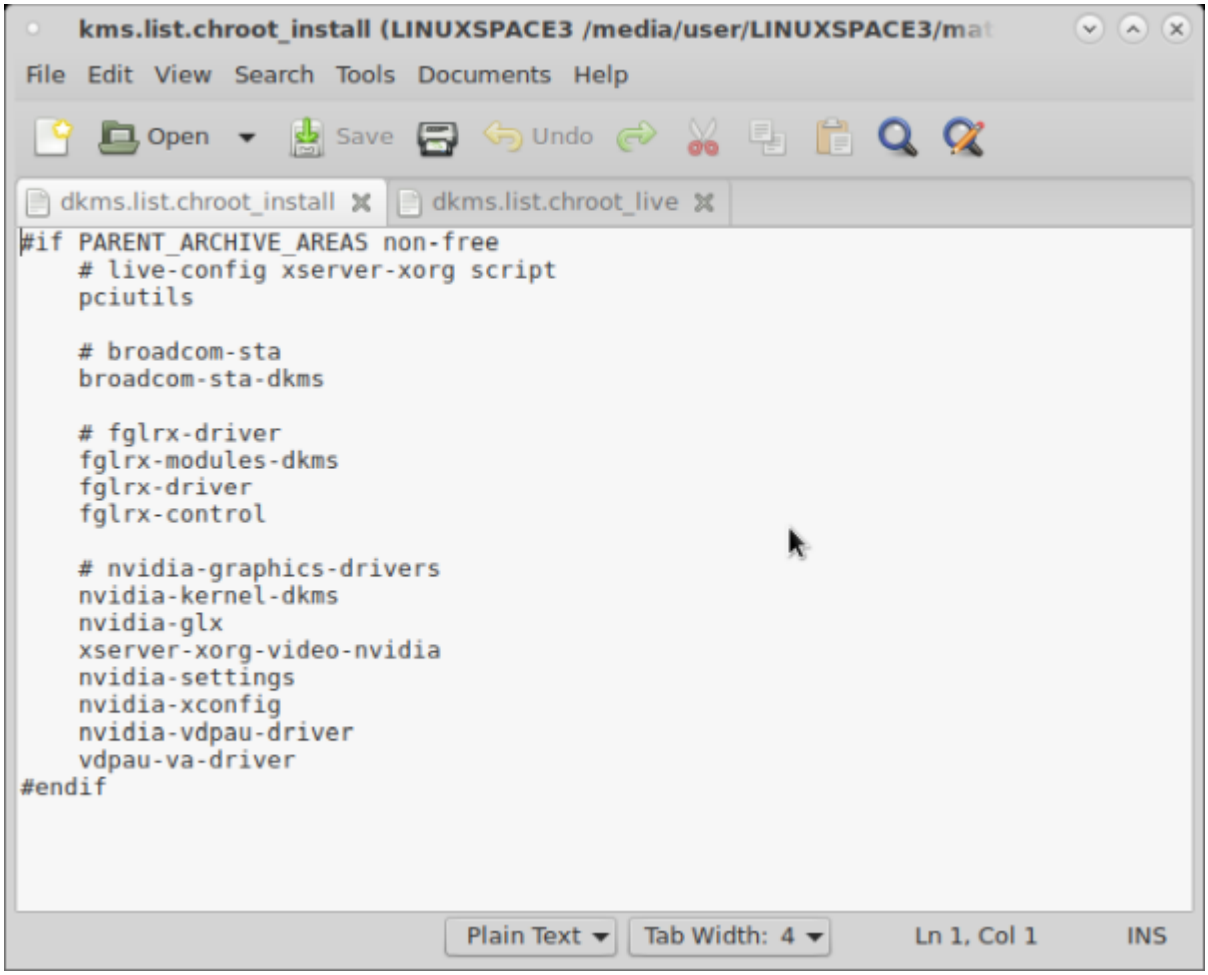
Now move the lone file above "live' and "normal" into "live". The prior version of the live build configuration did not have "live" and "normal", so if that lone configuration is important, it needs to go under one or the other of the subfolders.

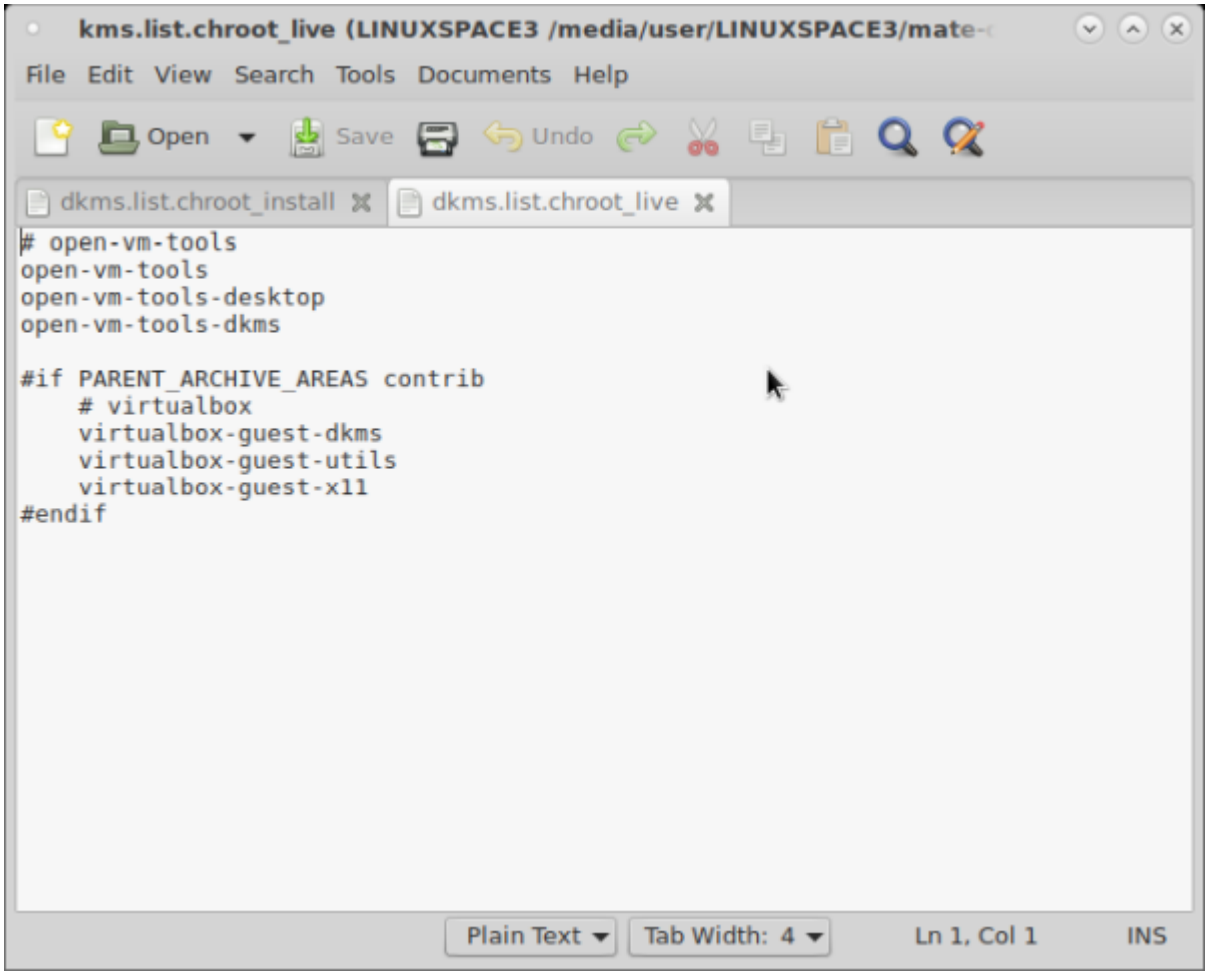


Now examine config/package-lists. These are lists of package names to be installed from the repositories. However, it seems the important ones are in "desktop.list.chroot" or "live.list.chroot". In fact, if you leave the ones starting woth "dkms." intact, they will kill the build.

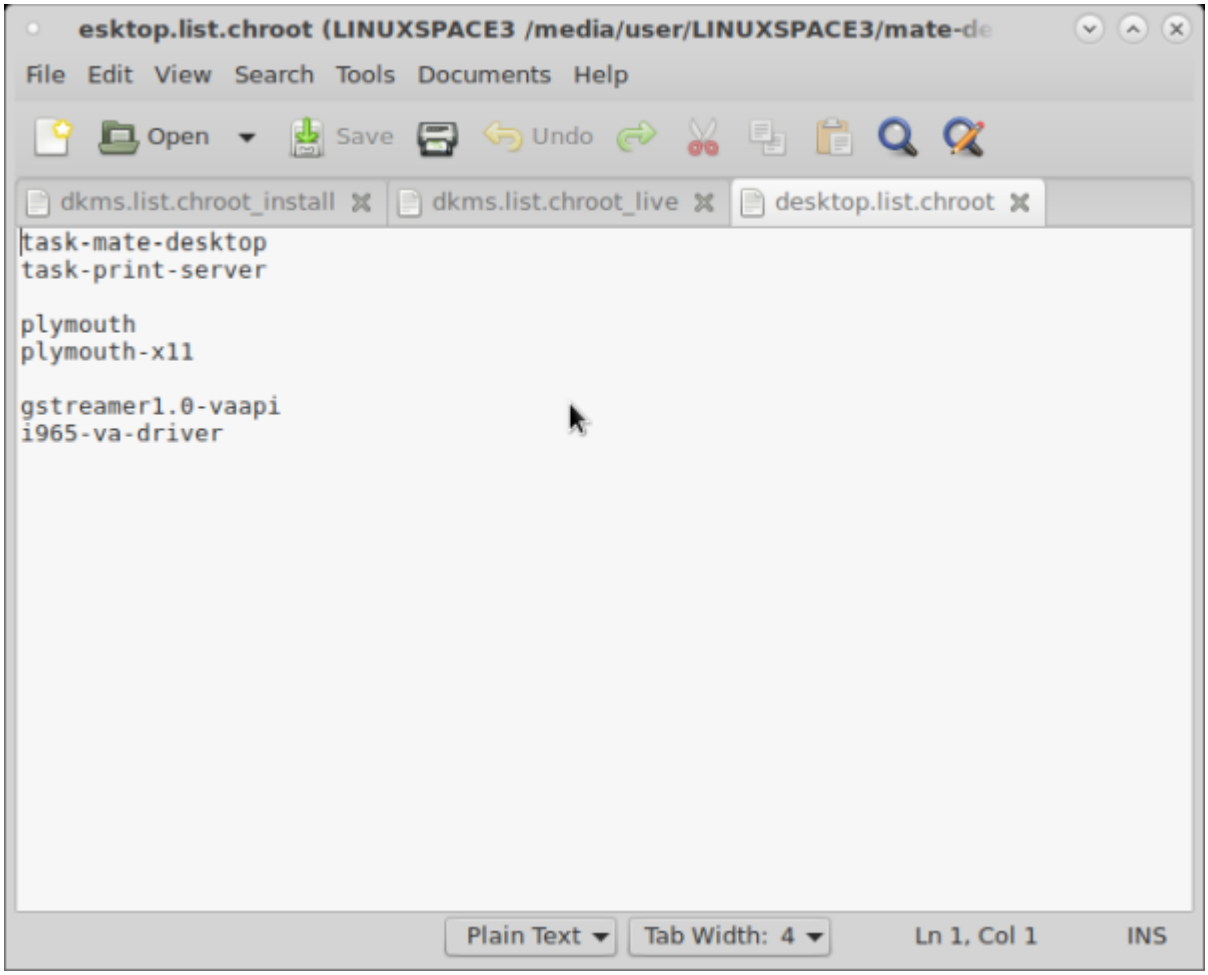


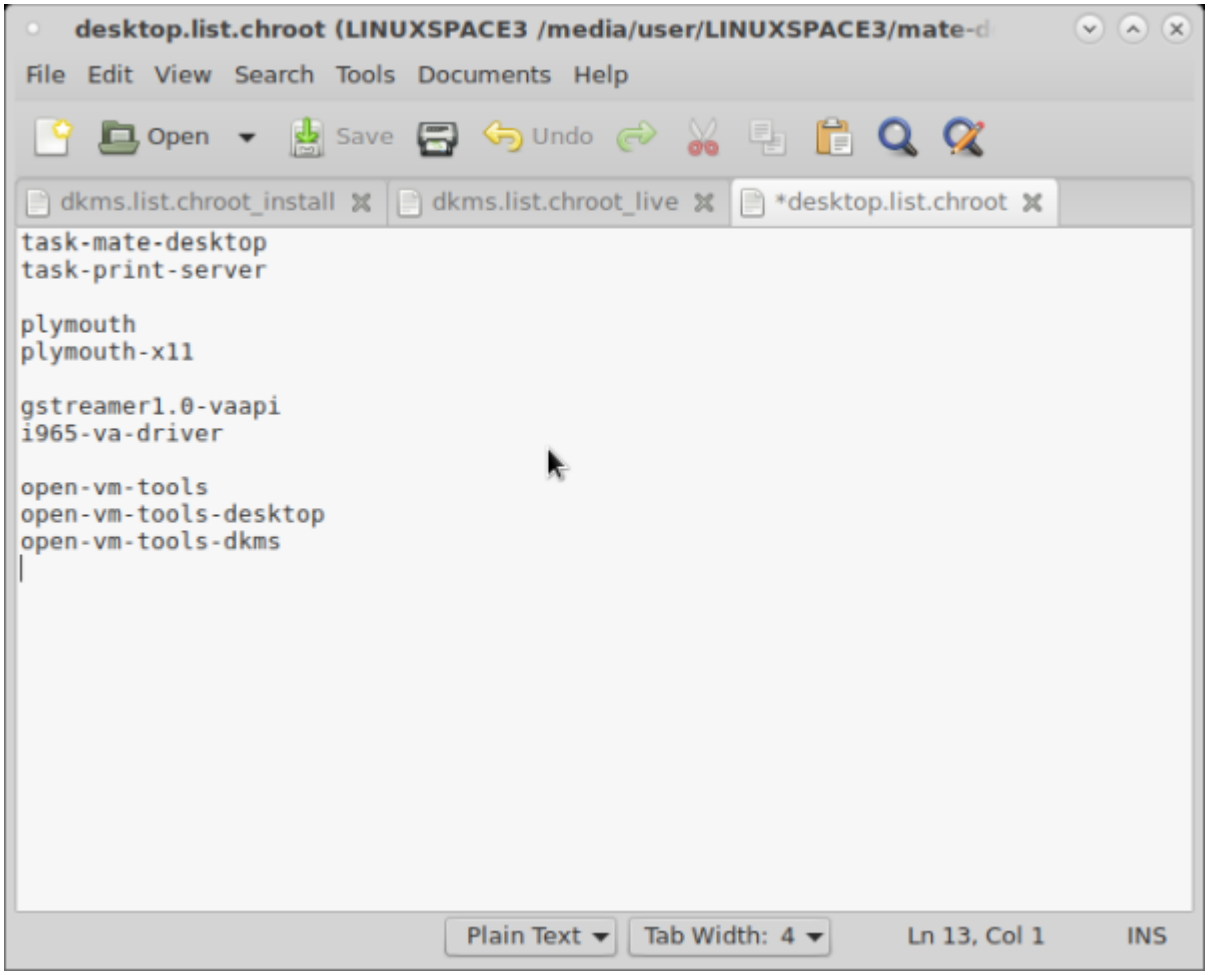
Look unside the two "dkms." files...





The only packages outside the "IFs" might be needed (VMWARE modules) so copy their names into the main list file "desktop.list.chroot".


esktop.list.chroot (LINUXSPACE3 /media/user/LINUXSPACE3/mate-de
File Edit View Search Tools Documents Help
Open
Save
Undo
dkms.list.chroot_install
dkms.list.chroot_live
desktop.list.chroot
task-mate-desktop
task-print-server
plymouth
plymouth-x11
gstreamer1.0-vaapi
i965-va-driver
Plain Text
Tab Width: 4
Ln 1, Col 1
INS



(Afterwards, save the file).

