6 Advanced use

6.1 Windows programs under MX Linux

There are a certain number of applications, both open-source and commercial, that will allow Windows applications to run under MX Linux. They are referred to as emulators, meaning that they replicate the functions of Windows on a Linux platform. Many MS Office applications, games and other programs can be run using an emulator with varying degrees of success ranging from near-native speed and functionality to only basic performance.

6.1.1 Open-source

Wine is the primary open-source Windows emulator for MX Linux. It is a kind of compatibility layer for running Windows programs, but does not require Microsoft Windows to run the applications. Installable through MX Package Installer (under Misc); if installing with Synaptic, select "winehq-staging" to get all wine-staging packages. Wine versions are rapidly packaged by the Community Repository members and made available to users, with the latest version coming from the test repo.

NOTE: In order to run Wine when running Live, you need to use home persistence (Section 6.6.3).

- Wine Home Page
- MX/antiX Wiki: Wine

DOSBox creates a DOS-like environment intended for running MS-DOS-based programs, especially computer games.

- DOSBox homepage
- DOSBox Wiki

DOSEMU is software available from the repos that allows DOS to be booted in a virtual machine, making it possible to run Windows 3.1, Word Perfect for DOS, DOOM, etc.

- DOSEMU Home Page
- MX/antiX Wiki: DOSEMU

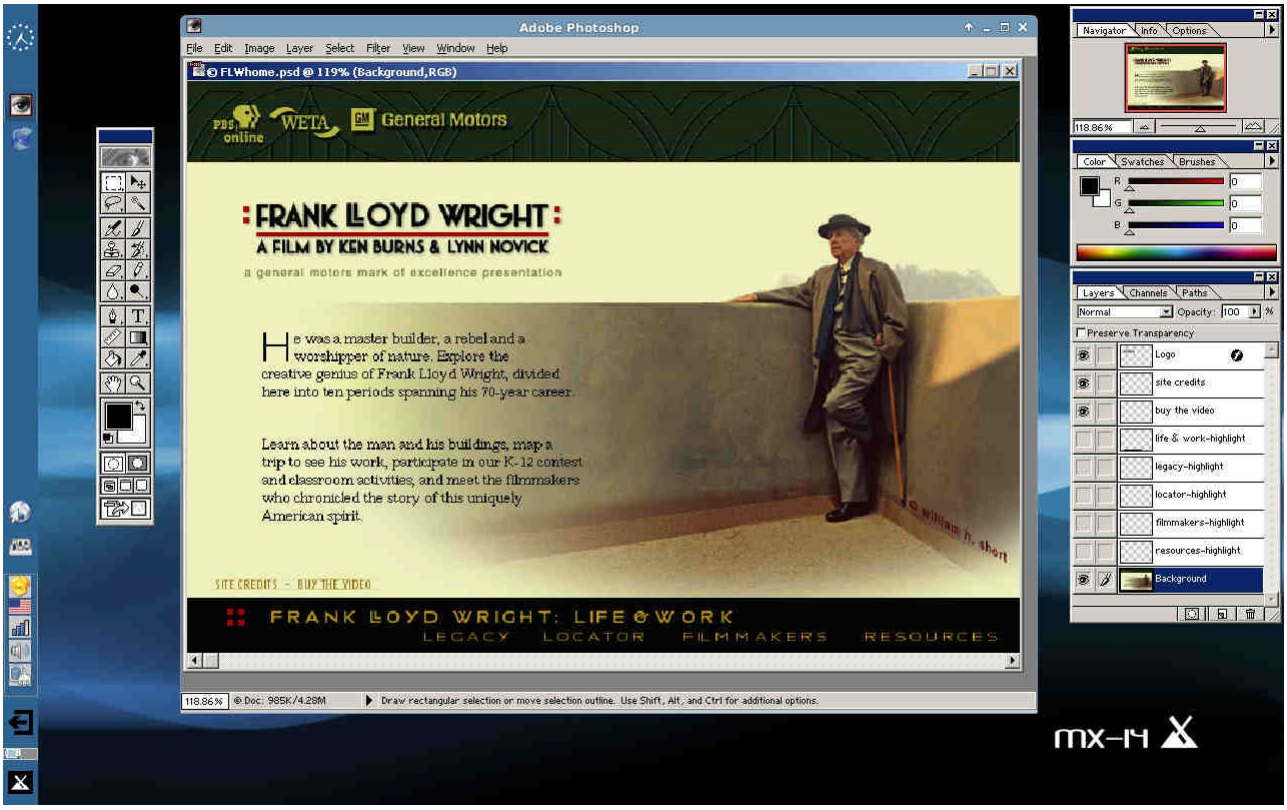

Figure 6-1: Photoshop 5.5 running under Wine

6.1.2 Commercial

CrossOver Office allows you to install many popular Windows productivity applications, plugins and games in Linux, without needing a Microsoft Operating System license. Supports Microsoft Word, Excel and PowerPoint (up to 2003) particularly well.

- CrossOver Linux Home Page
- Wikipedia: Crossover
- Application Compatibility

Links

- Wikipedia: Emulator
- Checklist of games and emulators
- DOS Emulators

6.2 Virtual machines

Virtual machine applications are a class of programs that simulate a virtual computer in memory, allowing you to install any operating system on the machine. It is useful for testing, running non-native applications, and providing users the feeling of having a machine of their own. Many MX Linux users make use of virtual machine software to run Microsoft Windows “in a window” to seamlessly provide access to software written for Windows on their desktop. It is also used for testing to avoid installation.

6.2.1 Setup


Virtual Box: install and configure (14.4)


Virtual Box: set up a shared folder (14.4)

A number of virtual machine software applications for Linux exist, both open-source and proprietary. MX makes it particularly easy to use **VirtualBox**, so we will focus on that here. For details and the most recent developments, see the Links section below. Here is an overview of the basic steps to set up and run VirtualBox:

- **Installation**. This is most easily done via the MX Package Installer (in MX Tools on the Menu). VirtualBox appears in the Misc section. This will enable the VirtualBox repository, download and install the latest version of VirtualBox. The repository will be left enabled, allowing automatic updates via apt-notifier.

- **Post-installation**. Check that your user belongs to the vboxusers group. Open MX User Manager > Group Membership tab. Select your username and make sure that ’vboxusers’ in the Groups list is ticked. Confirm and exit.

- **Extension Pack**. If you install VirtualBox from the repos, the Extension Pack will be included automatically. Otherwise, you should download and install it from the Oracle web site (see Links). After the file is downloaded, navigate to it with Thunar and click on the file’s icon. The Extension Pack will open VirtualBox and install automatically.

- **Location**. Virtual machine files are stored by default in your /home folder. They can be fairly large and if you have a separate data partition you may consider making the default folder there. Go to File>Preferences>General tab and edit the folder location.

