Developer Guide

Linux Mint

Jan 26, 2024

GETTING STARTED

If you want to help us develop Linux Mint, you've come to the right place!

Welcome to the Linux Mint Development Guide.

CHAPTER

ONE

REQUIREMENTS

You don't need much to get started. If you know more than us you'll teach us; if we know more than you, we'll teach you. :)

That said, there are a few things you'll require before going further. Let's go through them.

1.1 Speaking English

Developers come from all around the world; but, if you speak English, then you'll be able to work with just about anybody.

You're probably OK if you're here reading this guide. You don't need to be fluent or to have great English, but you need to understand English sufficiently to communicate with us.

1.2 Knowing how to use Git

Git is the version control system we're using to keep track of changes. We're using it all the time and everywhere.

If you don't know about Git, stop right there: you need to learn it.

To learn Git, visit Github.io.

Make sure you're familiar with the concepts of commits, branches, remotes, reverts and rebases.

Hint: If you're new to Git, enjoy! It's both easy and really fun.

If you ask our developers, then most of them will tell you that Git is by far their favorite toy.

1.3 Knowing how to use Github

We're using Github to host our Git repositories and to work together on the code.

You'll need to have a Github account set up.

To open a Github account, visit Github.com.

You'll also need to know how to use Github to browse code changes, to fork a project, to make pull requests, etc.

To set up your Github account properly and learn how to use Github, visit the Github Help.

1.4 Running Linux

For most projects, you'll need a computer running the latest version of Linux Mint or else the latest version of LMDE.

You can run an earlier version, or a different distribution, but if you run the latest Linux Mint or LMDE release you're guaranteed everything will work.

CHAPTER TWO

SET UP

This chapter explains how to get your computer set up.

2.1 Create a Sandbox

When you build projects it produces .deb packages in their parent directory, so it's a good idea to create a directory for all your development needs, in which you'll have subdirectories for each project, or each group of projects. This keeps things tidy and well organized in your computer so it becomes easier to search for code across different projects.

We commonly call our main development directory "Sandbox" and place it in our home folder.

```
mkdir ~/Sandbox
```

Of course, you can call your "Sandbox" whatever you want and place it anywhere you want as well.

2.2 Install mint-dev-tools

Install the *mint-dev-tools* package from the Linux Mint repositories.

```
apt update
apt install mint-dev-tools --install-recommends
```

It contains useful tools to help you compile and develop Linux Mint projects.

CHAPTER THREE

TECHNOLOGY

This chapter gives you an overview of the technology we're using.

3.1 Computer Languages

We use a variety of computer languages in Linux Mint.

You don't need to know them all and you don't need to know them well. It really depends on which project you want to work and what you want to achieve.

Here are the languages we use the most.

3.1.1 Python

Scripts which run in terminals or in the backgrounds are usually either written in Bash or in Python.

Some software applications and most configuration tools are also written in Python.

The advantage of Python is that it is easy to learn and fast to develop with.

3.1.2 C

Many software applications and most libraries are written in C.

The C language is low-level, hard to master and tedious to develop with, but it gives fast performance and it's the most supported language in Linux (everything is accessible from C).

3.1.3 Javascript

The graphical elements of Cinnamon, as well as Cinnamon applets, desklets and extensions are written in Javascript.

3.1.4 Vala

Vala is used in Slick Greeter (the login screen).

3.2 GNOME Toolkit and libraries

All our user interfaces use GTK3 toolkit.

Our development relies heavily on the GNOME libraries, in particular we use Gio, GLib, GObject and dconf a lot.

In C we access these libraries directly.

In Python and Javascript we access them via GObject Introspection.

3.3 Tools

3.3.1 Development environment

To write and edit code, you can use anything you want. Some people prefer lightweight editors while others prefer full-fledge IDEs. It's a matter of taste. Development is all about fun, so what matters the most is that you love the tools you use.

If you're not sure what to use, have a look around and try a few editors/IDE until you find your favorite one.

Many developers within the team use Sublime.

```
apt update
apt install sublime-text
```

If you install Sublime, also install its Package Control and then use it to install the *GitGutter* and *TrailingSpaces* extensions.

Visual Studio Code is also very popular within the team.

You can also check out Atom, Brackets and Geany.

And if you want a complete IDE, there's also Eclipse and Netbeans.

3.3.2 Version control

There's less choice when it comes to version control because we're all using git and nothing else. All our code is version-controlled with it.

That being said, you don't necessarily have to use the git command line.

Here are a few tools you can use to make using git easier.

gitk is ugly and looks dated (it was developed in Tcl/Tk) but it's very useful to quickly look at the commit history, to create branches and to cherry pick.