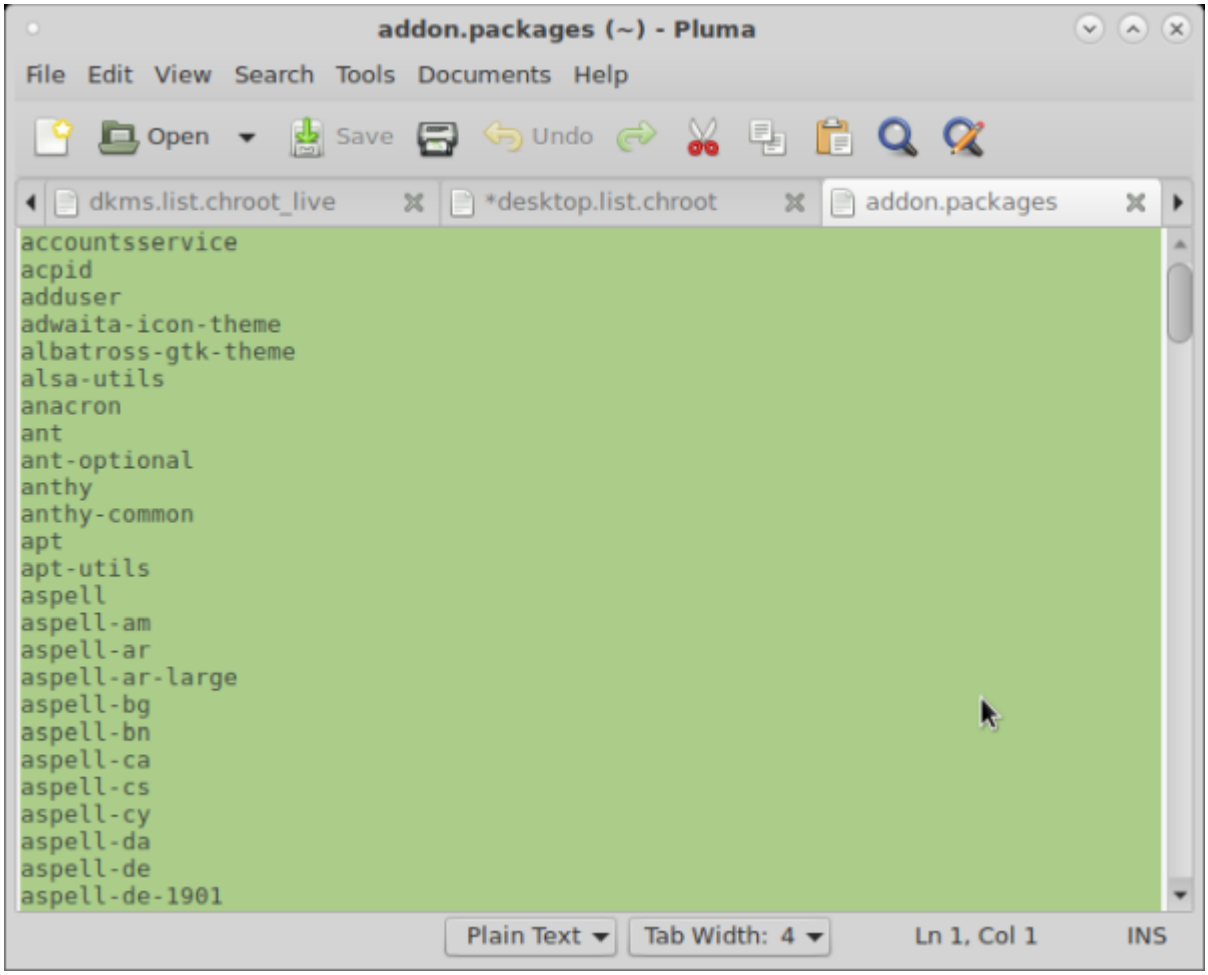
(go back to narrative)

8. Customize the remix by adding additional packages.

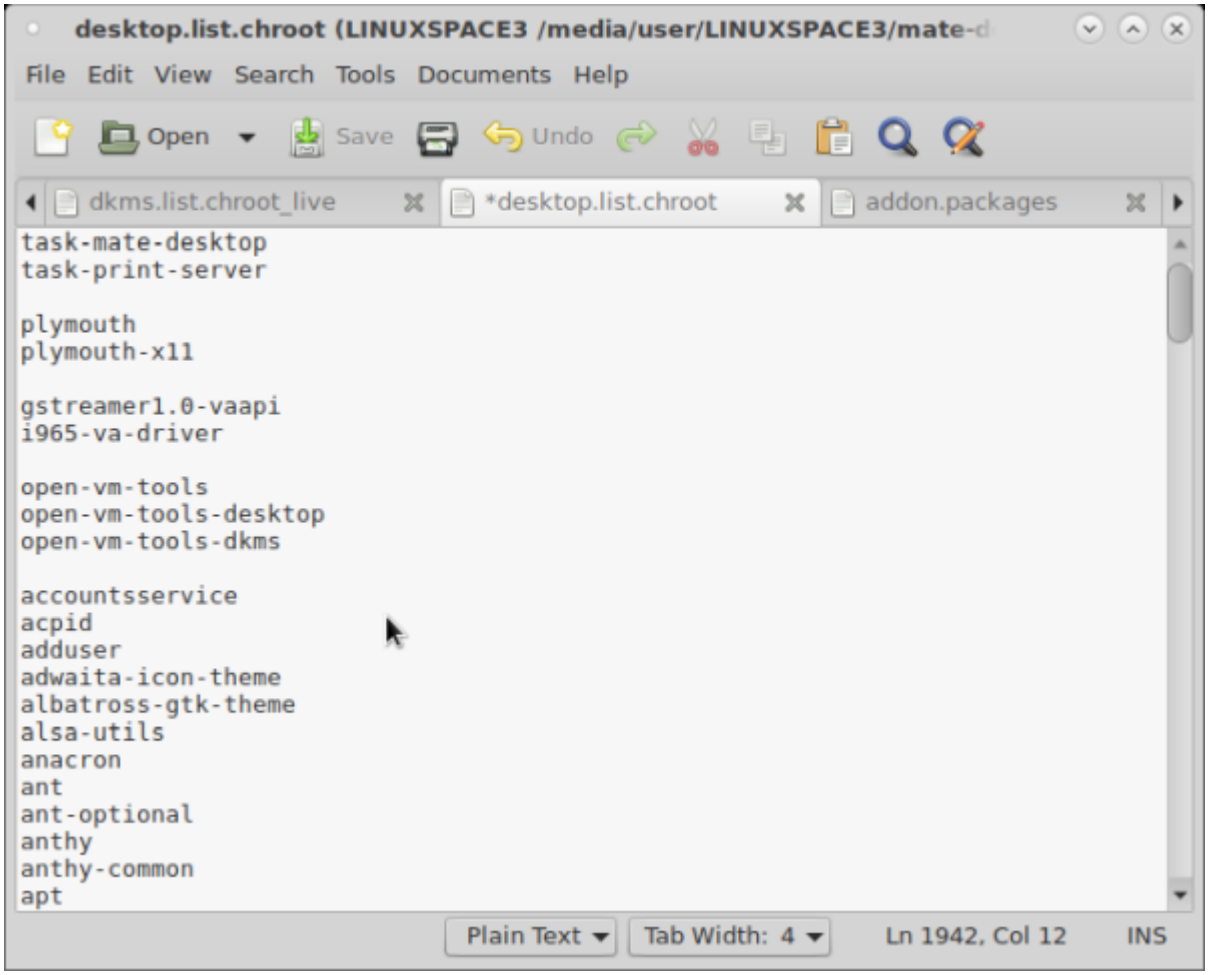
This is where you can customize the applications installed into the Remix by adding extra package names to the "desktop.list.chroot" file. You may add any package names you like.

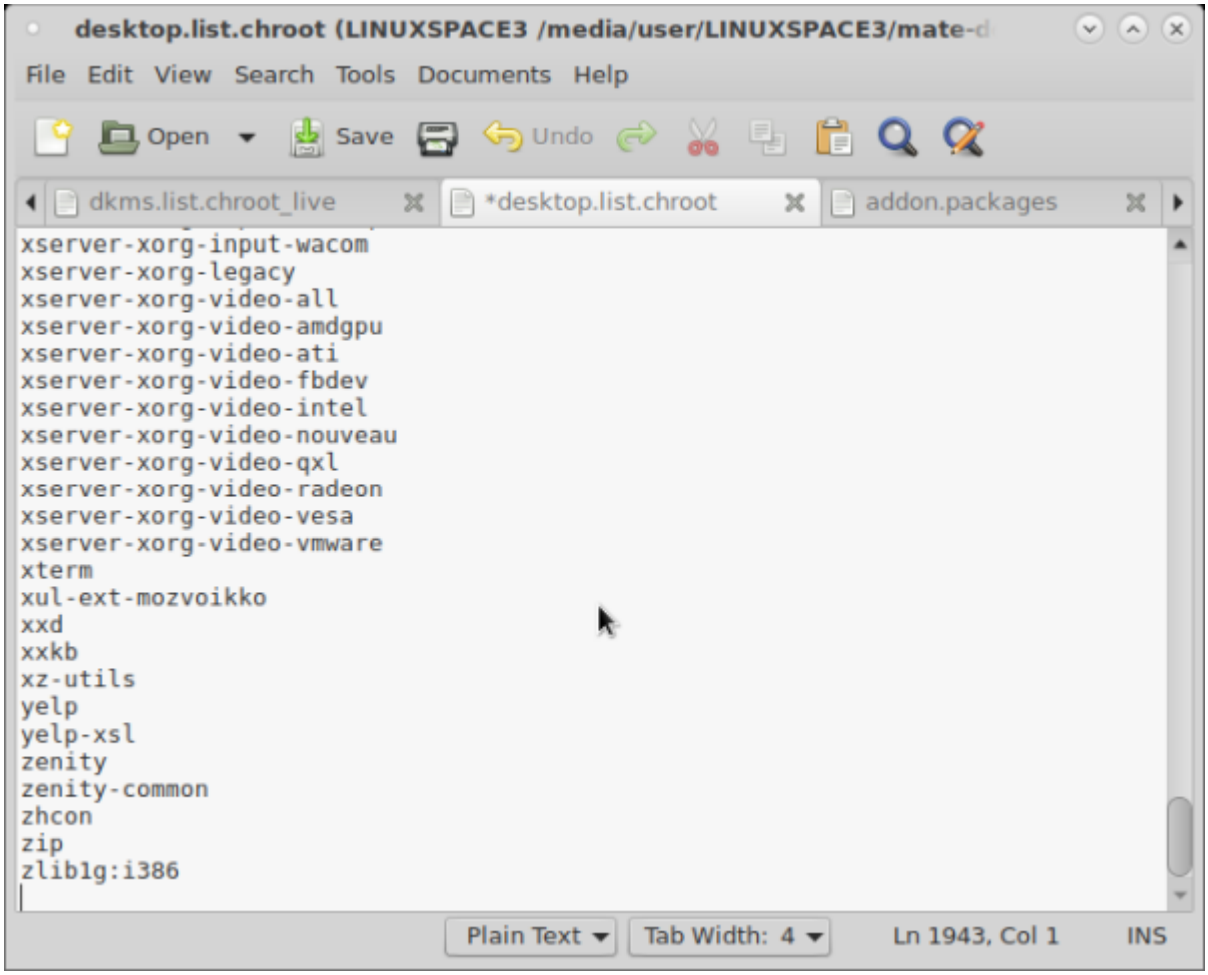
The existing (obsolete from 2015) configuration set specified the build options to generate the Debian Mate 8 release. It does not include all packages now included in the Debian Mate 9 official release. However, as an experiment, we will attempt to replicate the functionality of the current Debian 9 release using the package list.

Use the previously created inventory file to get a (long) list of package names to append to the "desktop.list.chroot" file. Any packages missing from the original configuration which are found in the appended list will be installed. Any duplicates will be skipped over. This will effectively replicate the functionality of the current official Debian 9 release in the Remix.

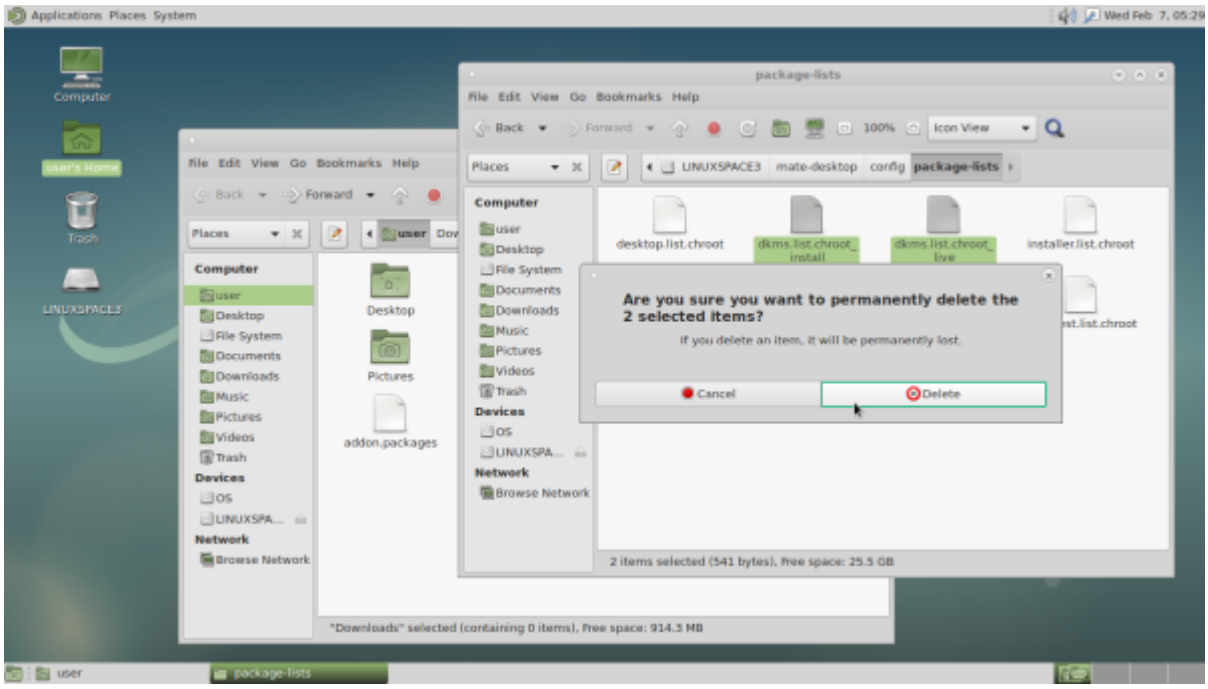


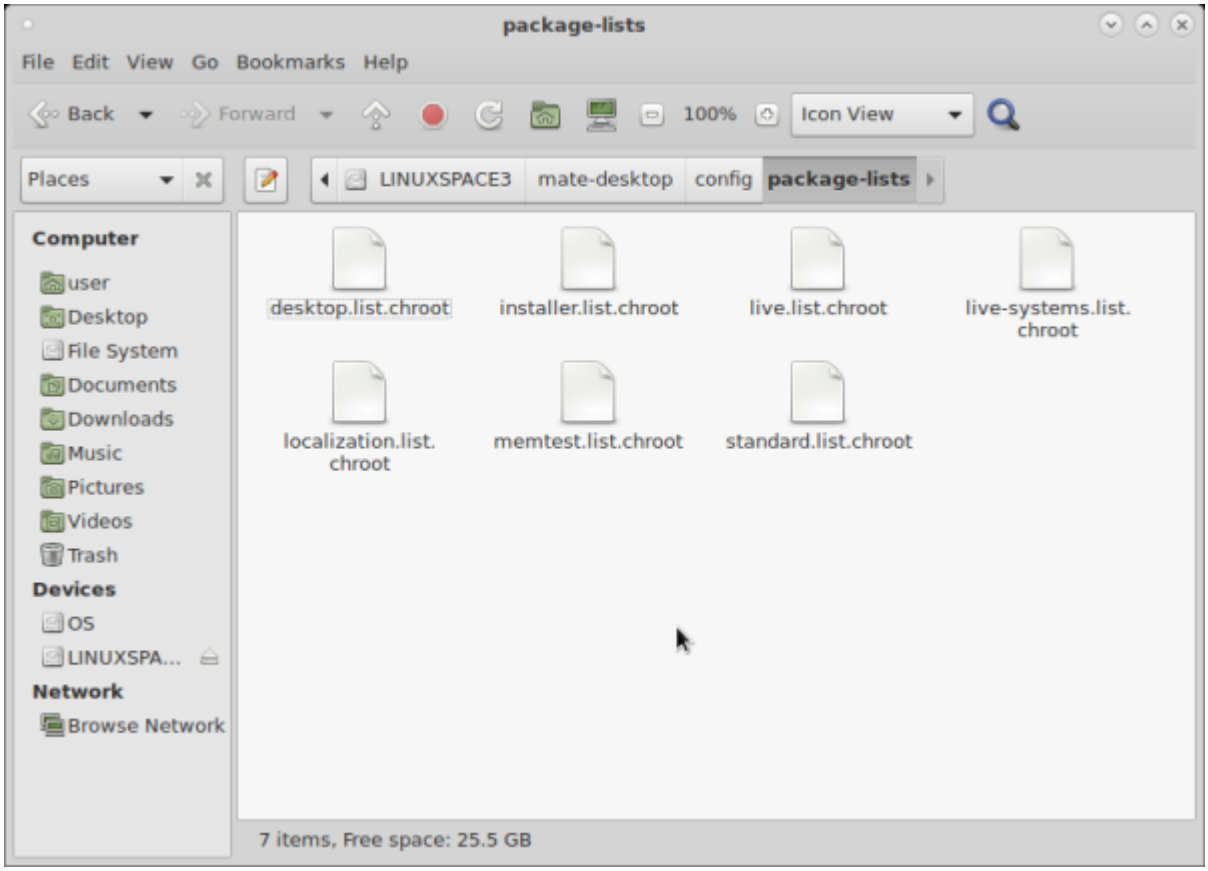
Cut and paste the original inventory list to append it to the basic Debian Mate configuration.


desktop.list.chroot (LINUXSPACE3 /media/user/LINUXSPACE3/mate-d
File Edit View Search Tools Documents Help
Open
Save
Undo
dkms.list.chroot_live
*desktop.list.chroot
addon.packages
task-mate-desktop
task-print-server
plymouth
plymouth-x11
gstreamer1.0-vaapi
i965-va-driver
open-vm-tools
open-vm-tools-desktop
open-vm-tools-dkms
accountsservice
acpid
adduser
adwaita-icon-theme
albatross-gtk-theme
alsa-utils
anacron
ant
ant-optional
anthy
anthy-common
apt
Plain Text
Tab Width: 4
Ln 1942, Col 12
INS

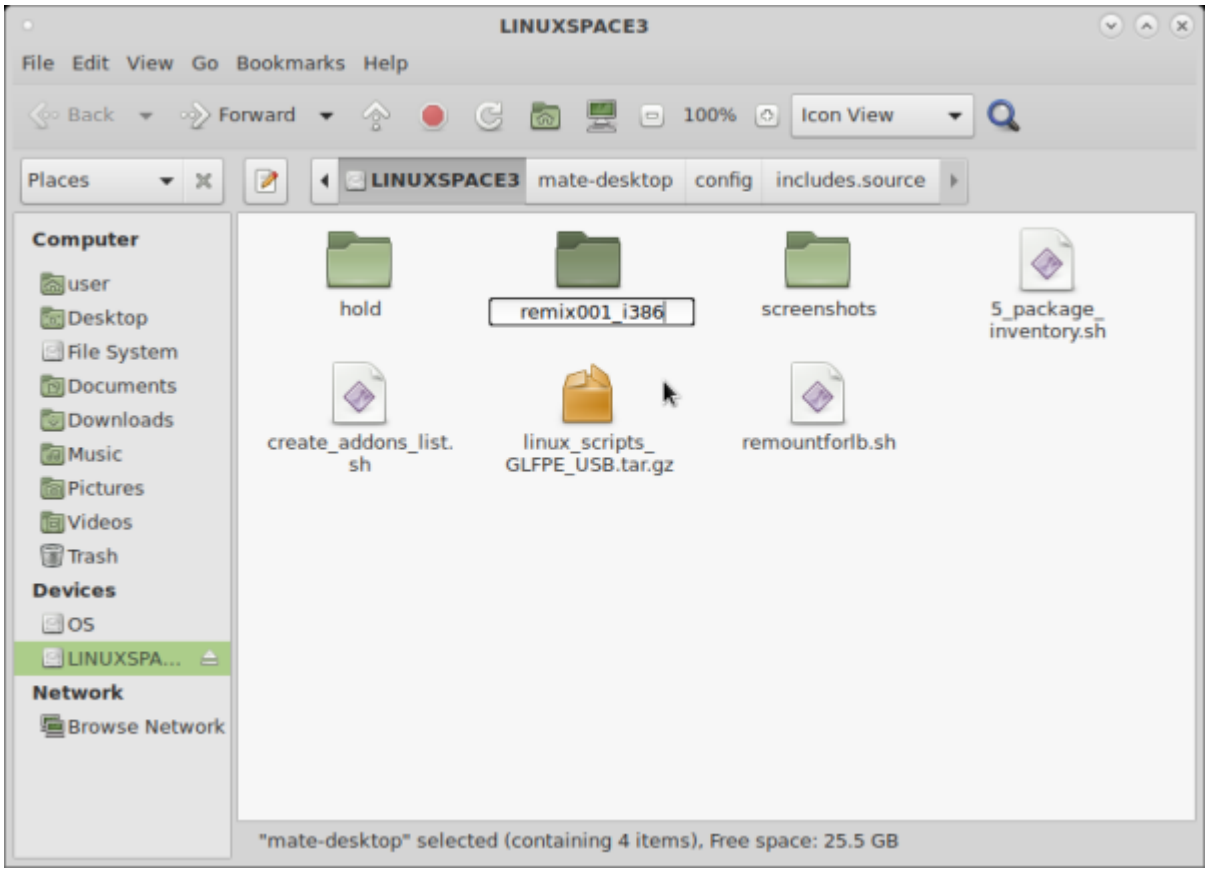


Save the final package list as edited. Now make sure you delete the two "dkms.list" files, or the build will fail (don't know why, but it's true).