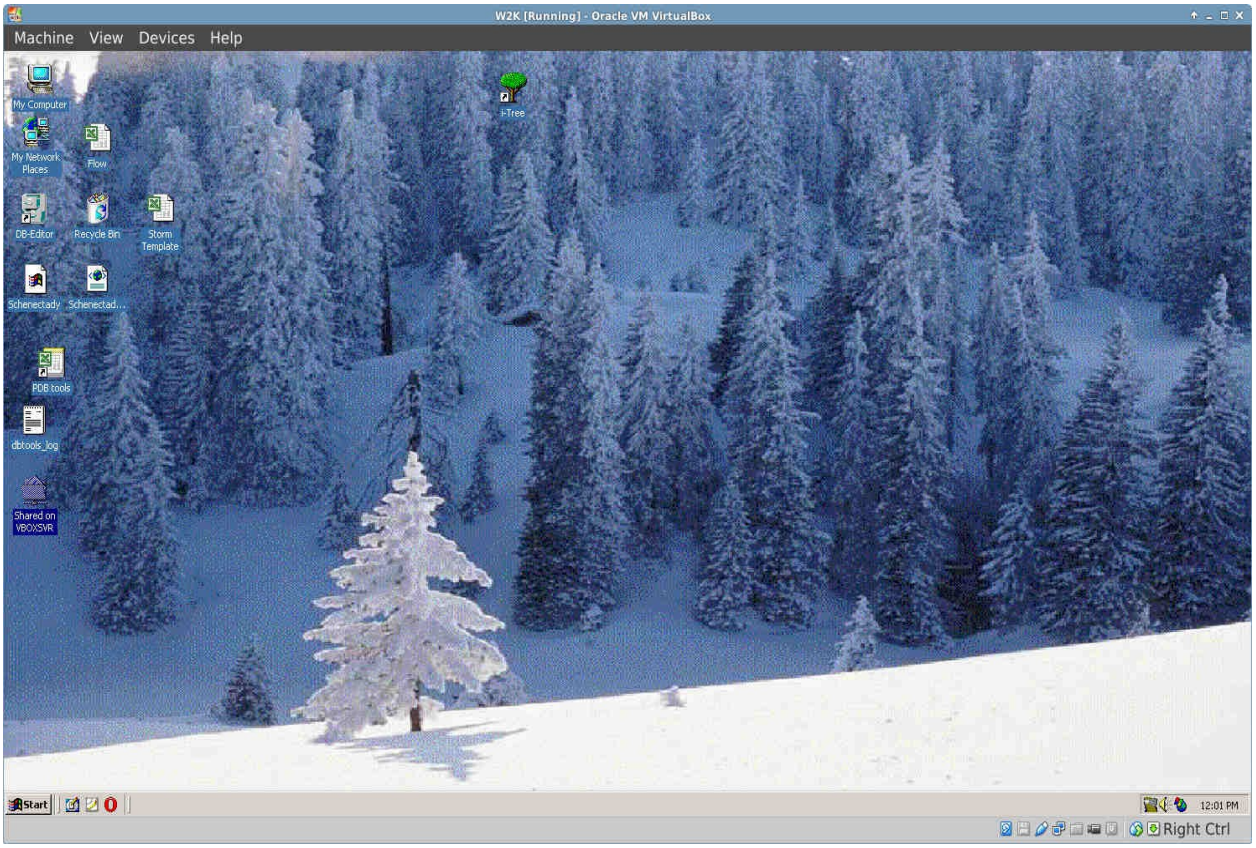

Figure 6-2: Windows 2000 running in VirtualBox

6.2.2 Use

- **Create a Virtual Machine**. To create a virtual machine start VirtualBox, then click the New icon on the toolbar. You will need a Windows CD or a Linux ISO (32bit only). Follow the wizard, accepting all suggested settings unless you know better — you can always change them later. If your ISO has PAE, click on System > Options tab and enable it. You may need to increase the memory allocated to the Guest above the minimum default figure, still leaving sufficient memory for your Host OS. For Windows Guests, consider creating a larger virtual HD than the 10GB default – while it is possible to increase the size later, it is not a straightforward process. Select a Host Drive or Virtual CD/DVD Disk File

- **Select a mount point**. Once the machine is set up, then you can select the mount point to be either the Host Drive or a Virtual CD/DVD Disk File (ISO). Click **Settings > Storage**, and a dialog box will pop up where you will see in the middle a Storage Tree with an IDE Controller and a SATA Controller below it. By clicking on the CD/DVD Drive icon in the Storage Tree, you will see the CD/DVD Drive icon appear in the Attributes section in the right side of the window. Click on the CD/DVD Drive icon in the Attributes section to open a drop-down menu where you can assign the Host Drive or a Virtual CD/DVD disc file (ISO) to be mounted on the CD/DVD Drive. (You can select a different ISO file by clicking on Choose a Virtual CD/DVD

disk file and navigating to the file.) Run the machine. The device you selected (ISO or CD/DVD) will be mounted when you start the virtual machine and your OS can be installed.

- **GuestAdditions**. Once your Guest OS is installed, be sure to install VirtualBox GuestAdditions by booting into the Guest OS, then clicking Devices > Install Guest Additions and pointing toward the VBoxGuestAdditions.iso that it will automatically locate. This will allow you to enable sharing files between Guest and Host and to adjust your display in various ways so that it suits your environment and habits.

- **Moving**. The safest way to move or change the settings of an existing Virtual Machine is to clone it: right-click the name of an existing machine > Clone, and fill in the information. To use the new clone, create a new Virtual Machine and in the wizard when you select the Hard Disk, choose "Use existing hard disk" and select the new clone's *.vdi file.

- **Documentation**. Excellent documentation for VirtualBox is available through Help on the menu bar or as a PDF from the website.

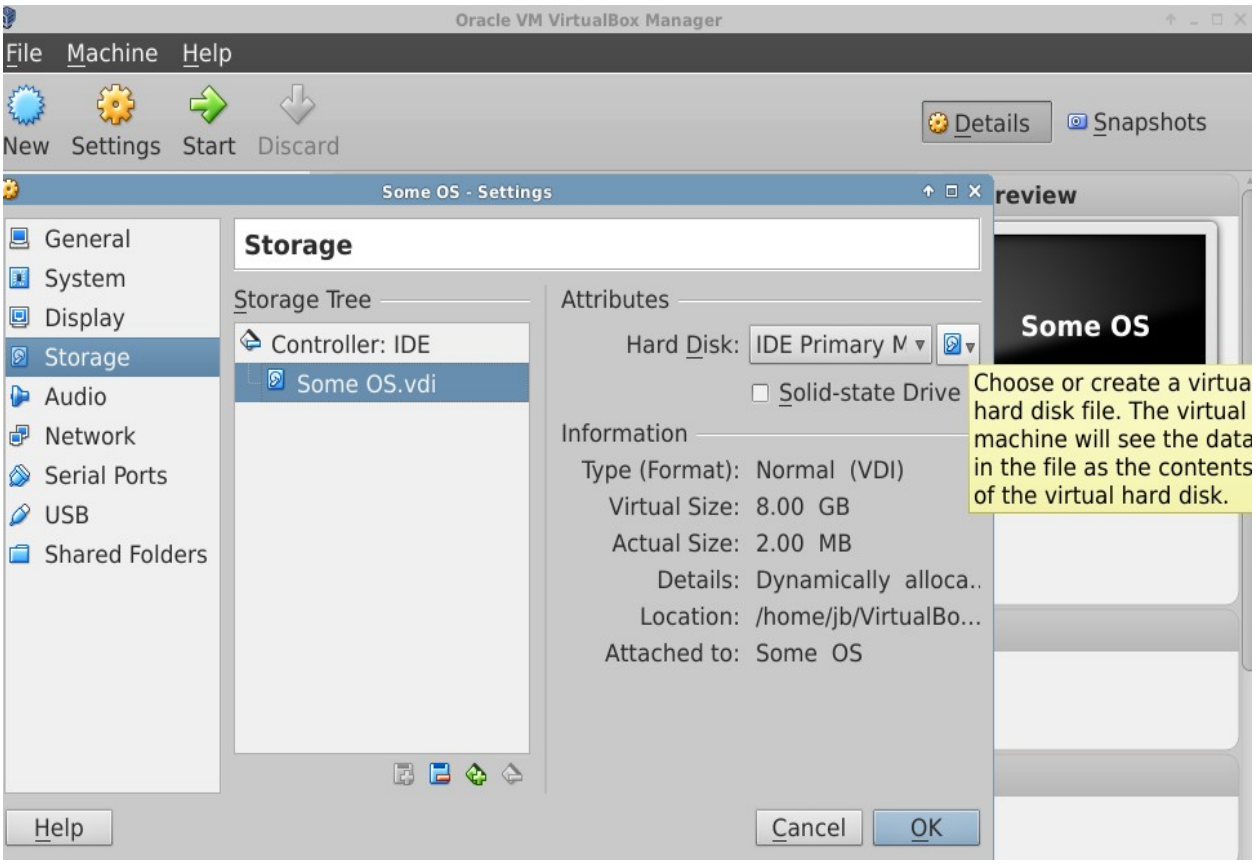

Figure 6-3: Setting the ISO for a new VirtualBox guest "Some OS"

Links

- Wikipedia: Virtual Machine
- Wikipedia: Comparison of virtual machine software
- VirtualBox home page
- VirtualBox Extension Pack

6.3 Alternate Window Managers

A window manager (originally WIMP: Window, Icon, Menu, and Pointing device) in Linux is essentially the component which controls the appearance of Graphical user interfaces and provides the means by which the user can interact with them. MX Linux is tightly tied to Xfce, as part of its overall approach, but other possibilities exist for users. MX Linux makes it easy to install the most popular alternatives through the MX Package Installer, as described below.

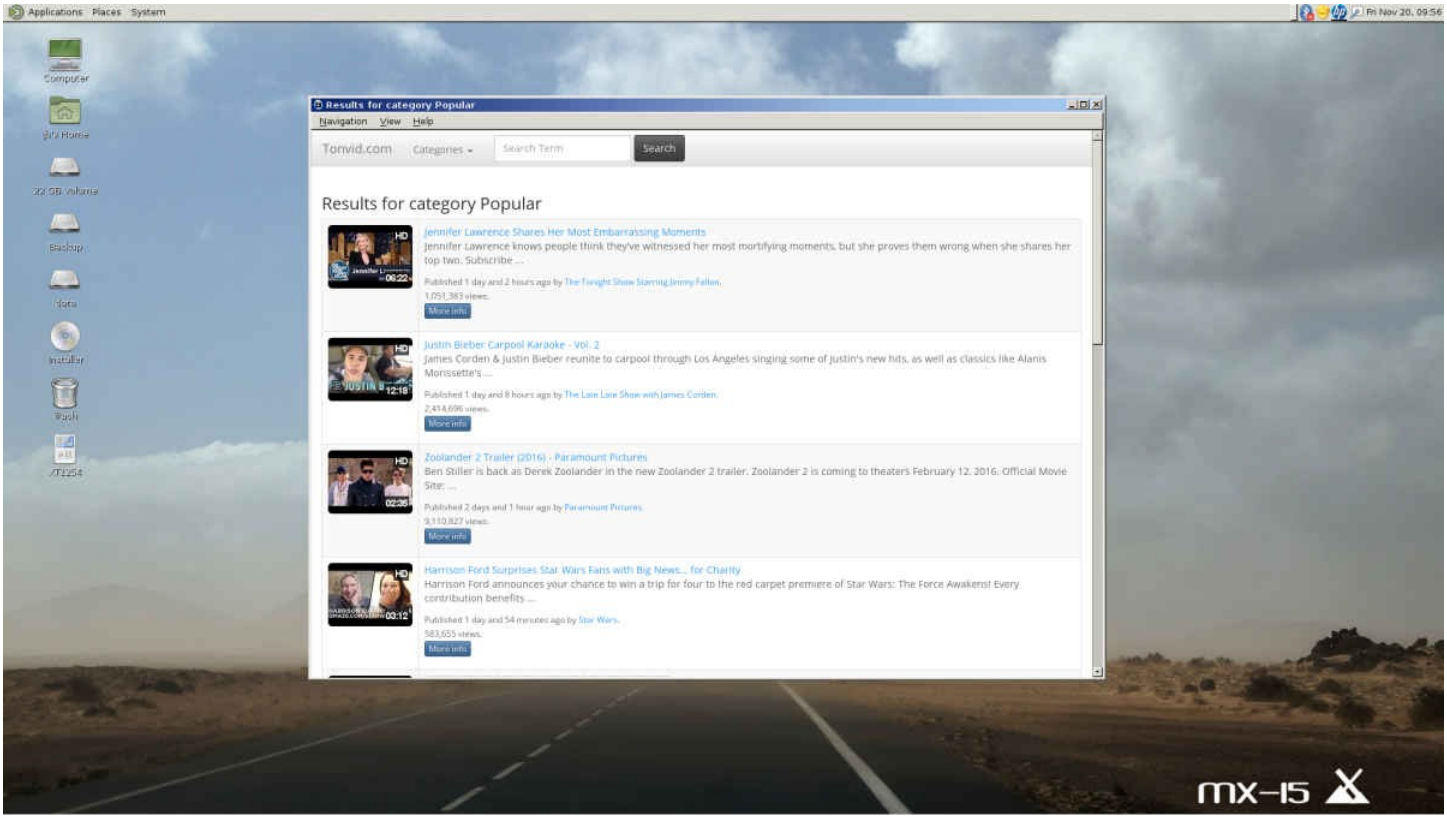

Figure 6-4: MATE running on top of MX Linux, with YouTube Browser open

- Gnome Ultra, a GTK+ based display manager and desktop that provides an ultra-light desktop environment.