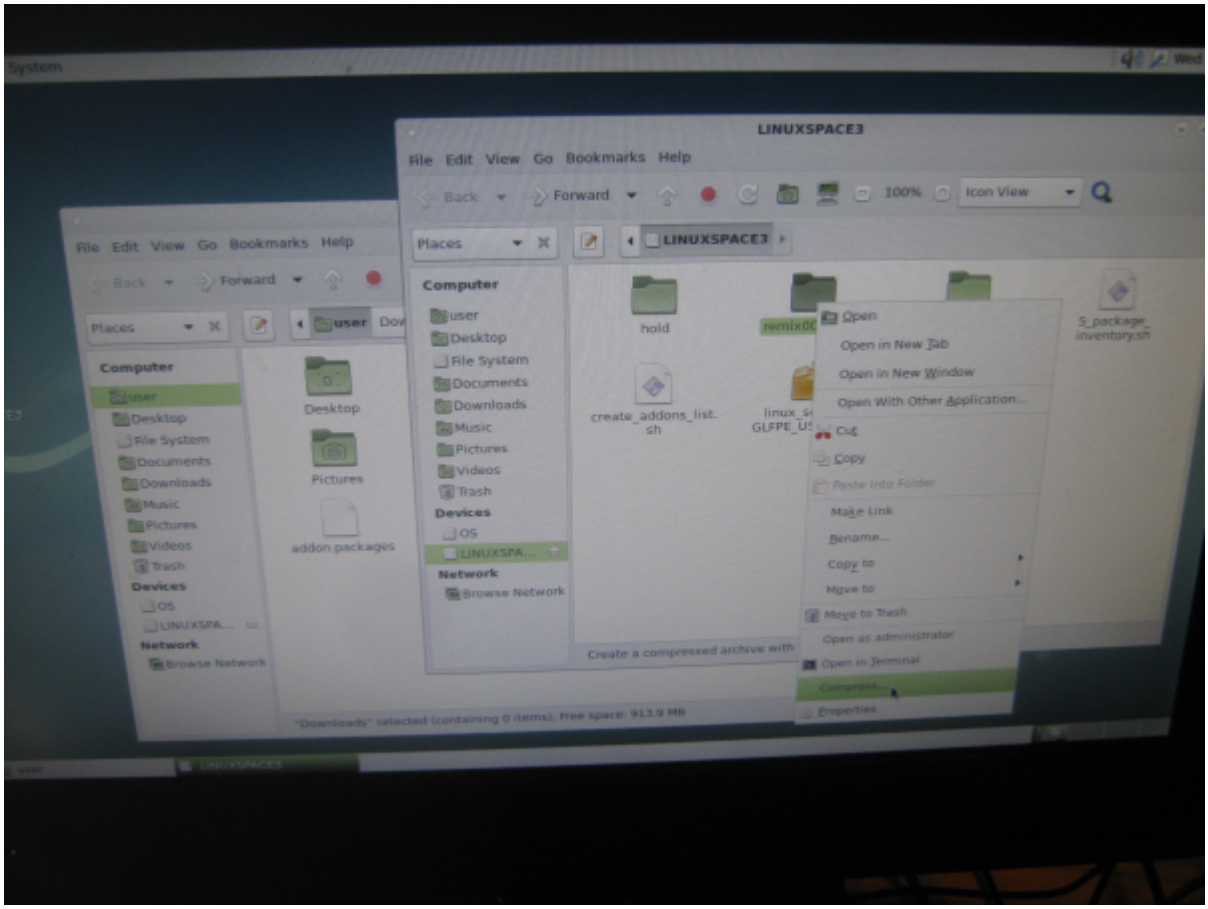
Rename the top level configuration directory to reflect your new Remix name.

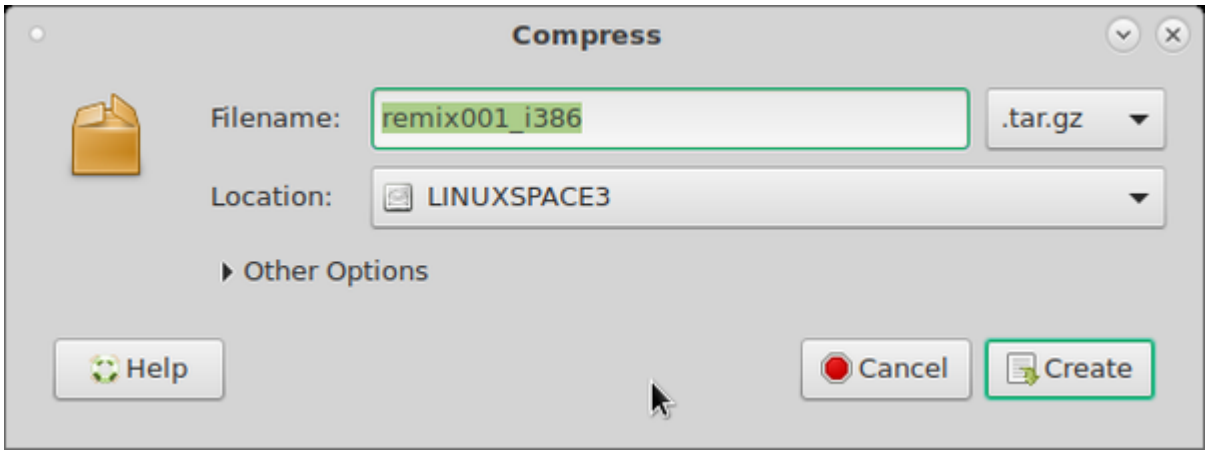


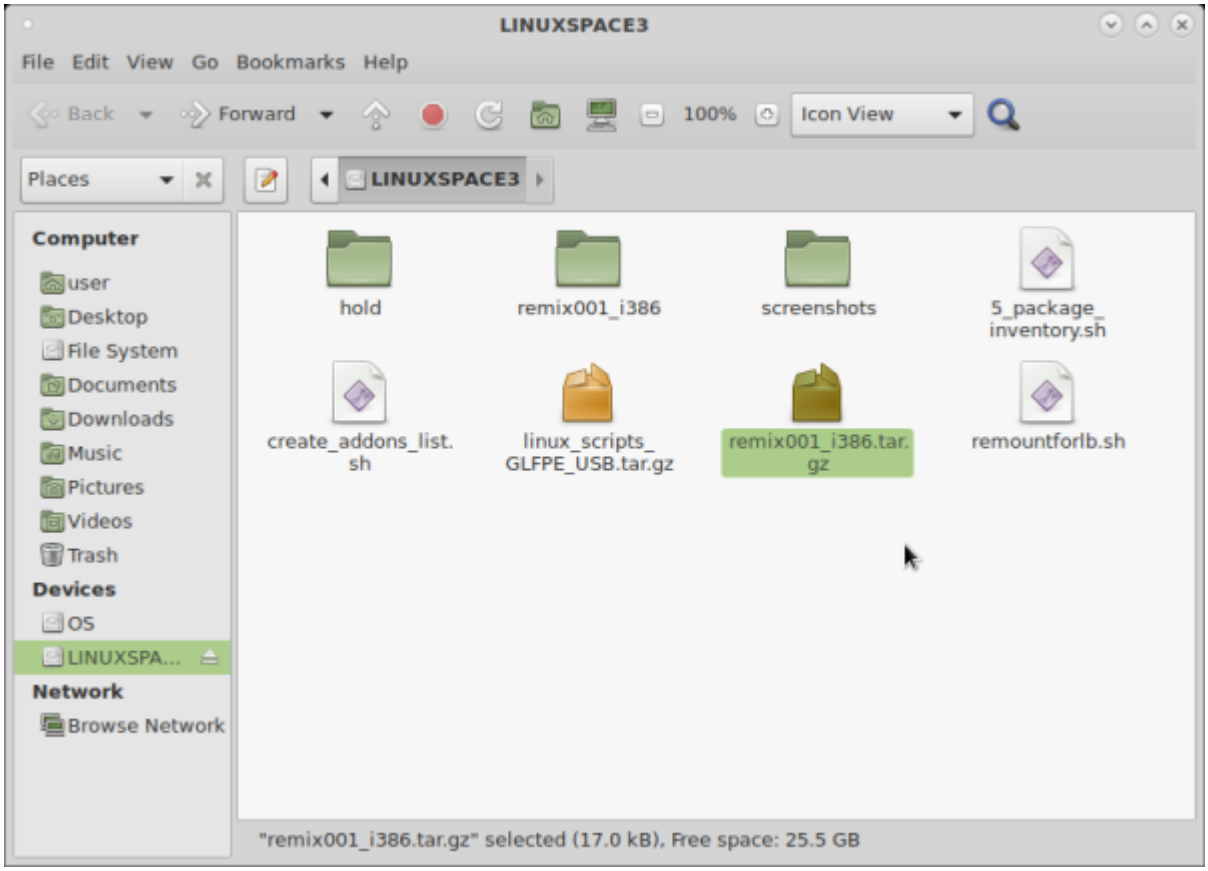
(go back to narrative)

9. SAVE YOUR CONFIGURATION using the file archiver.

Right-click the icon for your new configuration folder, select Compress. Archive to a .tar.gz file to save your configuration. Assuming there are no errors, you can use it later by simply unpacking it.







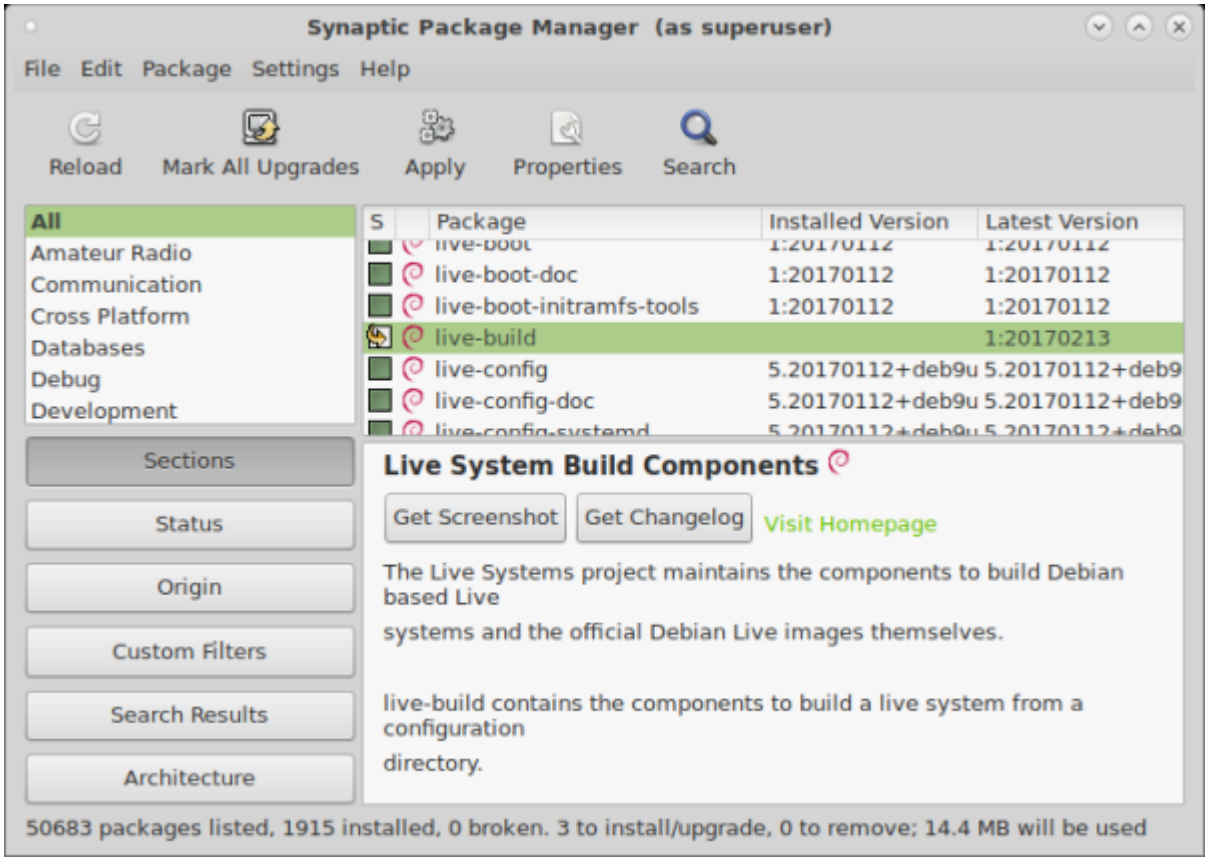
(go back to narrative)

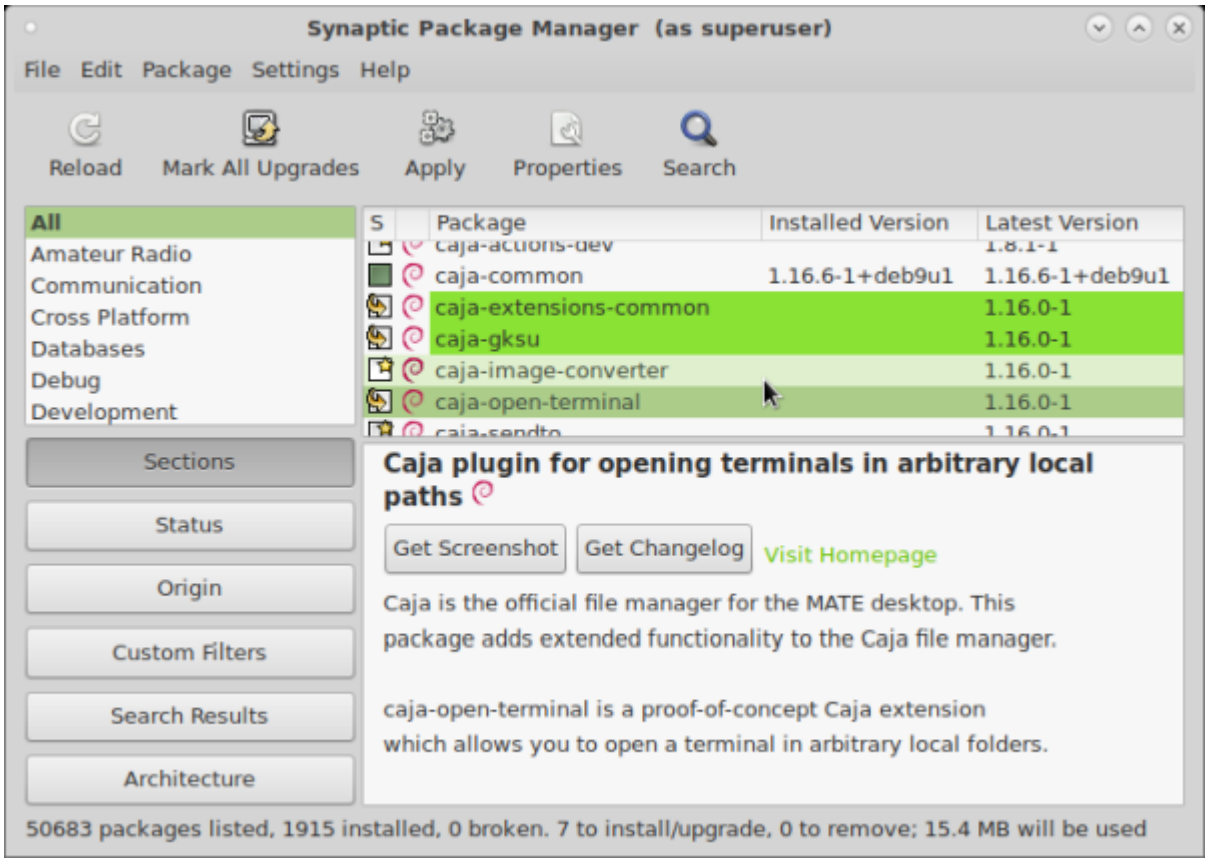
10. Reboot the computer if you wish.