    - Gnome Ultra (GOULD), an ultra-light desktop environment

- IceWM, a window manager for the X Window System whose goal is speed and simplicity.
    - IceWM Home Page
    - IceWM FAQ and Howto
- K Desktop Environment, or KDE for short, a very large and powerful environment. Two versions available: Lite and Standard; Lite offers more application choices. For installation and configuration when used over MX Linux, see the MX/antiX Wiki.
    - KDE Home Page
    - KDE Forum

 MX-16 with Kwin

- LXDE is a fast and light desktop environment whose components can be installed separately.
    - LXDE home page
    - LXDE Wiki
- MATE is the continuation of GNOME 2 providing an intuitive and attractive desktop environment.
    - MATE home page
    - MATE Documentation

Once installed, you can choose the window manager you want from the Session Button on the default login screen and log in to as you normally would. If you replace the login manager with another from the repos, make sure you always have at least one available upon reboot.

MORE: Wikipedia: X Window Managers

6.4 Command Line

Although MX offers a complete set of graphical tools for installing, configuring, and using your system, the command line (also called the console, terminal, BASH, or shell) is still a useful and at times indispensable tool. Here are some common uses:

- Launch a GUI application to see its error output.
- Speed up system administration tasks.
- Configure or install advanced software applications.
- Execute multiple tasks quickly and easily.
- Troubleshoot hardware devices.

The default program to run a terminal in an MX desktop window is **XFCE Terminal**, which can be found at **Start Menu > System > Xfce Terminal (Terminal Emulator)**. Some commands are only recognized for super user (root), while others may vary the output depending on user.

To obtain temporary root permissions:

1. open Xfce Terminal.
2. type **su**.
3. enter root's password (nothing will show on the screen)

You will recognize when Xfce Terminal is running with root privileges by looking at the prompt line right before the space where you type. Instead of a **$**, you will see a **#**; in addition, the user name changes to **root** written in red. If you try to run as a regular user a command that requires root privileges such as **iwconfig**, you may receive an error message that the command was not found, see a message box that the program must be run as root, or simply find yourself at the prompt again with no message at all.

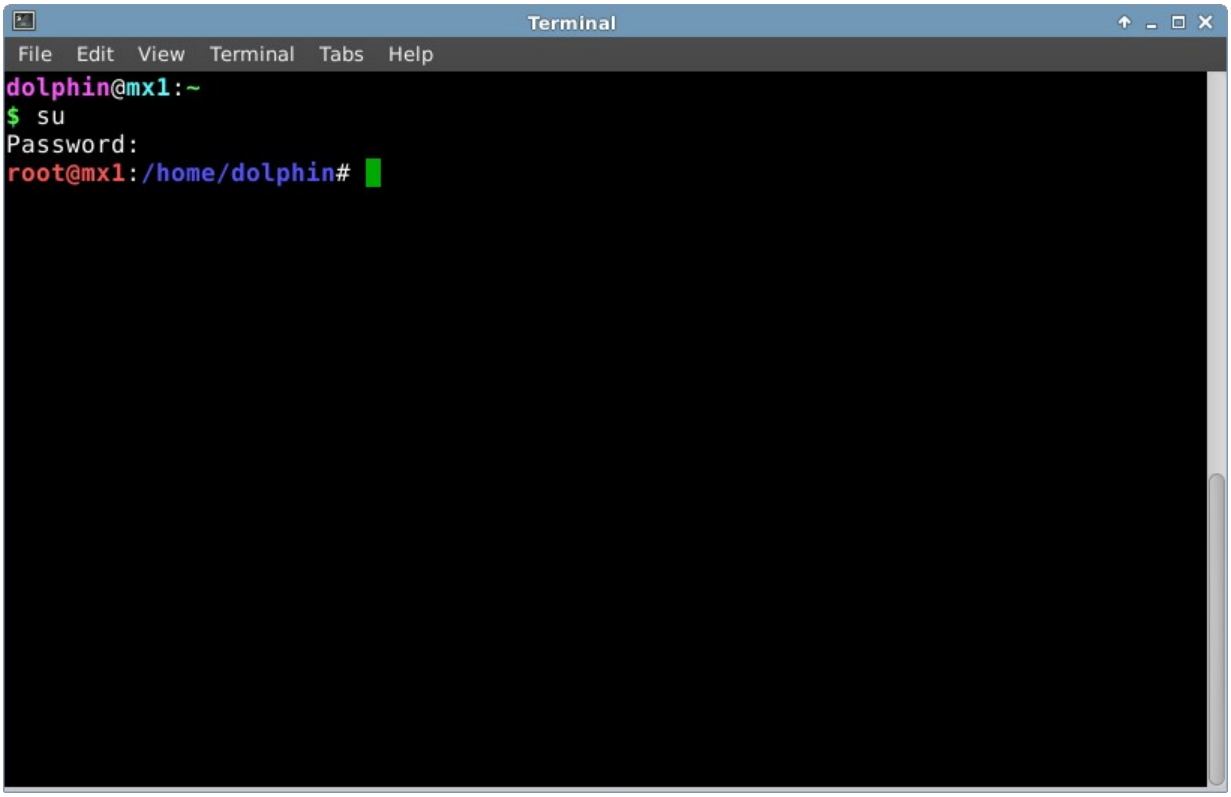

Figure 6-5: User now has temporary administrative (root) privileges

6.4.1 First steps

- For more information on running Xfce Terminal for solving system problems, please refer to the topic **Troubleshooting** at the end of this section. Also, it is advisable to make backups of the files you are working on as a root user with the commands **cp** and **mv** (see below).

- Though terminal commands can be fairly complex, understanding the command line is just a matter of putting together simple things. To see how easy it can be, open Xfce Terminal and try a few basic commands. This will all make more sense if you do it as a tutorial exercise rather than just reading it. Let's start with a simple command: **ls**, which lists the contents of a directory. The basic command lists the contents of whatever directory you are currently in:

  ```
  ls
  ```

- That's a useful command, but it's just a few short columns of names printed across the screen. Suppose we want more information on the files in this directory. We can add a **switch** to the command to make it print out more information. A **switch** is a modifier we append to a command to change its behavior. In this case, the switch we want is:

  ```
  ls -l
  ```

- As you can see on your own screen if you are following along, this switch provides more detailed information on the files in any directory.