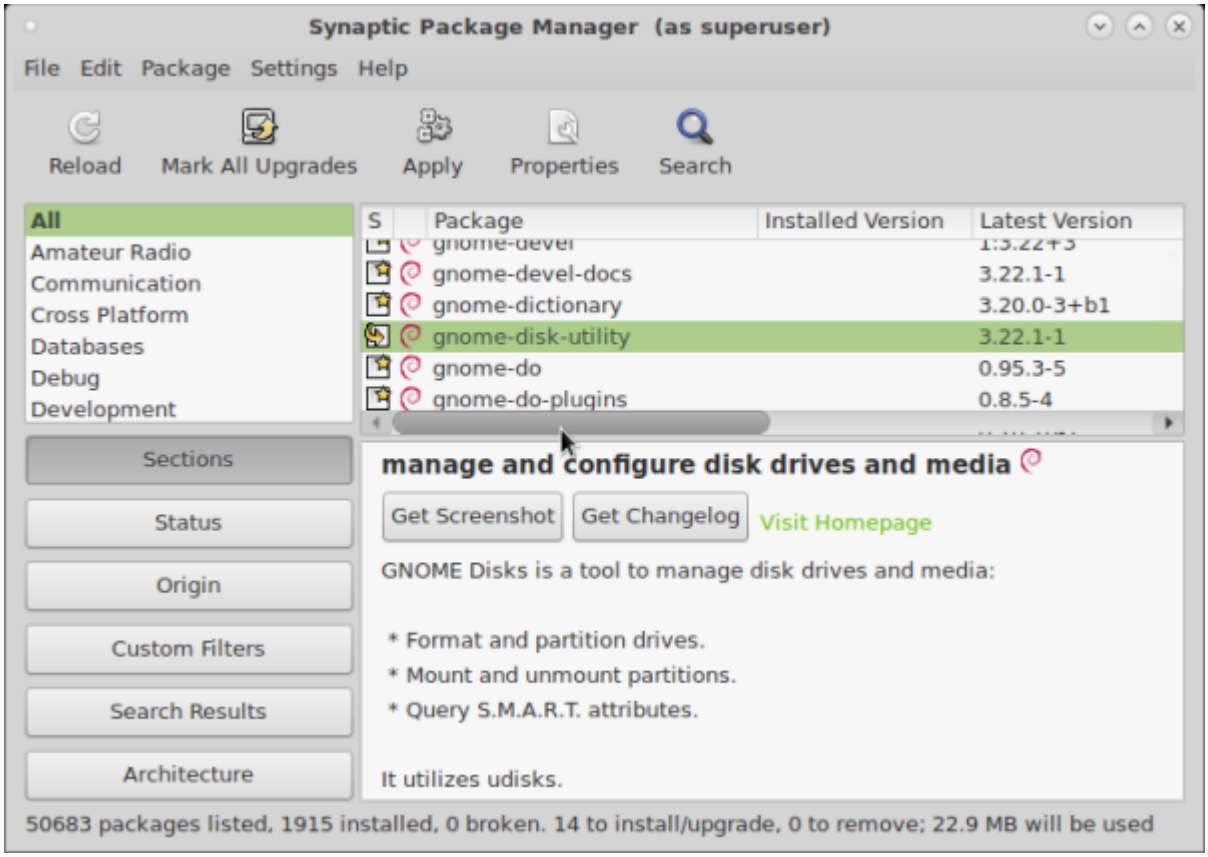
(go back to narrative)

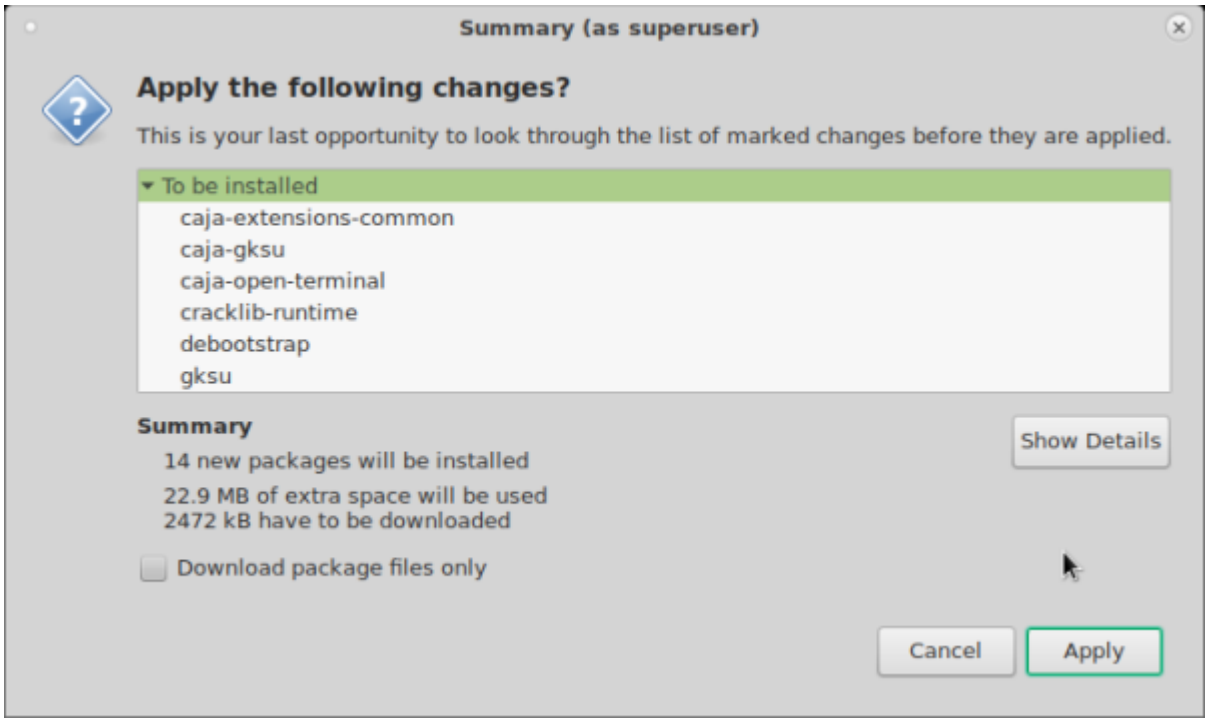
11. Live boot the Debian OS you wish to use to build the remix. Reinstall the necessary packages, since rebooting wiped them out.

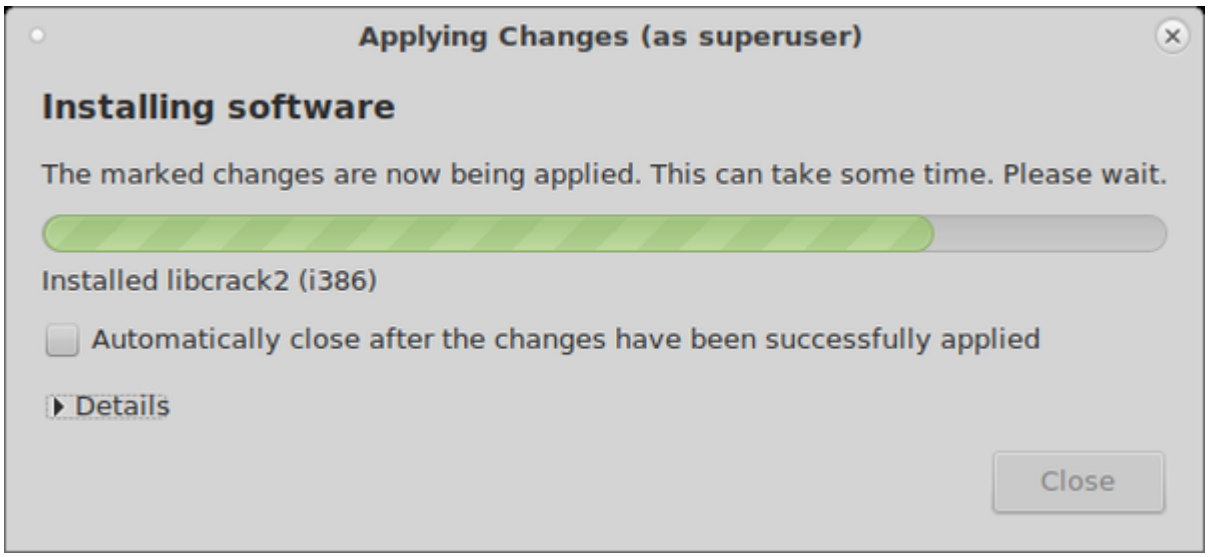
Install the necessary live-build, and the recommended caja-gksu, caja-open-terminal, and gnome-disk-utility packages.


Synaptic Package Manager (as superuser)
File Edit Package Settings Help
Reload
Mark All Upgrades
Apply
Properties
Search
All
Amateur Radio
Communication
Cross Platform
Databases
Debug
Development
Sections
Status
Origin
Custom Filters
Search Results
Architecture
S
Package
Installed Version
Latest Version
live-boot-doc 1:20170112 1:20170112
live-boot-initramfs-tools 1:20170112 1:20170112
live-build 1:20170213
live-config 5.20170112+deb9u 5.20170112+deb9
live-config-doc 5.20170112+deb9u 5.20170112+deb9
Live System Build Components
Get Screenshot
Get Changelog
Visit Homepage
The Live Systems project maintains the components to build Debian based Live
systems and the official Debian Live images themselves.
live-build contains the components to build a live system from a configuration
directory.
50683 packages listed, 1915 installed, 0 broken. 3 to install/upgrade, 0 to remove; 14.4 MB will be used

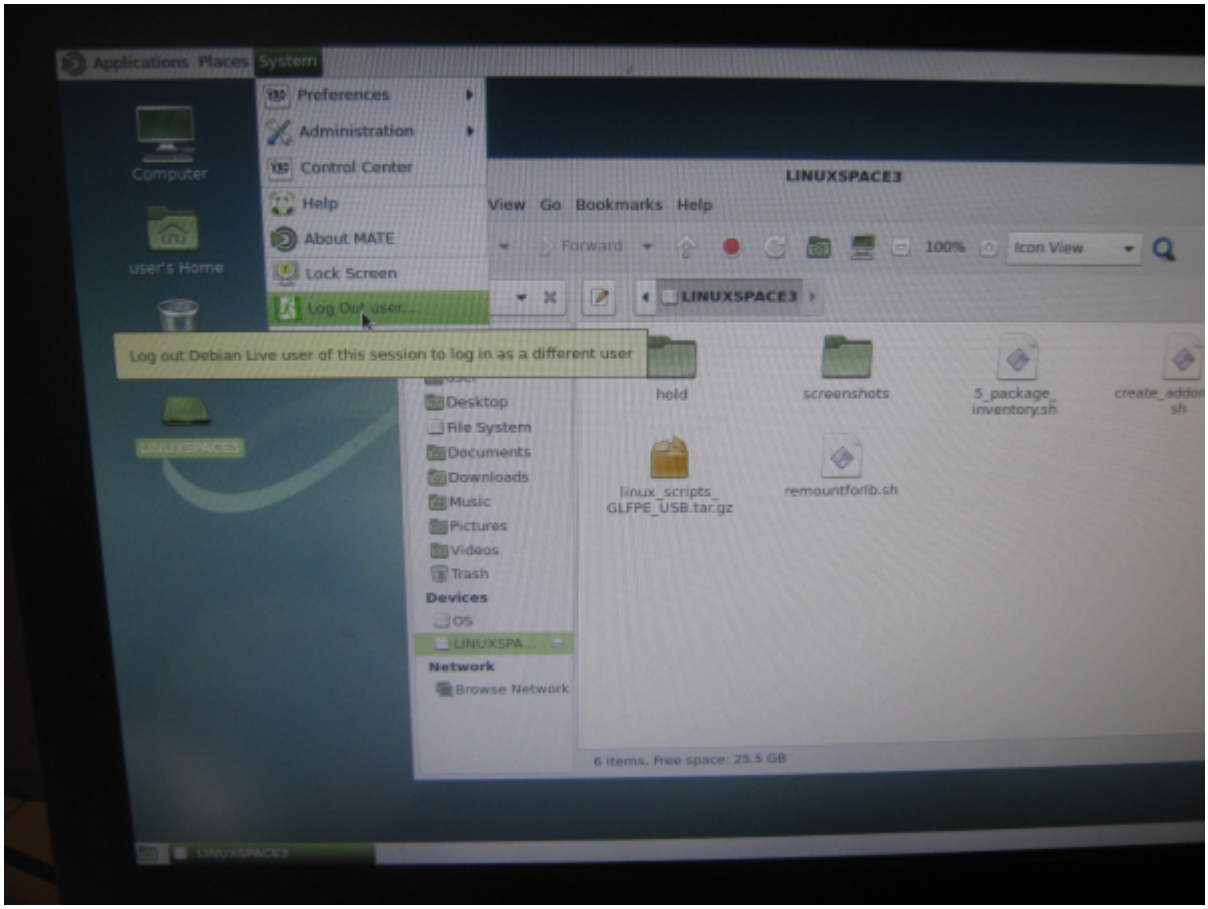

Synaptic Package Manager (as superuser)
File Edit Package Settings Help
Reload
Mark All Upgrades
Apply
Properties
Search
All
Amateur Radio
Communication
Cross Platform
Databases
Debug
Development
Sections
Status
Origin
Custom Filters
Search Results
Architecture
S
Package
Installed Version
Latest Version
caja-common 1.16.6-1+deb9u1 1.16.6-1+deb9u1
caja-extensions-common 1.16.0-1
caja-gksu 1.16.0-1
caja-image-converter 1.16.0-1
caja-open-terminal 1.16.0-1
Caja plugin for opening terminals in arbitrary local paths
Get Screenshot
Get Changelog
Visit Homepage
Caja is the official file manager for the MATE desktop. This
package adds extended functionality to the Caja file manager.
caja-open-terminal is a proof-of-concept Caja extension
which allows you to open a terminal in arbitrary local folders.
50683 packages listed, 1915 installed, 0 broken. 7 to install/upgrade, 0 to remove; 15.4 MB will be used

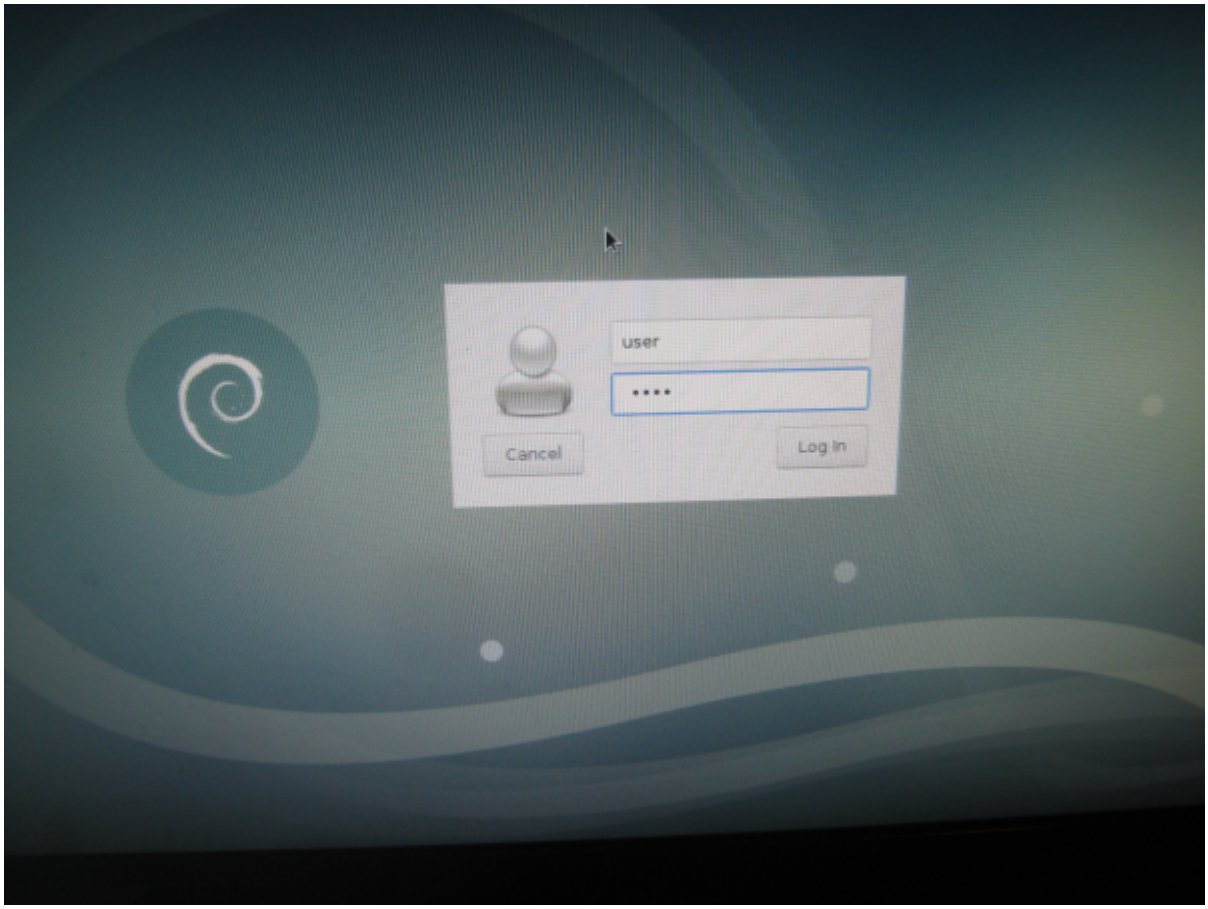

Synaptic Package Manager (as superuser)
File Edit Package Settings Help
Reload
Mark All Upgrades
Apply
Properties
Search
All
Amateur Radio
Communication
Cross Platform
Databases
Debug
Development
Sections
Status
Origin
Custom Filters
Search Results
Architecture
S
Package
Installed Version
Latest Version
gnome-devel-docs 3.22.1-1
gnome-dictionary 3.20.0-3+b1
gnome-disk-utility 3.22.1-1
gnome-do 0.95.3-5
gnome-do-plugins 0.8.5-4
manage and configure disk drives and media
Get Screenshot
Get Changelog
Visit Homepage
GNOME Disks is a tool to manage disk drives and media:
* Format and partition drives.
* Mount and unmount partitions.
* Query S.M.A.R.T. attributes.
It utilizes udisks.
50683 packages listed, 1915 installed, 0 broken. 14 to install/upgrade, 0 to remove; 22.9 MB will be used


Summary (as superuser)
Apply the following changes?
This is your last opportunity to look through the list of marked changes before they are applied.
To be installed
caja-extensions-common
caja-gksu
caja-open-terminal
cracklib-runtime
debootstrap
gksu
Summary
14 new packages will be installed
22.9 MB of extra space will be used
2472 kB have to be downloaded
Show Details
Download package files only
Cancel
Apply



Assuming you installed the "caja" file manager extensions above, you will need to temporarily log out and log back in to get them to take effect. Those extensions will get you right click options to "Open in Terminal" and "Open as Administrator". The default Debian user name is "user" and the password is "live".





(go back to narrative)

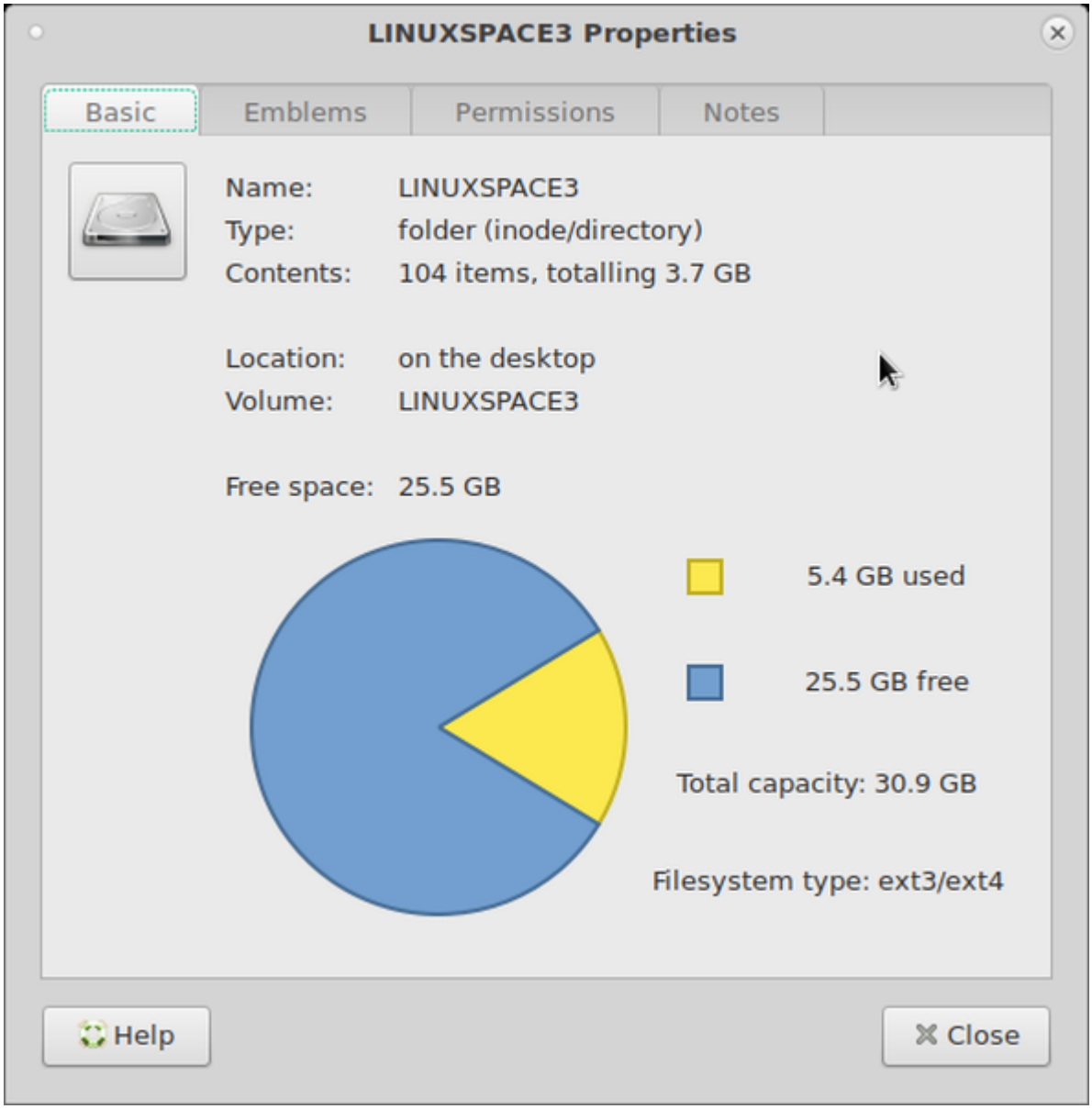
12. Prepare a 32Gb USB work drive -- format as a Linux EXT4 disk.

If you have not yet prepared an EXT4 work drive, acquire an empty (or expendable) USB stick with at least 32Gb capacity. If you installed "gnome-disk-utility" above, you can use it to reformat the work drive to EXT4. Insert the USB stick into an available port. It should mount (probably as a FAT32 formatted drive). Get to the disk utility from the drop-down menu (Applications, Accessories, Disks).

Assuming you now have an EXT4 drive, insert the preformatted USB stick in EXT4 format as a work drive (here named LINUXSPACE3).

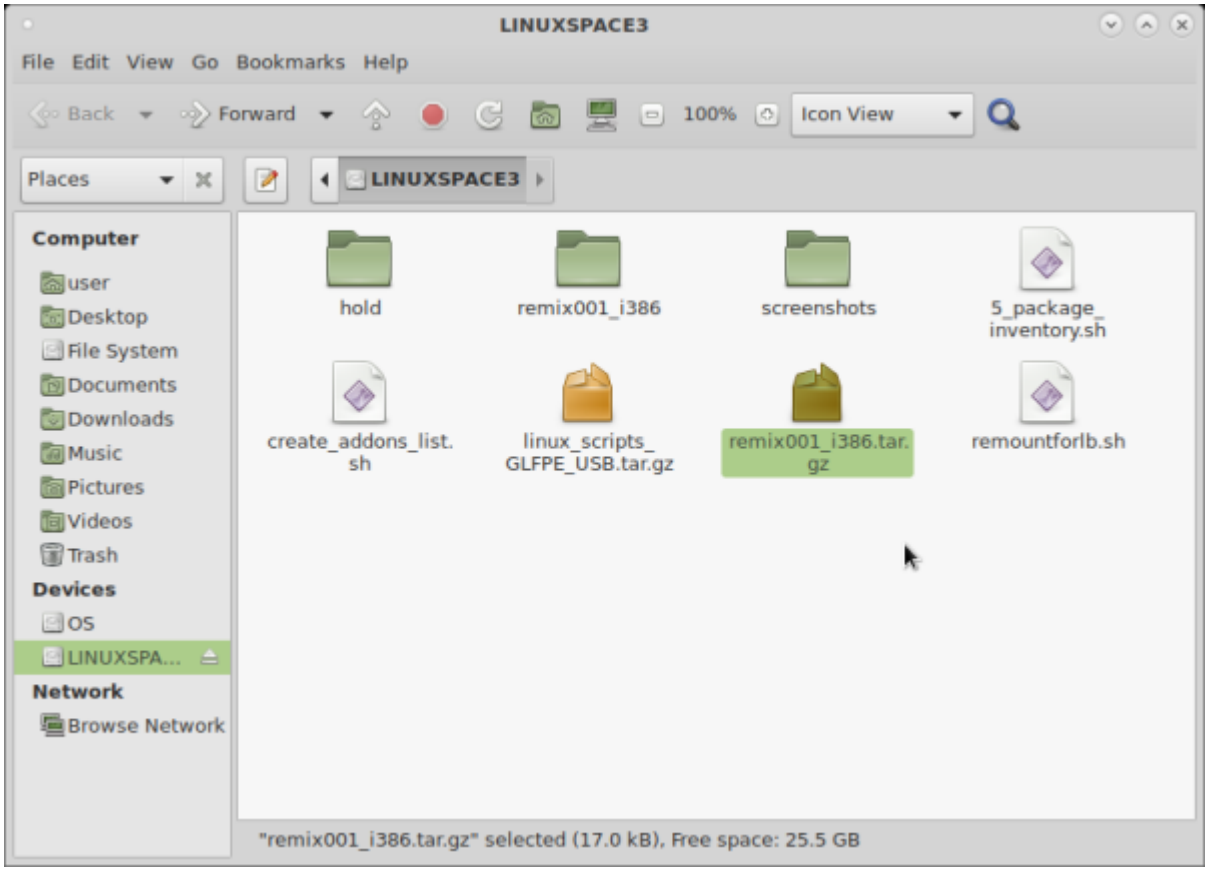


Right-click drive icon, bring up properties, verify is EXT4 formated drive with at least 20 Gb free.



(go back to narrative)

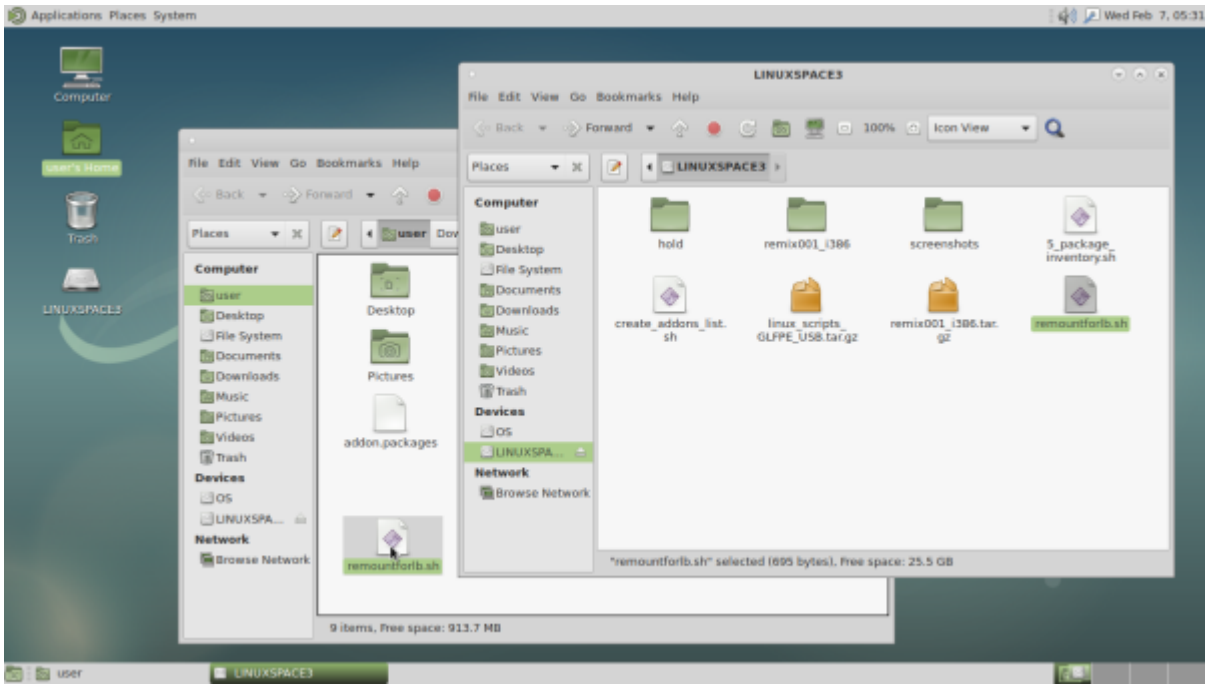
13. Unpack the previously prepared compressed configuration set to the work drive (if not already there). The original configuration could have been prepared and compressed using any drive (home folder or USB stick), but this time it MUST be unpacked to the Linux EXT4 work drive to perform the build.

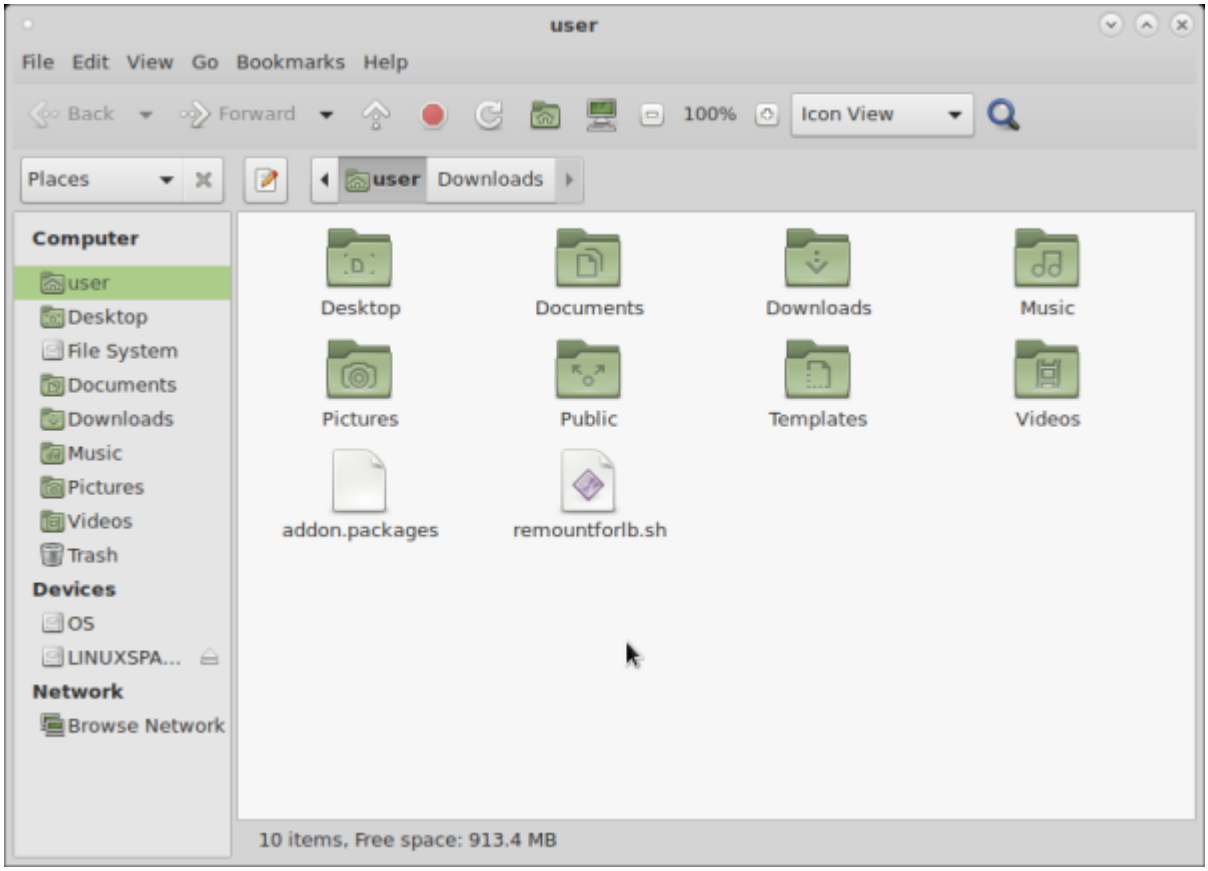


(go back to narrative)

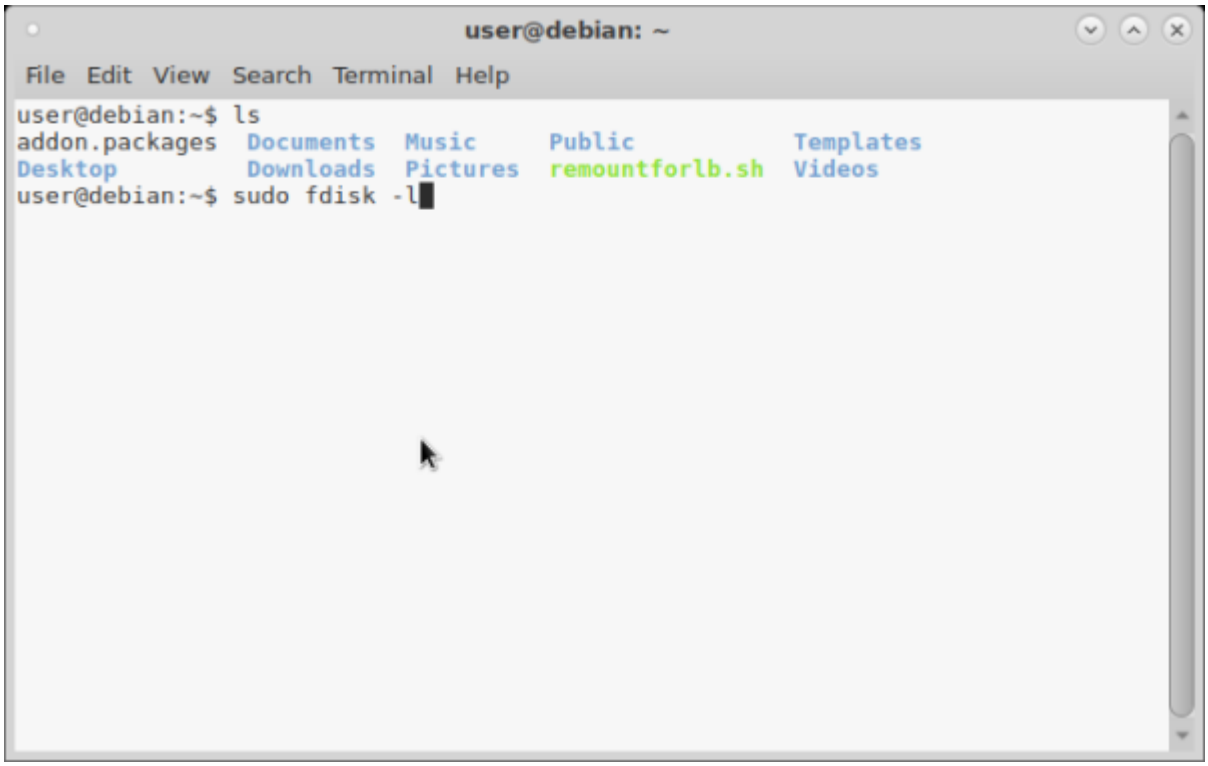
14. Special batch -- remount the new Linux EXT4 work drive with special permissions.

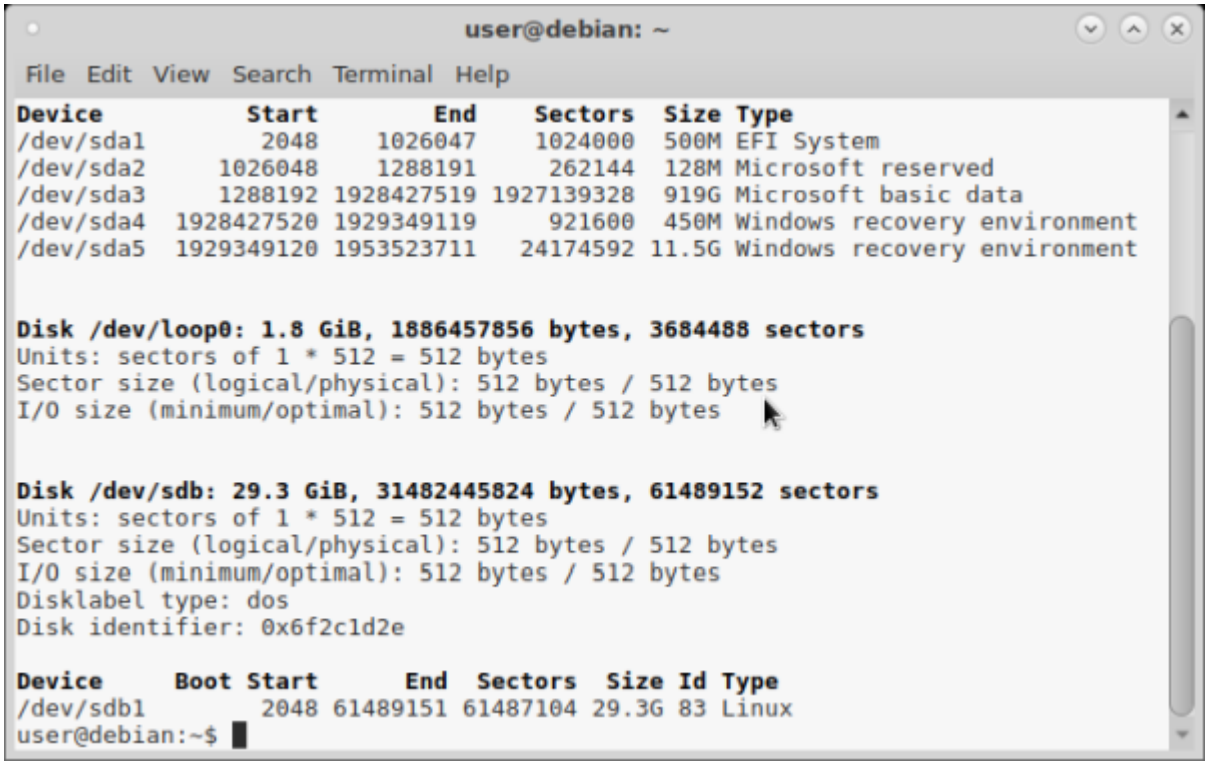
Copy the prepared script "remountforlb.sh" to the user's home folder.