- Of course, we might want to see the contents of another directory (without going there first). To do this, we add an **argument** to the command, specifying which file we want to look at. An **argument** is a value or reference we add to a command to target its operation. By giving an argument of **/usr/bin**/, we can list the contents of that directory, rather than the one where we currently are.

  ```
  ls -l /usr/bin
  ```

- There are a lot of files in **/usr/bin**/! It would be nice if we could filter this output so that only entries that contained, say, the word "**fire**"would be listed. We can do this by **piping** the output of the **/ls/** command into another command, **grep**. The **pipe**, or │ character, is used to send the output of one command to the input of another. The command **grep** searches for the pattern you give it and returns all matches, so piping the output of the previous command to it filters the output.

  ```
  ls -l /usr/bin | grep fire
  ```

- Finally, suppose we want these results saved in a text file for use at a later time. When we issue commands, the output is usually directed to the console display; but we can redirect this output somewhere else, such as to a file, using the > (redirect) symbol to instruct your computer to make a detailed list of all the files that contain the word "**fire**" in a particular directory (by default your Home directory, and to create a text file containing that list, in this case named "**FilesOfFire**"

  ```
  ls -l /usr/bin | grep fire > FilesOfFire.txt
  ```

- As you can see, the command line can be used to perform complex tasks very easily by combining simple commands in different ways.

6.4.2 Common commands

Here is a list of rudimentary terminal commands. For a complete reference, see the Links section, below.

Filesystem navigation

Table 6: Filesystem navigation commands

Command	*Comment*
cd /usr/share	Changes current directory to the given path: "/usr/share". With no argument, **cd** takes you to your home directory.
pwd	Prints the current working directory path
ls	Lists the contents of the current directory. Use the **-a** switch to show hidden files as well, and the **-l** switch to show details on all files. Often combined with other terms. **lsusb** lists all the usb devices, **lsmod** all the modules, etc.

File management

Table 7: File management commands

Command	*Comment*
cp sourcefile destinationfile	Copy a file to another filename or location. Use the **-R** switch ("**recursive**") to copy entire directories.
mv sourcefile destinationfile	Move a file or directory from one location to another. Also used to rename files or directories and to make a backup: for example before changing a critical file such as **xorg.conf** you might use this command to move it to something like **xorg.conf_bak**.
rm filename	Delete a file. Use the **-R** switch to delete a directory, and the **-f** switch ("**force**") if you don't want to be prompted to confirm each deletion.
cat file.txt	Prints the contents of a file on the screen. Only use on text files.
grep	Find a given string of characters in a given piece of text, and print the entire line it was on. Usually used with a pipe, e.g. cat **somefile.txt I grep /somestring/** will display the line from somefile.txt that contains **somestring**. To find a network usb card, for instance, you could type: **lsusb I grep Network**. The grep command is case sensitive by default, use the **-i** switch to make it case-insensitive.
dd	Copies anything bit by bit, so can be used for directories, partitions, and whole drives. Basic syntax is **dd if=<some file> of=<some other file>**

Symbols

Table 8: Symbols

Command	*Comment*
|	The pipe symbol used to send the output of one command to the input of another. Some keyboards show two short vertical bars instead of on
>	The redirect symbol, used to send the output of a command into a file of device. Doubling the redirect symbol will cause the output of a command to be added to an existing file rather than replacing it.
&	Adding the ampersand to the end of a command (with a space before it) causes it to run in the background so that you don't have to wait for it to complete to issue the next command. Double ampersand indicates that the second command should only be run if the first has been successful.

Troubleshooting

For most new Linux users, the command line is mainly used as a troubleshooting tool. Terminal commands give quick, detailed information that can be easily pasted into a forum post, search box, or email when seeking help on the web. It is strongly recommended that you keep this information at hand when asking for help. Being able to refer to your specific hardware configuration will not only speed up your process of obtaining help, but also it will let others offer you more accurate solutions. Here are some common troubleshooting commands (see also Section 3.4.4). Some of them may not output information, or not as much information unless you are logged in as root.

Table 9: Troubleshooting commands

Command	*Comment*
lspci	Shows a quick summary of detected internal hardware devices. If a device shows as /unknown/, you usually have a driver issue. The **-v** switch causes more detailed information to be displayed.
lsusb	Lists attached usb devices.
dmesg	Shows the system log for the current session (i.e. since you last booted). The output is quite long, and usually this is piped through **grep**, **less** (similar to **most**) or **tail** (to see what happened most recently). For example, to find potential errors related to your network hardware, try **dmesg I grep -i net**.
top	Provides a real-time list of running processes and various statistics about them. Also available from the Start menu as **Htop** along with a nice graphical version **Task Manager**.

Accessing documentation for commands

- Many commands will print out a simple “usage information” message when you use the “ \SpecialChar nobreakdash\SpecialChar nobreakdashhelp” or “-h” switch. This can be helpful for quickly recalling the syntax of a command. For example: *cp \SpecialChar nobreakdash\SpecialChar nobreakdashhelp*

- For more detailed information on how to use a command, consult the command’s man page

- Manual pages are conveniently accessed through the Start menu by placing a hash mark and the command name into the search box at the top. For example, you can view the man page for the copy command with this entry in the search box: *#cp*

- The man page may also be read at the console with the command **man** <**commandname**>. By default, man pages are displayed in the terminal’s **most** pager, meaning that only one screenful of the file is displayed at a time. Keep these tricks in mind to navigate the resulting screen:

- The space bar (or PageDown key) advances the screen.
- The letter **b** (or PageUp key) moves the screen backward.
- The letter **q** exits the help document.

Alias

You can create an **alias** (personal command name) for any command, short or long, that you want. Details in the MX/antiX Wiki.

6.4.3 Links

- BASH Beginners Guide
- Command Line Basics

6.5 Scripts

A script is a simple text file that can be written directly from a keyboard, and consists of a logically sequenced series of operating system commands. The commands are handled one at a time by a command interpreter which in turn requests services from the operating system. The default command interpreter in MX is **Bash**. The commands must be understandable to Bash, and command lists have been established for programming use. A shell script is the Linux counterpart of batch programs in the Windows world.

Scripts are used throughout the Linux OS and applications that run on it as an economical method of executing multiple commands in an easily created and modified manner. During boot, for instance, many scripts are invoked to start up specific processes such as printing, networking, etc. Scripts are also used for automated processes, system administration, application extensions, user controls, etc. Finally, users of all kinds can employ scripts for their own purposes.

6.5.1 A simple script

Let’s do a very simple (and famous) script to get the basic idea.

1. Open the text editor Leafpad (**Start Menu > Accessories**), and type:

```
#!/bin/bash
clear
echo Good morning, world!
```

2. Save that file in your home directory with the name **SimpleScript.sh**

3. Right-click the file name, select Properties, and check “Allow this file to run as a program” on the Permissions tab.

4. Open a terminal and type:

   ```
   sh /home/<username>/SimpleScript.sh
   ```

5. The line “Good morning, world!” will appear on your screen. This simple script doesn’t do very much, but it does establish the principle that a simple text file can be used to send commands to control your system’s behavior.