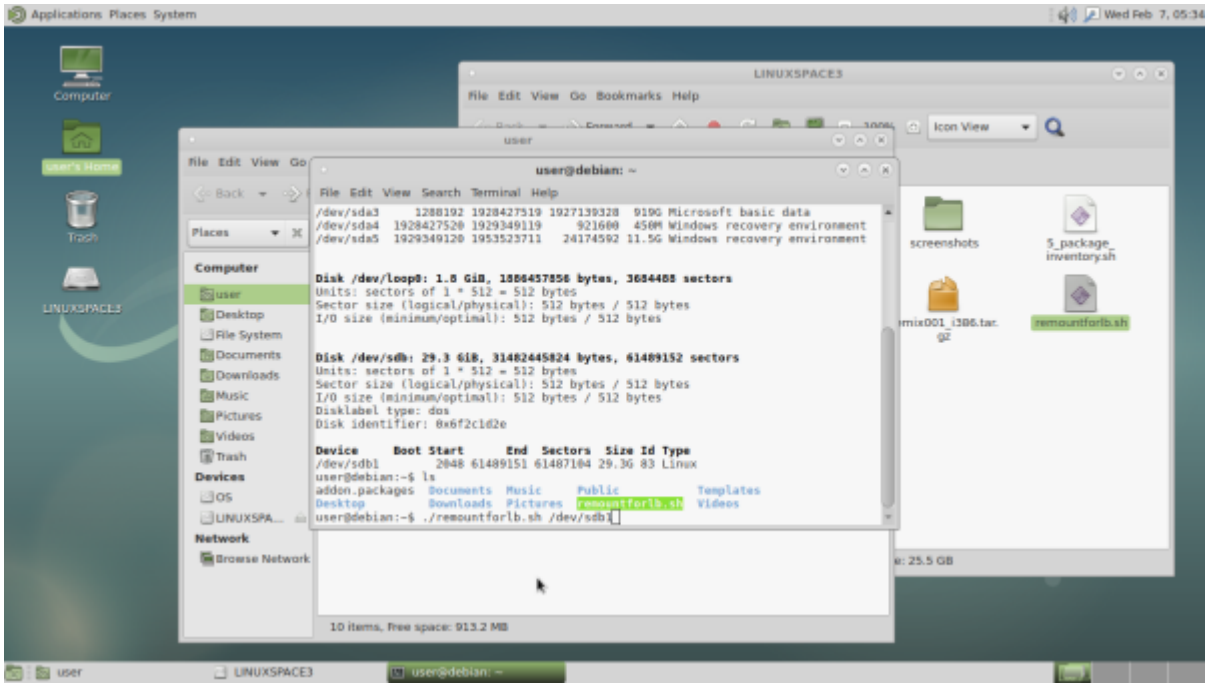


Bring up a terminal (defaults to home folder). Verify the device name of the EXT4 USB workdrive using "sudo fdisk -l". The drive you want is PROBABLY the last one listed, but be sure.

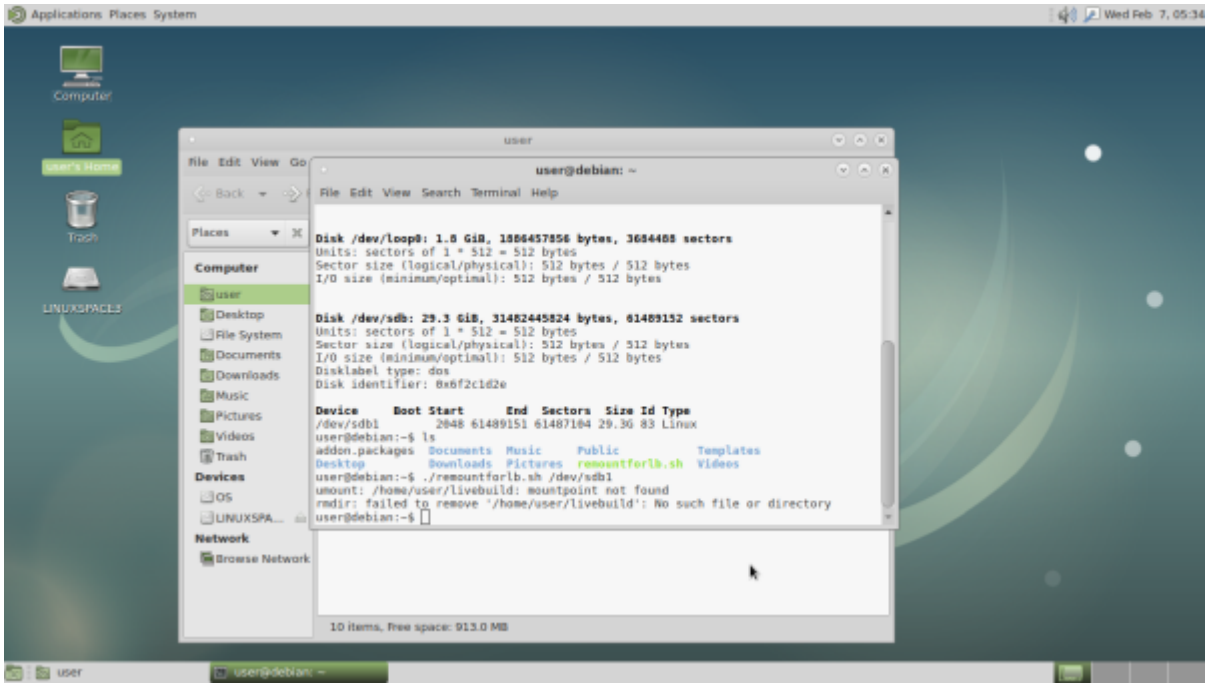




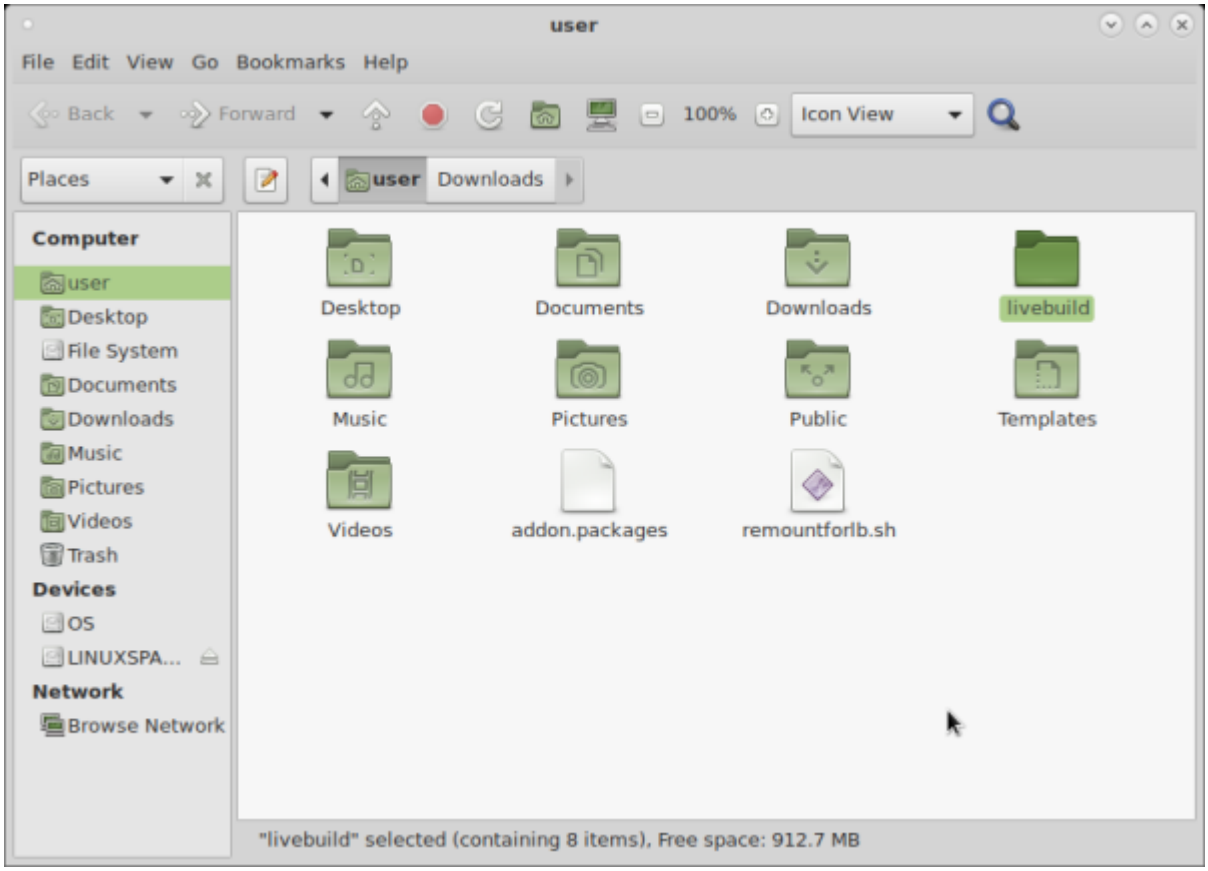
Use the drive device name found (here /dev/sdb1) to perform the remount procedure.



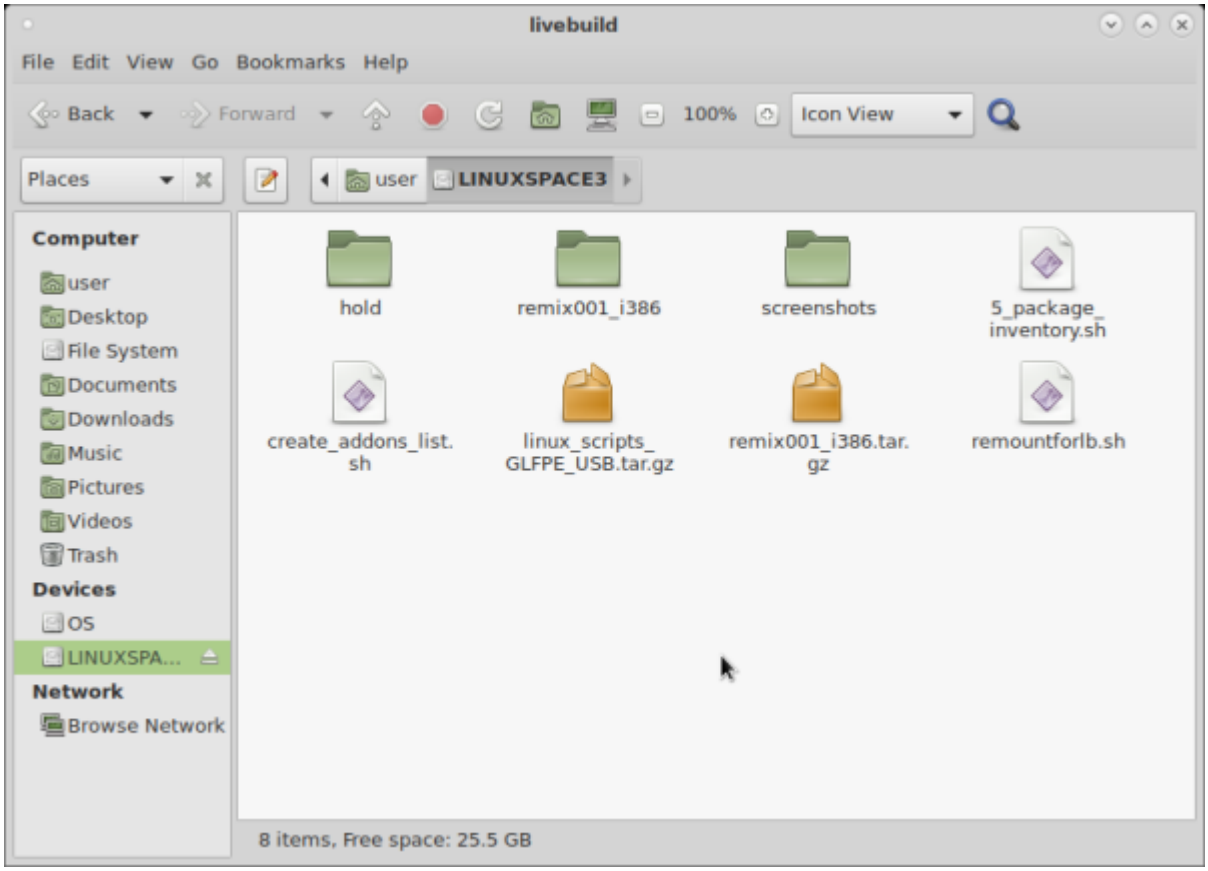
Note how the originally mounted work drive file manager display disappears from view.



Now you should find a "livebuild" folder under the home directory...



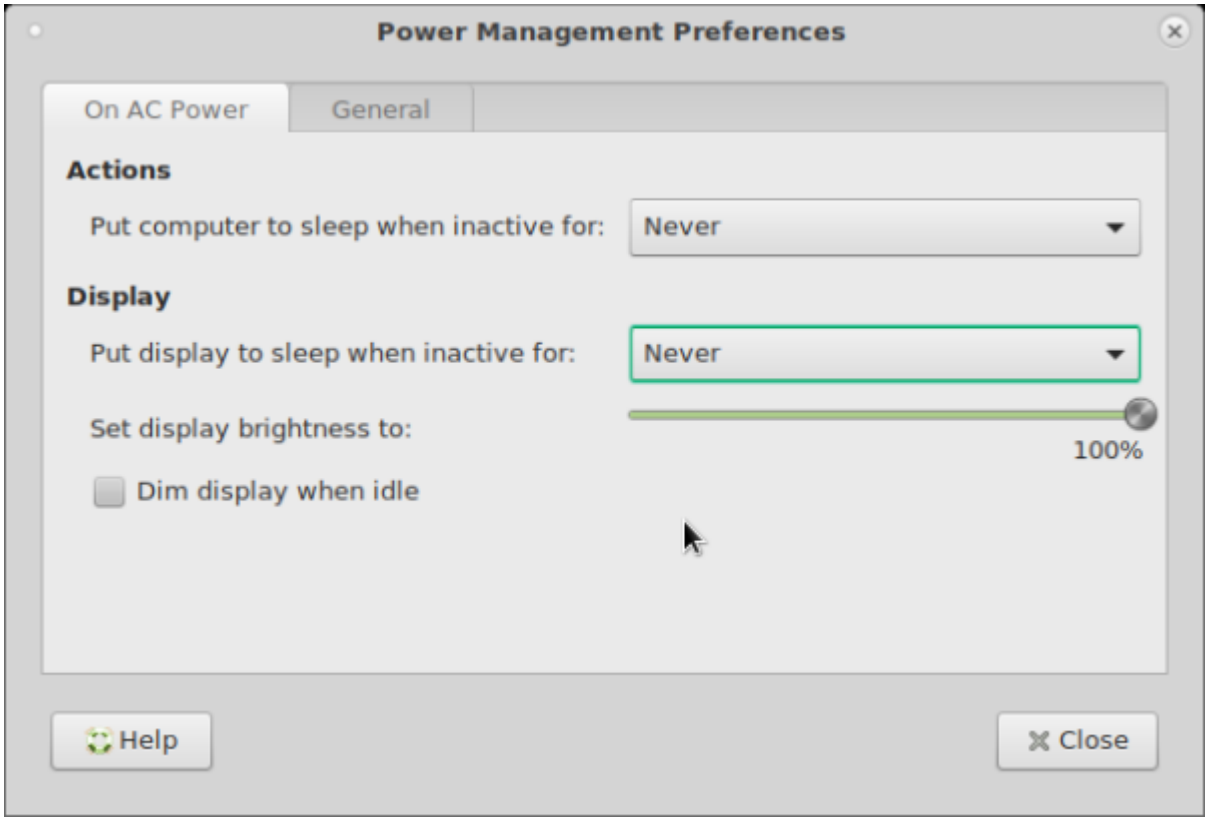
...and when you look underneath the "livebuild" folder, you now see the entire contents of the original work drive. If the remount was successful, the drive will now have additional permissions (dev,exec) necessary to the build.

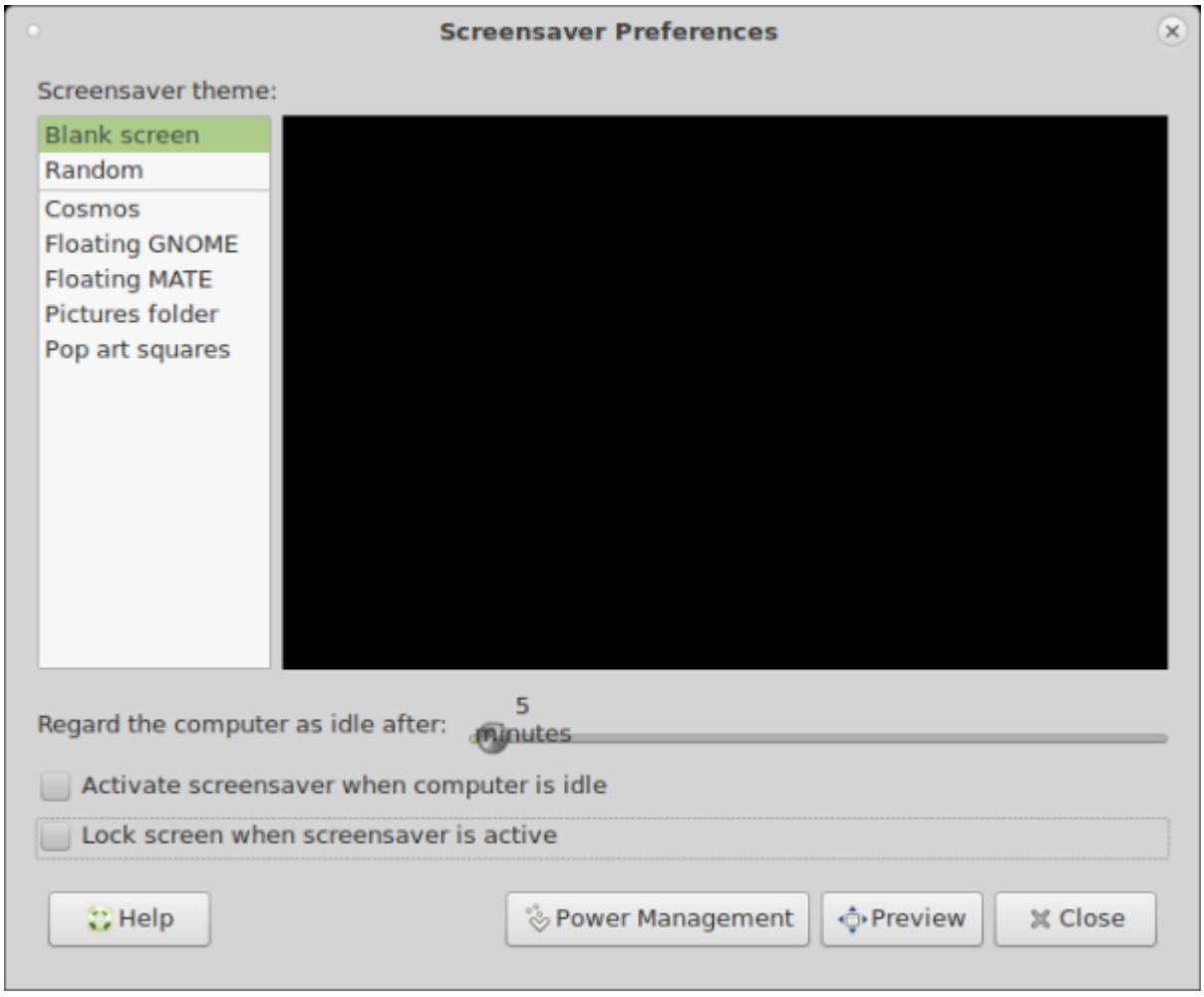


(go back to narrative)

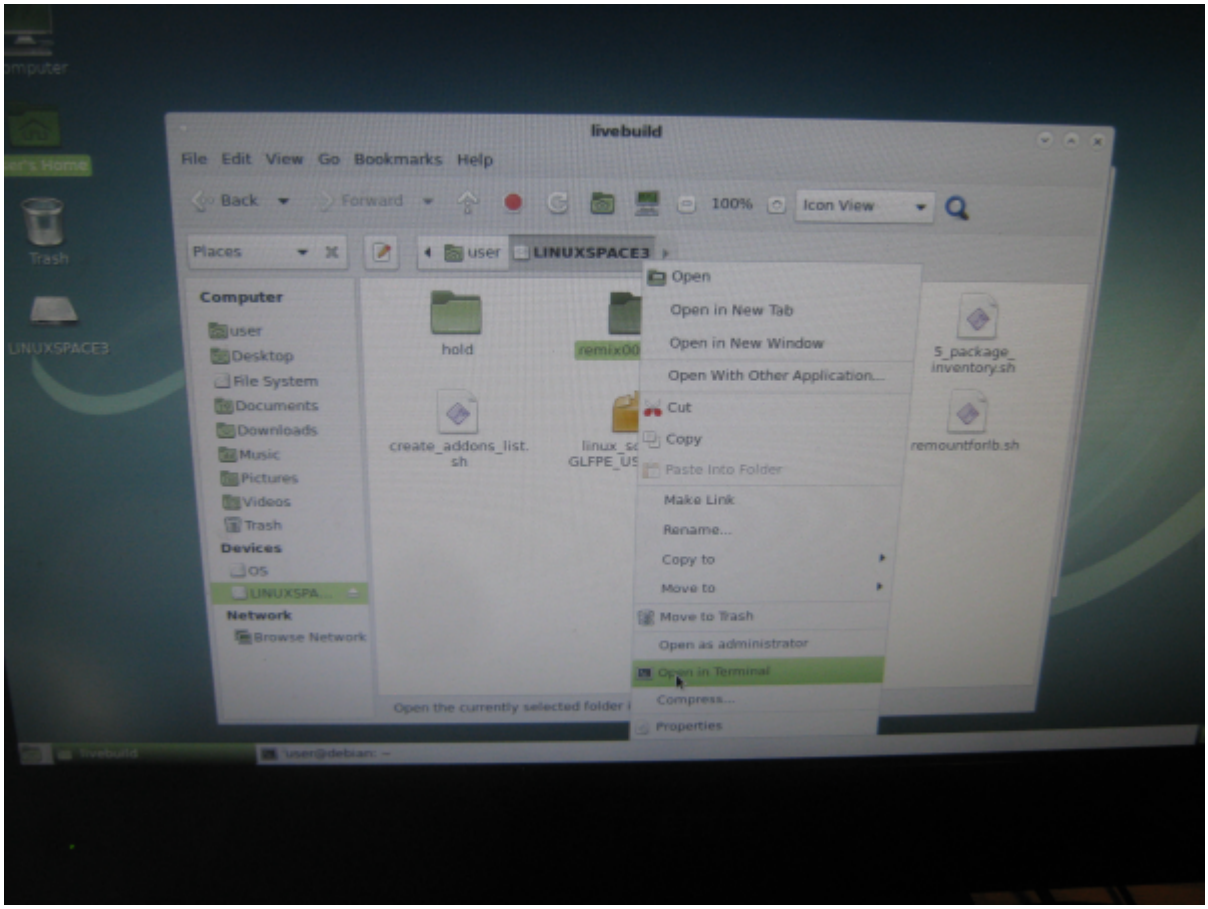
15. Start the build process, then WAIT.

Before starting the build, it would be prudent to turn off any screen blanking or screensaver so you don't need to log in repeatedly to see progress (the default for Debian 9 release is to blank the screen and require login to get back to the desktop).

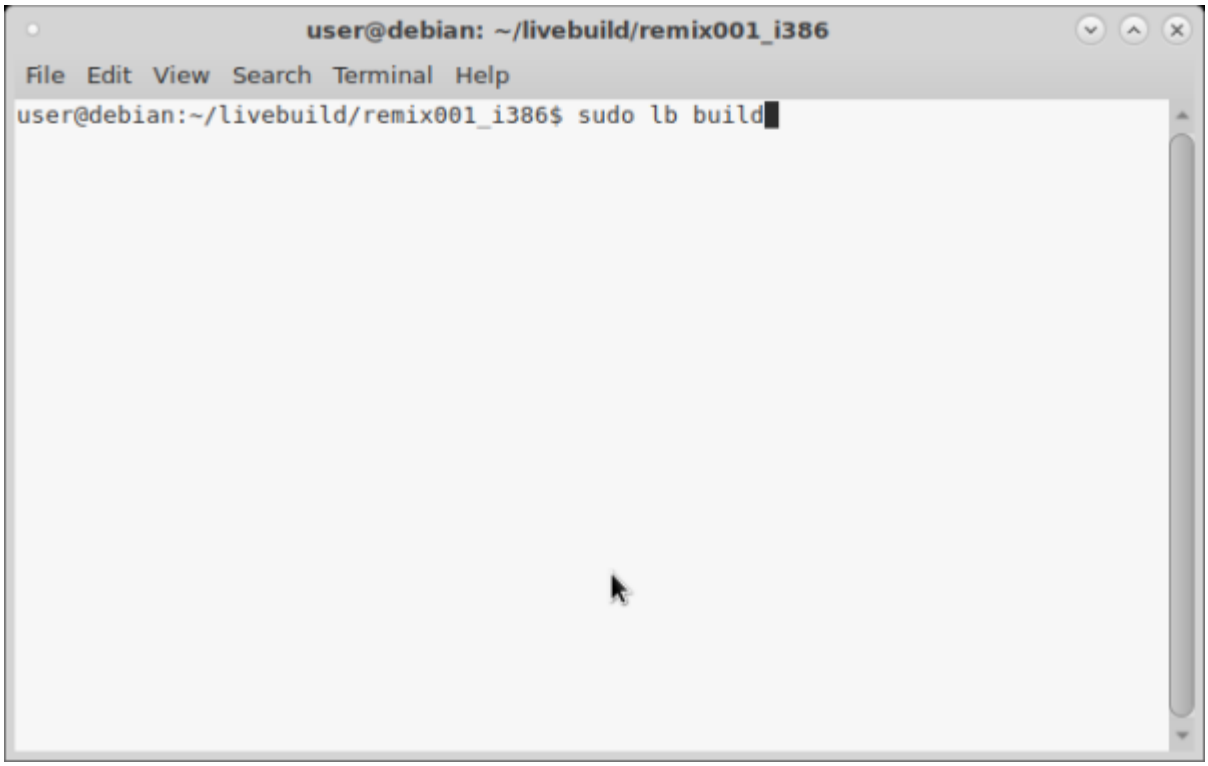


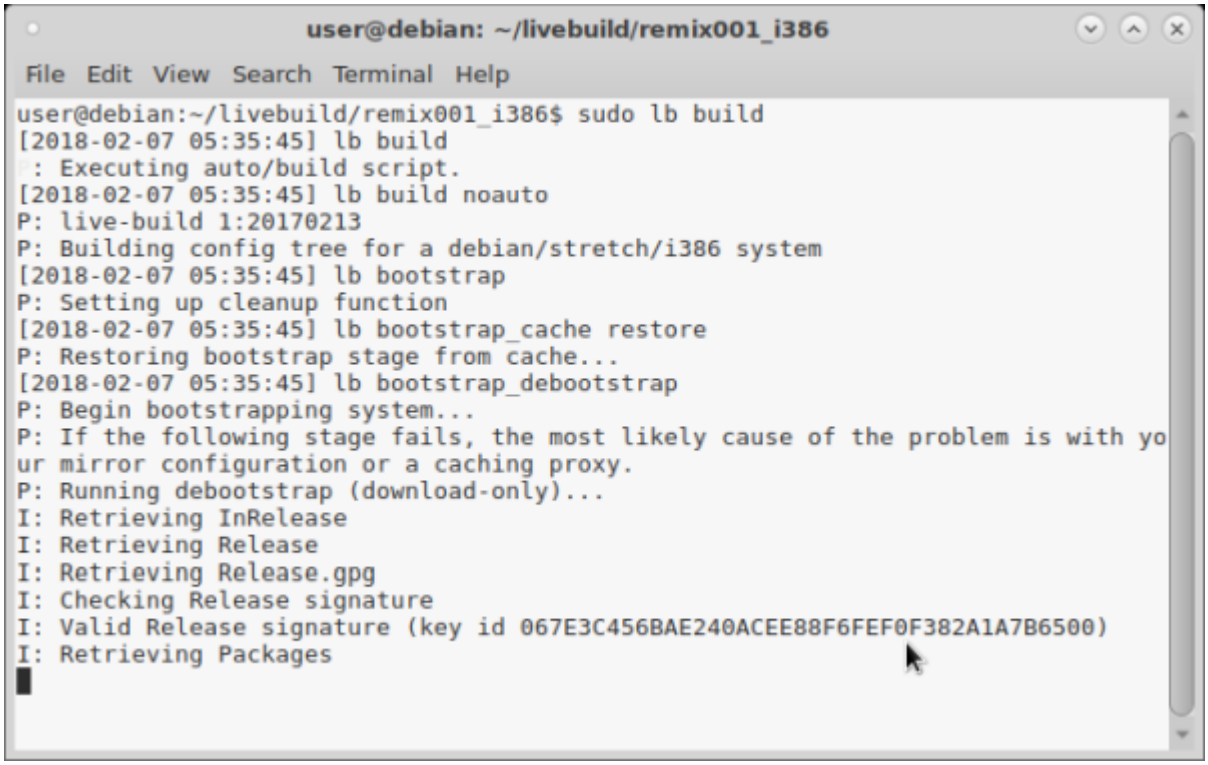


When ready to build, bring up a terminal into the unbuilt configuration folder. The easy way (if you installed "caja-open-terminal") is to right-click on the configuration folder, and select "Open in Terminal".

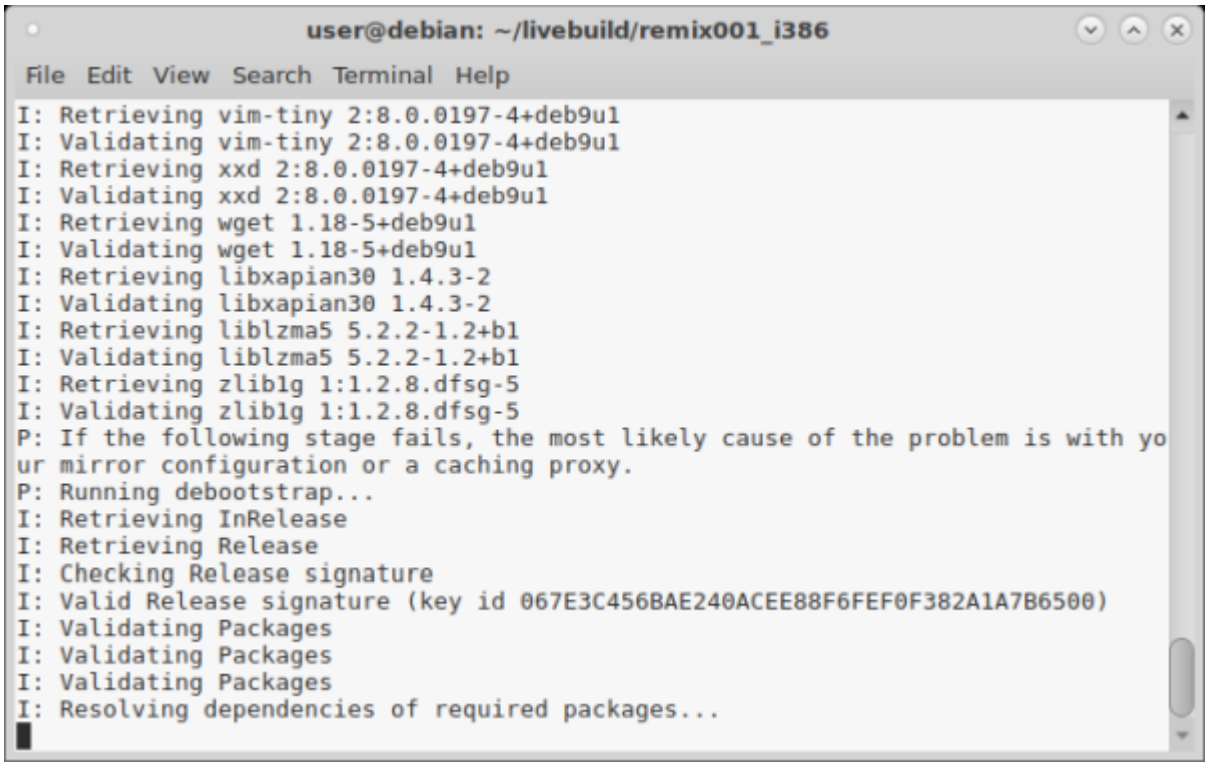


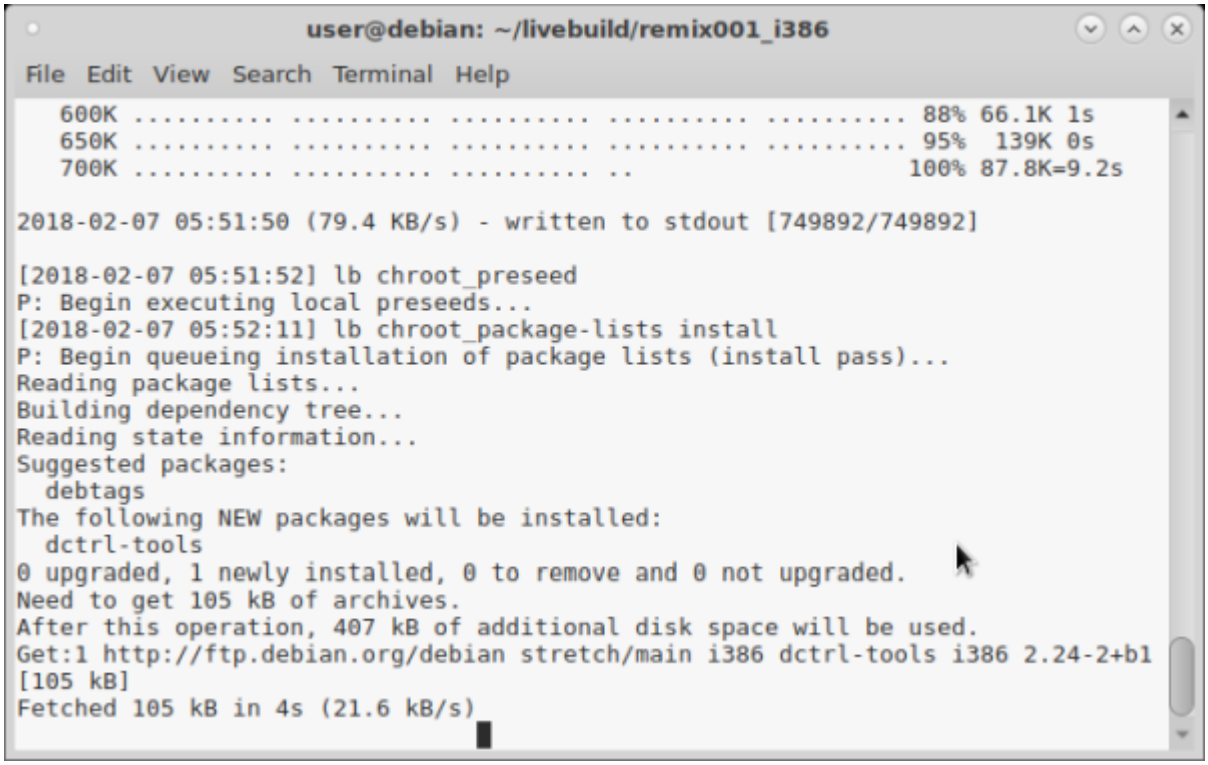
In the terminal, type "sudo lb build" to start the build.