NOTE: All scripts open with a **shebang** as in the the beginning of the first line: it is a combination of a hash sign (#), an exclamation point, and the path to the command interpreter. Here, Bash is the interpreter and it is found in the standard location for user applications.

6.5.2 A useful script

Let’s look at a useful script for the ordinary user that reduces all the moves involved in backing up multiple sets of files into a single keystroke. The script below relies itself on a system script called **Rdiff-backup** that would need to be installed from the repos for the script to work. It copies one directory to another, keeping a record of the differences in a special subdirectory so you can still recover files lost some time ago. (Incidentally, Rdiff-backup relies in turn on a script called **Diff**.)

In this example, a user named “newbie” wants to set up a script to back up documents, music, mail and pictures from the /home directory to an external drive.

```
 1 #!/bin/bash
 2 #
 3 # This Rdiff-Backup script backs up to a second hard drive
 4 # It must be run as root in order to mount the second hard drive
 5
 6 # To restore files, issue the command: cp -a /mnt/sda1/username /home
 7 # To restore, but not overwrite:
 8 # cp -a -i \SpecialChar nobreakdash\SpecialChar nobreakdash”reply=no
/mnt/sda1/username /home
 9
 10 # Mount the external devices
 11
 12 mount /dev/sdb1
 13 mount /dev/sdb2
 14 mount /dev/sdb3
 15
 16 # Execute the backup
 17
 18 rdiff-backup /home/newbie/Documents /mnt/sdb2/Documents
 19 rdiff-backup /home/newbie/Music /mnt/sdb1/Music
 20 rdiff-backup /home/newbie/Mail /mnt/sdb2/Mail
 21 rdiff-backup /home/newbie/Pictures /mnt/sdb3/Pictures
 22
 23 # Unmount the external devices
 24
```

```
25 umount /dev/sdb1
26 umount /dev/sdb2
27 umount /dev/sdb3
```

Now let's look at this script's components:

- Lines 2-8: a number sign has been placed in front of these lines (called "commenting them out") to indicate to Bash that they are not part of the sequence of commands to be executed. Their purpose here is to provide anyone who looks at this script with information about such things as the script's origin, creator, purpose, and license (metadata).

- Line 10: good scripts separate the commands into clearly labeled procedural sections, also in lines 16 and 22.

- Lines 12-14: the three devices to be used for the backup have to first be mounted so they are available to the system.

- Lines 18-21: here bash is told to use the system script rdiff-backup to compare the original directories (sources) with the backup directories (targets), copy over the differences it finds, and keep a record of the changes.

- Lines 25-27: once the backup work is done, the external drives are unmounted from the system.

Anyone who wanted to use such a script would have to carry out a few execution steps:

1. Copy the whole script.

2. Right-click the desktop and select **Create New > Text file**...

3. Give the file a name that makes sense (no spaces, though), and add the "sh" extension so you will recognize it is a script. For this example, you might select **Backup_DocsMusicMailPictures.sh**

4. Open the new text file and paste in the script.

5. Change any names, locations, etc. to what they are on your particular system. In the example above, you may well have different names and/or locations for the directories to be backed up, and different devices where they are supposed to go.

6. Save that script in a place you can easily find it when you need it, let's say you make a new directory **/home/scripts** for it.

7. Right-click the script, select Properties, click on the Permissions tab, and check the **Is executable** box and click OK.

8. When you are ready to backup, open a terminal and type:

   ```
   sh /home/scripts/Backup_DocsMusicMailPictures.sh
   ```

   HINT: use the tab key to autocomplete the file name after you type the first few letters.

Links

- Linux Shell Scripting Tutorial
- Directory of Linux Commands

6.5.3 Pre-installed user scripts

The following scripts allow users to help keep their MX Linux installation up-to-date and running as a rolling release.

smxi

When run, smxi allows users to install a new kernel, install ATI and Nvidia graphics drivers, run apt-get upgrade or apt-get dist-upgrade safely, and lots more! Written by a programmer known as "h2", the script is pretty much self explanatory, but for usage options, execute *smxi -h*.

smxi must be run outside of the X window system (i.e., not from the desktop) for most functions.

- From your desktop:
  - Press Ctrl-Alt-F1 to get to a terminal prompt.
  - Log in as root (su and password)
  - Run command:

    ```
    smxi
    ```
- At boot:
  - Type e when on the GRUB menu to be able to edit it
  - Add "3" to the end of the kernel line in GRUB's menu entry for MX Linux, just after the word "quiet"

- The system will boot directly to the console.
- Log in as “root” (without quotation marks) and supply the appropriate password
- Run command:

```
smxi
```

smxi will ask a series of questions the first time it is run, including which system options you want to run. The following options are recommended:

- continue-no-changes
- apt-get
- apt-get dist-upgrade

After smxi has completed its operations it will ask if you want to restart the desktop. **NOTE**: Running smxi -G in a root terminal while inside a running X session allows certain features of smxi to run, such as removing unwanted kernels etc.

smxi home page

smxi documentation

sgfxi

This h-2 script runs inside smxi or separately, and deals with installing graphical drivers. Sgfxi currently supports ATI, fglrx and Nvidia drivers. It also supports converting from or to xorg free drivers like ati, intel, or nv. Follow the procedure above to start the script, replacing smxi with sgfxi.

Sgfxi requires a working internet connection! Some wireless internet connections may be dropped when operating outside of X. If this applies to your internet connection either temporarily switch over to a wired internet connection before proceeding or use the ’Partial install in X, completion outside of X method’ instructions in the next section below.

The sgfxi script automatically downloads and installs the kernel headers and everything else it needs. Then it downloads the binary graphic driver installers from either Nvidia or ATI, prepares system, installs, then sets up **xorg.conf**, all in a fairly clean, reasonably intuitive way. Plus it updates itself so any new drivers released will be installed. Finally, sgfxi allows you to easily switch between proprietary non free drivers like ATI’s fglrx and Nvidia’s nvidia driver and the free xorg drivers.

NOTE: running sgfxi on an nvidia chipset system will remove **ddm-mx**, the software package used by the MX Tools installers, so you need to reinstall it if you wish to use it.

sgfxi manual

inxi

A third script from h-2 included in MX Linux is inxi, a convenient command-line system information script. Enter *inxi -h* in a terminal to see all the options available, which include an entire range from sensor output to the weather.

MORE: MX/antiX Wiki

6.6 Advanced MX Tools

In addition to the configuration MX Apps discussed in Section 3.2, MX Linux includes utilities for the advanced user available from MX Tools.

6.6.1 Live-usb kernel updater (CLI)

WARNING: for use in a Live session only!