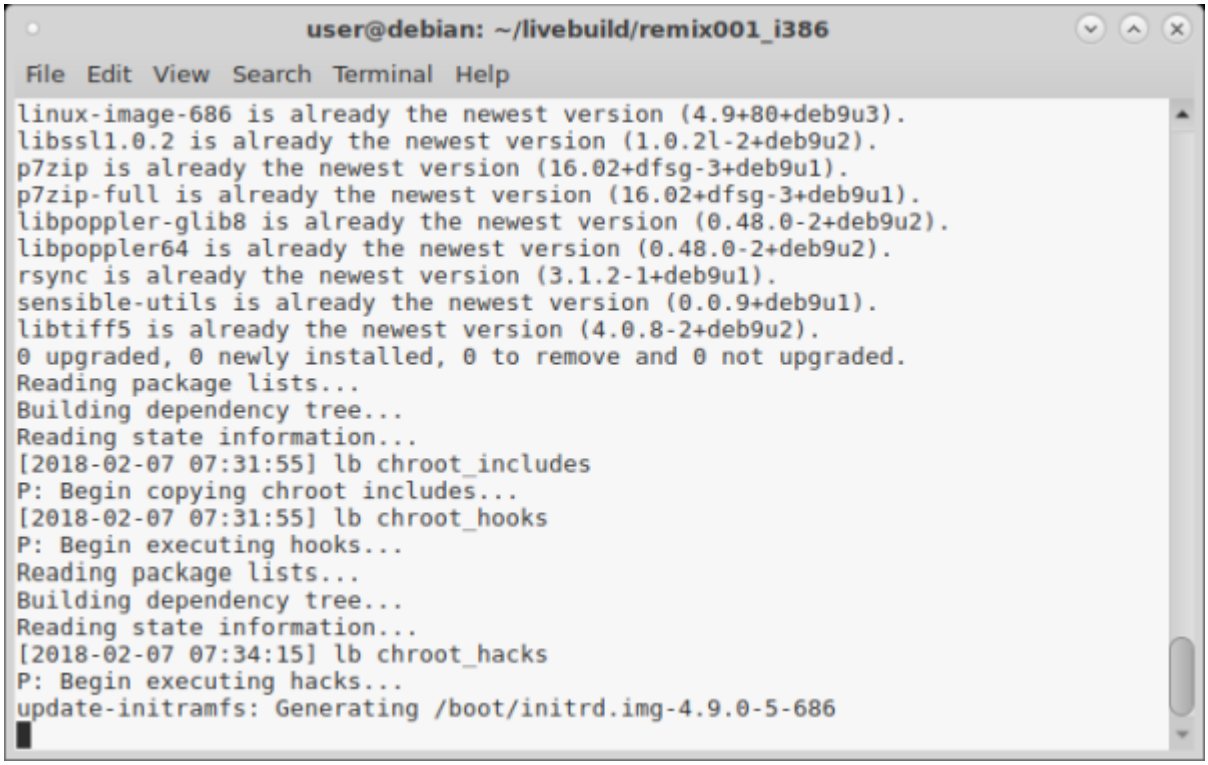


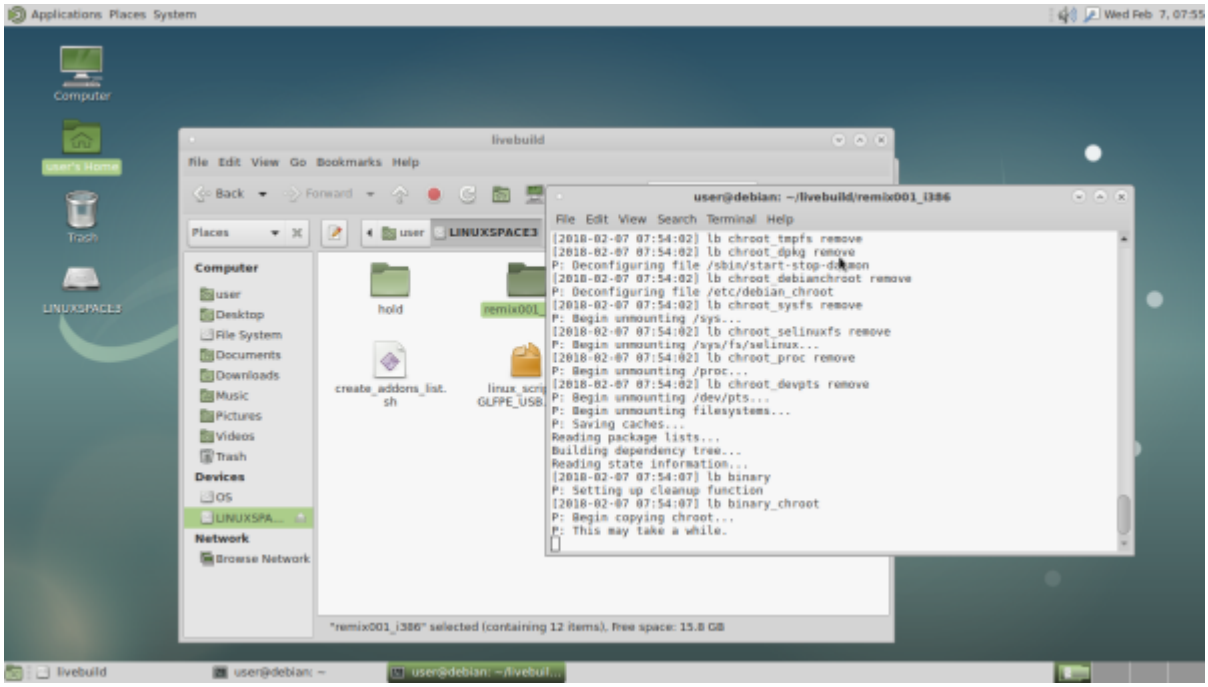


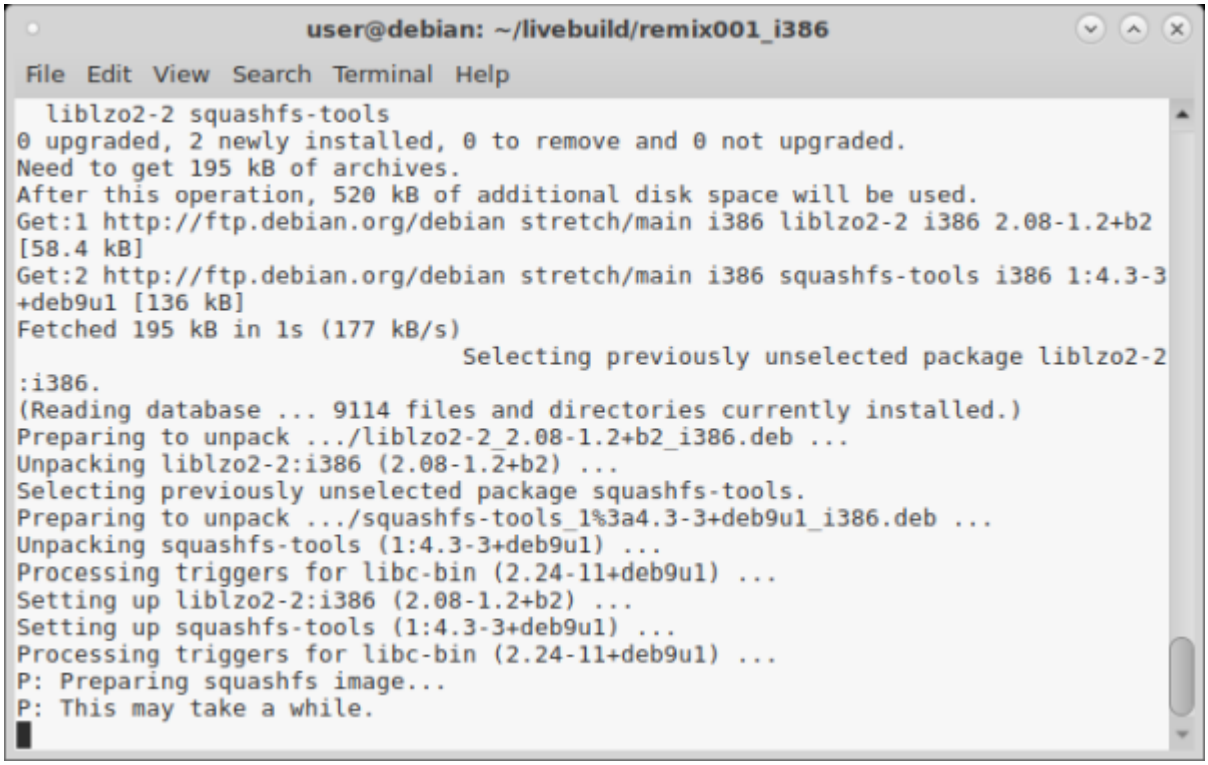
The build will take 2-4 hours, deepending on the speed of the computer and the internet connection. If the build fails from a bad configuration, it will usually fail early. However, any misspelled package name will kill the build, and that could be early, or an hour or more into the run. Assuming it runs to completion, it should consume about 15 Gb of work drive space and create the image file with ".iso" extension under the top level configuration set folder.




user@debian: ~/livebuild/remix001_i386
File Edit View Search Terminal Help
600K 88% 66.1K 1s
650K 95% 139K 0s
700K .... 100% 87.8K=9.2s
2018-02-07 05:51:50 (79.4 KB/s) - written to stdout [749892/749892]
[2018-02-07 05:51:52] lb chroot_preseed
P: Begin executing local preseeds...
[2018-02-07 05:52:11] lb chroot_package-lists install
P: Begin queueing installation of package lists (install pass)...
Reading package lists...
Building dependency tree...
Reading state information...
Suggested packages:
debtags
The following NEW packages will be installed:
dctrl-tools
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 105 kB of archives.
After this operation, 407 kB of additional disk space will be used.
Get:1 http://ftp.debian.org/debian stretch/main i386 dctrl-tools i386 2.24-2+b1
[105 kB]
Fetched 105 kB in 4s (21.6 kB/s)


user@debian: ~/livebuild/remix001_i386
File Edit View Search Terminal Help
linux-image-686 is already the newest version (4.9+80+deb9u3).
libssl1.0.2 is already the newest version (1.0.2l-2+deb9u2).
p7zip is already the newest version (16.02+dfsg-3+deb9u1).
p7zip-full is already the newest version (16.02+dfsg-3+deb9u1).
libpoppler-glib8 is already the newest version (0.48.0-2+deb9u2).
libpoppler64 is already the newest version (0.48.0-2+deb9u2).
rsync is already the newest version (3.1.2-1+deb9u1).
sensible-utils is already the newest version (0.0.9+deb9u1).
libtiff5 is already the newest version (4.0.8-2+deb9u2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Reading package lists...
Building dependency tree...
Reading state information...
[2018-02-07 07:31:55] lb chroot_includes
P: Begin copying chroot includes...
[2018-02-07 07:31:55] lb chroot_hooks
P: Begin executing hooks...
Reading package lists...
Building dependency tree...
Reading state information...
[2018-02-07 07:34:15] lb chroot_hacks
P: Begin executing hacks...
update-initramfs: Generating /boot/initrd.img-4.9.0-5-686





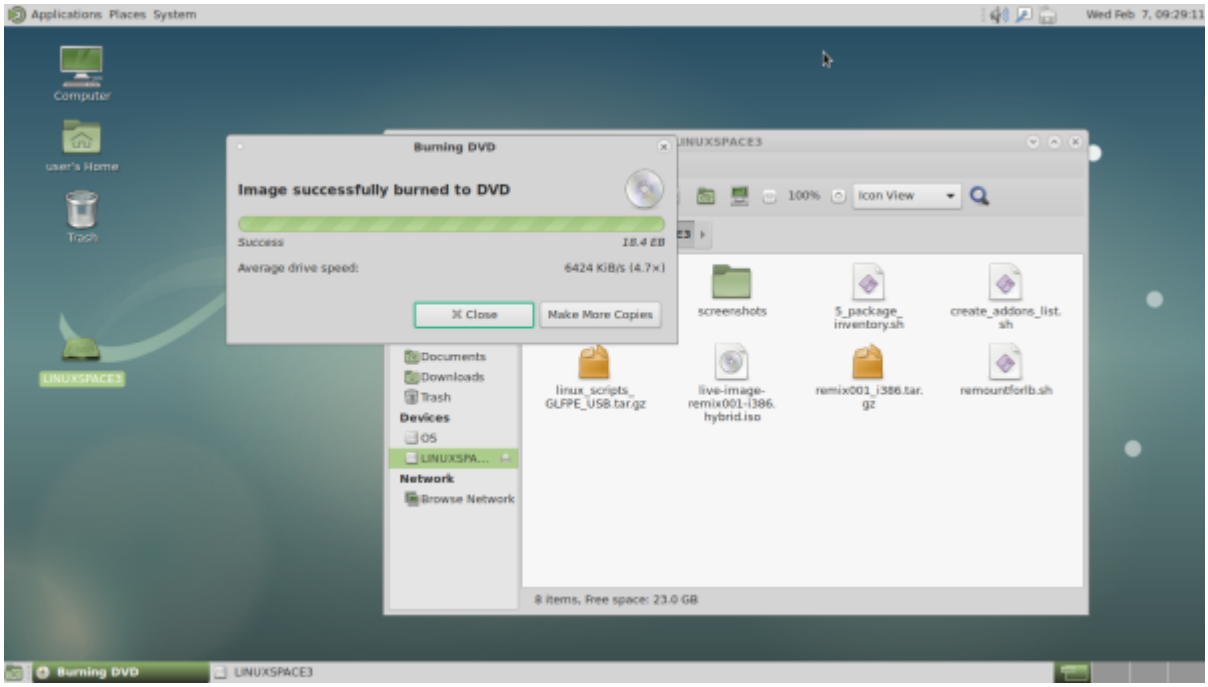
(go back to narrative)

16. *Fix user permissions for the produced ISO file and clean up after the build.*

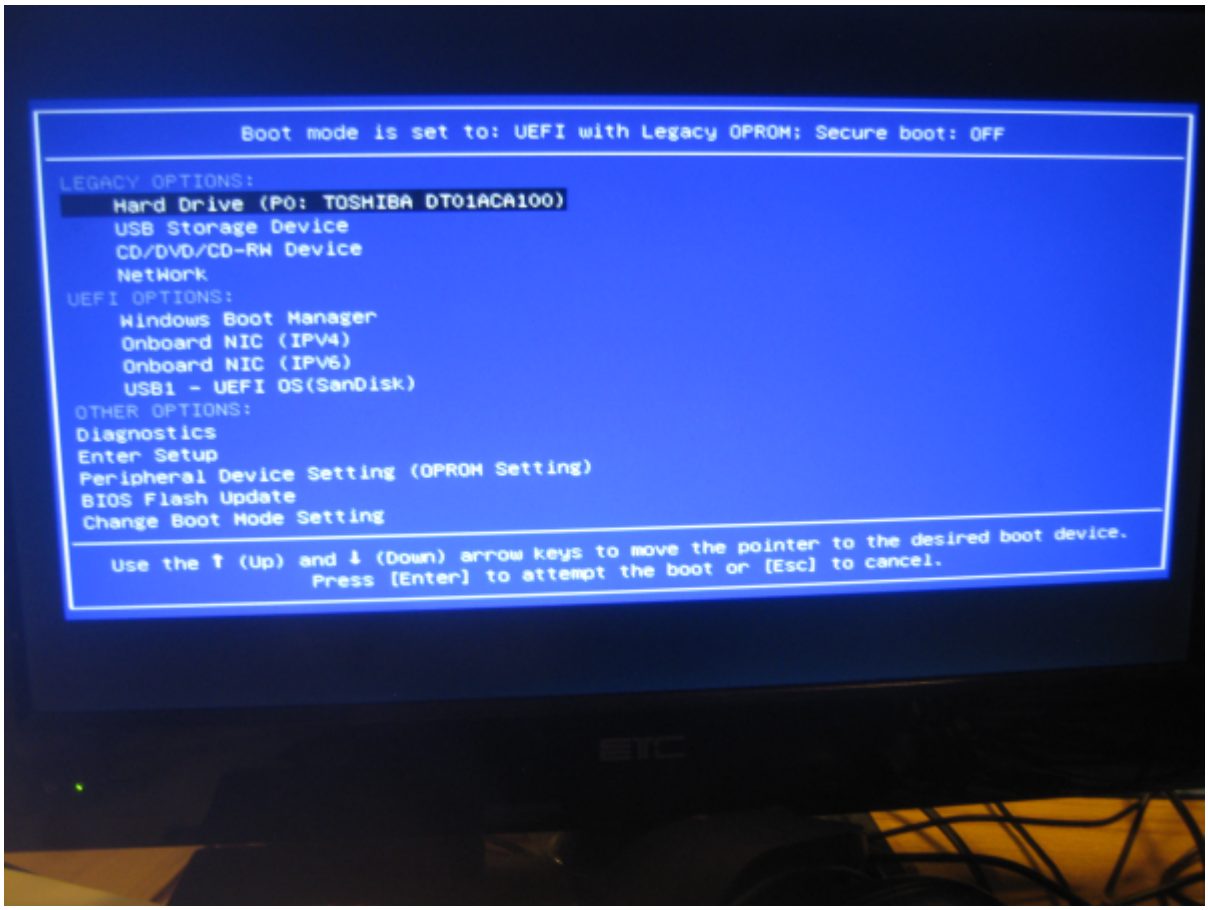
(go back to narrative)

17. *Test the new ISO by burning it to DVD-R+ or USB and booting it.*

Burn the ISO image to DVD or USB. You might need to reboot and try again with a second copy of the original Debian OS on USB to be able to burn a DVD (you can't burn to the same drive you run from). You can also use a Windows utility to burn the ISO to DVD after rebooting Debian. If you boot from DVD-R and you also have an external DVD-RW drive, you can try burning with that.



Shut down. Reboot. Load the new DVD and boot from it to test it. NOTE: the boot menu produced using live-build will look different than the boot menu in the official Debian 9 release, because the official release is built using "live-wrapper", and the exact boot configurations were not replicated here. It will still boot the Linux kernel correctly.







(go back to narrative)

18. Step

(go back to narrative)

Document last edited 9 Feb 2018 GLF