This command line application can update the kernel on an MX LiveUSB with any kernel that has been installed. This application will only show in MX Tools when running a Live session.

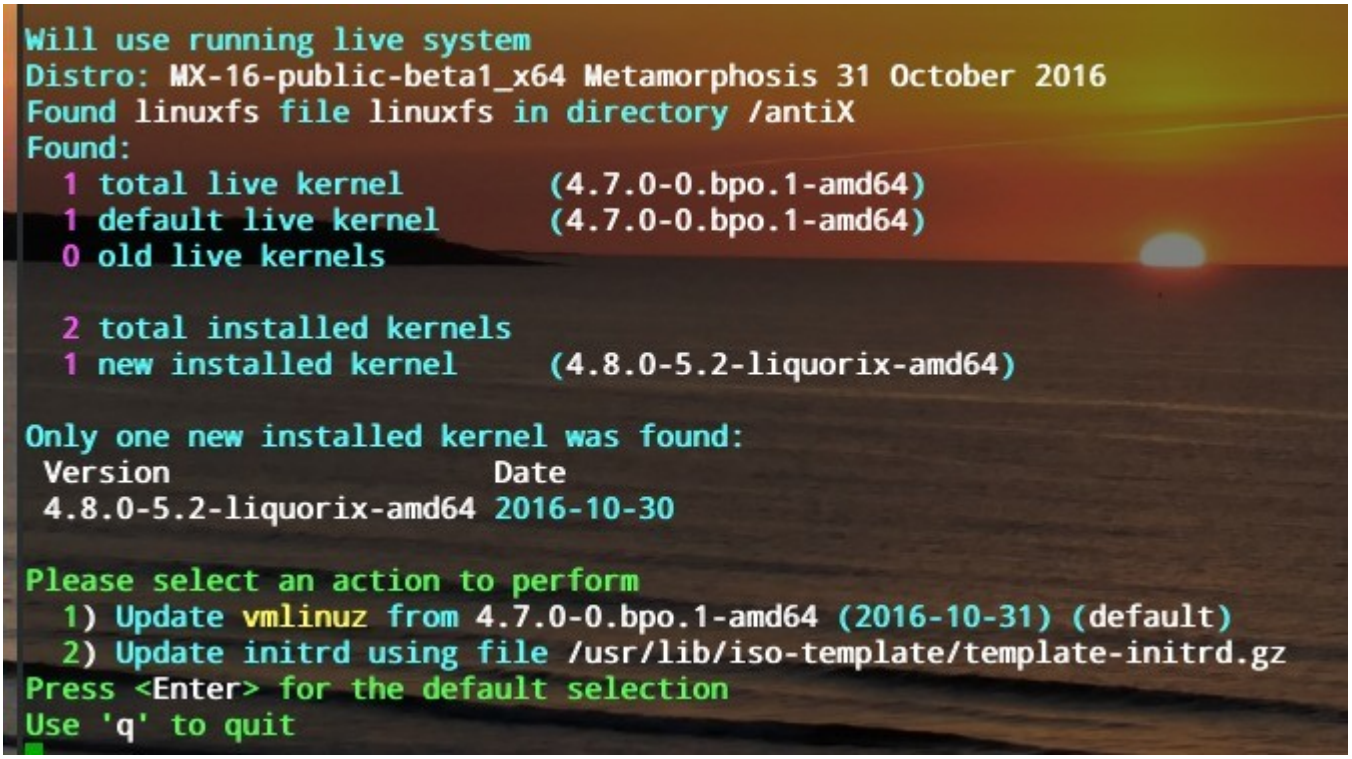

Figure 6-6: The live-usb kernel updater tool ready to switch to a new kernel

HELP: here.

6.6.2 Live-usb maker (GUI)

Use to create a live-usb starting from an iso file, a live -cd/dvd or an existing live-usb or even a running live system. Although UNetbootin is also available by default (see Section 2.2.3), live-usb maker has a number of advantages:

- It is faster.
- Saves state files across reboots
- LiveUSB-Storage for saving files directly on the live-usb
- Persistence
- Remastering
- Live kernel updating

NOTE: the CLI form (**live-usb-maker**, run as root) offers many advanced options.

Figure 6-7: The live-usb maker tool ready to create a live usb

HELP: here.

6.6.3 Live remaster/persistence (RemasterCC)

Make a snapshot of an installed system

NOTE: this application will only show in MX Tools when running a Live session.

Remaster

WARNING: for use in a Live session only!

The primary purpose of live remastering is to make it as safe, easy, and convenient as possible for users to make their own customized version of MX Linux that can be distributed to other computers.

The idea is that you use a LiveUSB (or a LiveHD: a frugal install; see the MX/antiX Wiki) to a hard drive partition as the development and testing environment. Add or subtract packages and then when you are ready to remaster, use a simple remaster script or GUI to do the remaster and then reboot. If something goes horribly wrong, simply reboot again with the rollback option and you will boot into the previous environment.

MX Community members use live remastering to produce unofficial spins, such as a KDE version and set of tools known as Workbench.

MX 16 - Remaster your Live-USB

MX Spins: Workbench!

 MX Spins: Stevo's KDE!

Persistence

WARNING: for use in a Live session only!

WARNING: may fail with large upgrades that can overwhelm the RAM. Alternative procedures are available.

- Carry out the upgrade in stages, using small (e.g., 200MB) amounts each time
- Wait for the next monthly snapshot, and reinstall. Make sure to copy any non-system folders off the stick beforehand

 Live USB with persistence (legacy mode)

 Live USB with persistence (UEFI mode)

 MX-16 live-USB with Persistence

Persistence is a hybrid between a LiveMedium and a full install; it allows you to retain any files you install or add during a live session. Programs installed or removed from, and customizations to, the "demo" user files during live persistence will be carried over to the installed system.

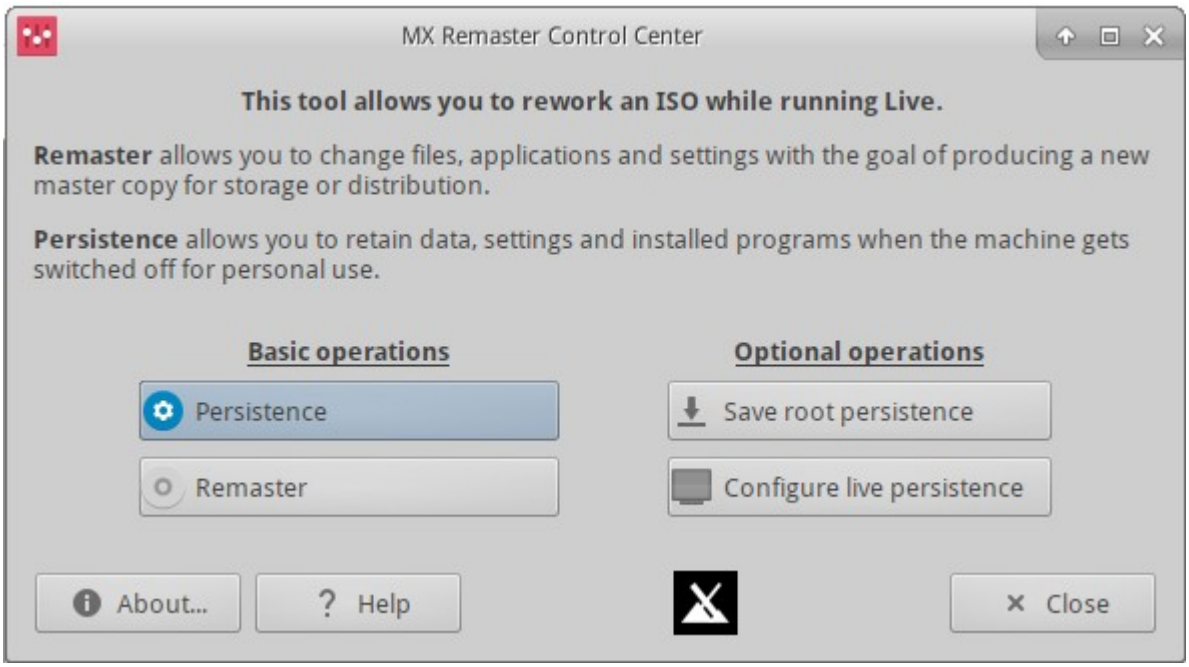

Figure 6-8: The remaster and persistence tool

HELP: here.

6.6.4 Snapshot

This tool makes a copy of your running system and creates an ISO from it.

The ISO can be put on a LiveMedium in the usual manner (see Section 2.2). To then install from the LiveMedium, open a root terminal and enter the command: *minstall*.

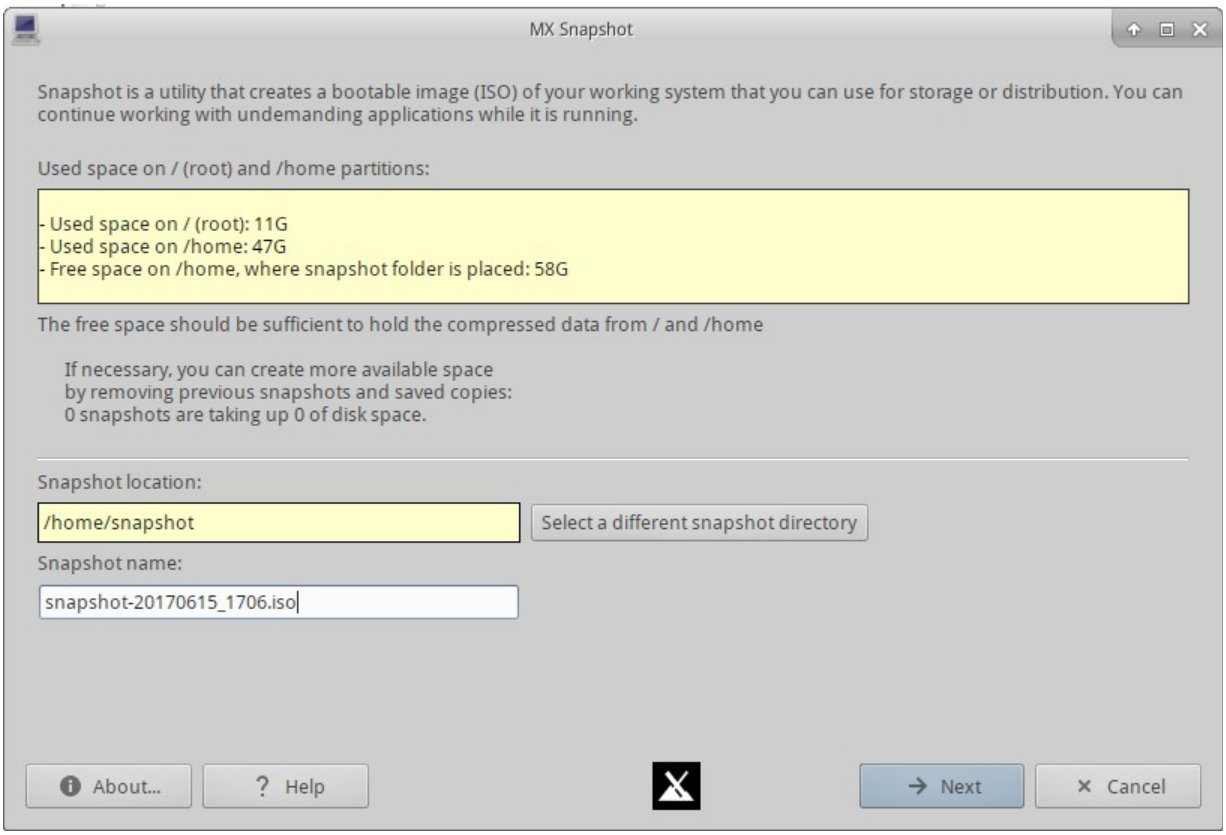

Figure 6-9: Opening screen of Snapshot

HELP: here.

6.7 SSH

SSH (Secure Shell) is a protocol used to securely log onto remote systems. It is the most common way to access remote Linux and Unix-like computers. MX Linux comes with the main packages necessary to run SSH in active mode, the main one being OpenSSH, a free implementation of the Secure Shell that consists of a whole suite of applications.

- Start or restart the ssh daemon as root with the command:

  /etc/init.d/ssh start

- To start the ssh daemon automatically when the computer starts, click **All Settings > Session and Startup > Application Autostart**. Click the Add button, then in the dialog box insert a name such as StartSSH, a short description if you want, and the command

  /etc/init.d/ssh start

  Press OK and you are done. The next time you restart, the SSH daemon will be active.

- KDE users on MX Linux can do the same using **Preferences > Settings > Start & Stop > Automatic Start**.

6.7.1 Troubleshooting

Occasionally, SSH does not work in passive mode, sending a message of denied connection. Then you can try the following:

- Edit as root the file ’/etc/ssh/sshd-config’. About line 16 you will find the parameter ’UsePrivilegeSeparation yes’. Change that to

  UsePrivilegeSeparation no

- Add yourself (or the intended users) to the group ’ssh’ using MX User Manager or editing as root the file /etc/group.

- Sometimes the certificates can be missing or outdated; an easy way to rebuild them is to run (as root) the command

  ssh-keygen -A

- Check if sshd is running by typing

  /etc/init.d/ssh status

  The system should answer ’[ok] sshd is running.’

- If you are using a firewall, check that port 22 is not blocked. It must allow IN and OUT traffic.

MORE: Openssh manual