You can install it from the repositories:

```
apt update
apt install gitk
cd ~/Sandbox/
git clone https://github.com/linuxmint/mintsystem.git
cd mintsystem
gitk
```

From a project directory, simply type *gitk* to see the history of commits. You can also specify a branch name to see that branch instead, or a subdirectory to only see the history of a particular directory.

gitg is very similar and it looks better (it's using Gtk), but its feature set is slightly different.

```
apt update
apt install gitg
cd ~/Sandbox/
git clone https://github.com/linuxmint/mintsystem.git
cd mintsystem
gitg
```

Note: gitg is included in mint-dev-tools. You can find gitg and other tools that will help with development on mint already installed.

From the repository you can also look at *git-cola* and *git-gui*.

If you're looking for a more complete solution, have a look at Gitkraken.

And last but not least, check the plugins and features available in your IDE/editor. Visual Studio Code, Atom and Sublime in particular come with a lot of support for Git and Github.

3.3.3 Glade

We can write our user interfaces in programming language, or we can use Glade and draw them with the mouse.

Glade is a tool to design and edit GTK user interfaces and save them in XML (in a .glade or .ui file).

```
apt update
apt install glade
```

Once a user interface is saved, we simply tell our program to open that file and we can access the widgets from it programmatically.

Many of our projects separate the code from the user interface.

3.3.4 devhelp

Devhelp shows the reference manuals for the development libraries installed on your computer. For most libraries, the documentation is included in their *-dev* or *-doc* package (for instance, if you're working with GTK3, make sure to install *libgtk-3-dev* and *libgtk-3-doc*).

```
apt update
apt install devhelp
```

You can launch DevHelp from the applications menu and use it to browse or search the libraries reference manuals. You'll often need to check the syntax or the arguments of a particular function. It's nice to be able to get the information locally without having to search online.

3.3.5 d-feet

Some programs use DBus to communicate with others. We use d-feet to browse and troubleshoot DBus.

```
apt update
apt install d-feet
```

With d-feet you can quickly find a service on DBus, browse its interface and even call some of its functions manually.

3.3.6 meld

Meld is a visual diff tool. It shows the differences between two files and it's great at it.

```
apt update
apt install meld
```

3.3.7 Other cool tools

Most of our configuration is stored in dconf and we use gsettings (from the command line) to look at it or modify it. If you want to do it graphically, you can install dconf-editor.

awf is useful to test widgets when working on GTK themes.

```
apt update
apt install awf dconf-editor
```

CHAPTER

FOUR

MINT TOOLS

The first Mint tools were developed around 2006 when the Linux Mint project was born. Throughout the years, new tools were added to Linux Mint to implement functions that it was missing, or to make the user experience easier and more comfortable.

Some tools, which were very useful in the past, also disappeared when they were no longer needed. Here's a list of the currently active tools projects developed by Linux Mint.

4.1 mint-common

Common utility functions and libraries used by the Mint tools are placed in the mint-common project.

This project is developed in Python and its source code is available on Github.

4.2 mintbackup

The *Backup Tool*, mintbackup, makes it easy to save and restore backups of files within the home directory.

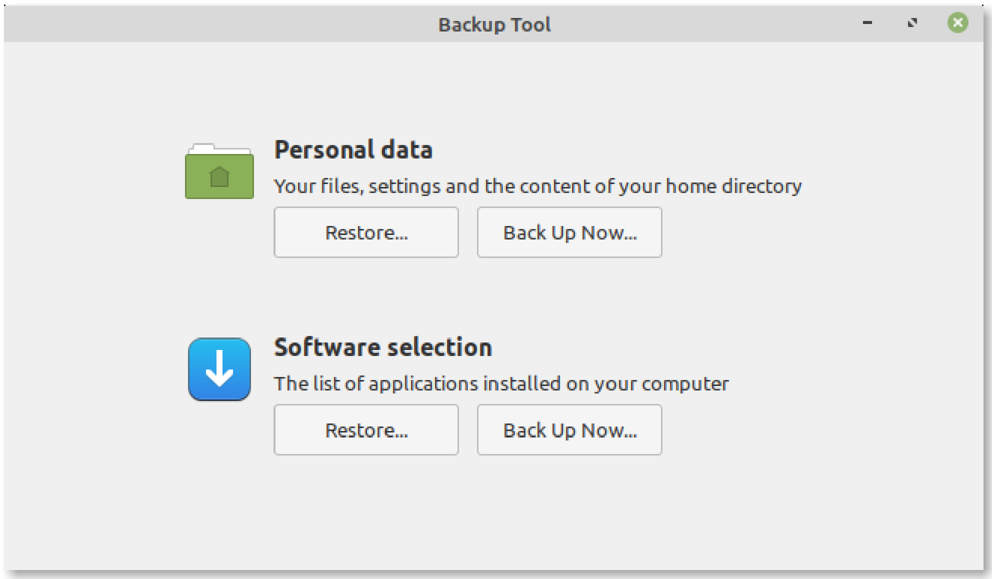

Backup Tool
Personal data
Your files, settings and the content of your home directory
Restore...
Back Up Now...
Software selection
The list of applications installed on your computer
Restore...
Back Up Now...

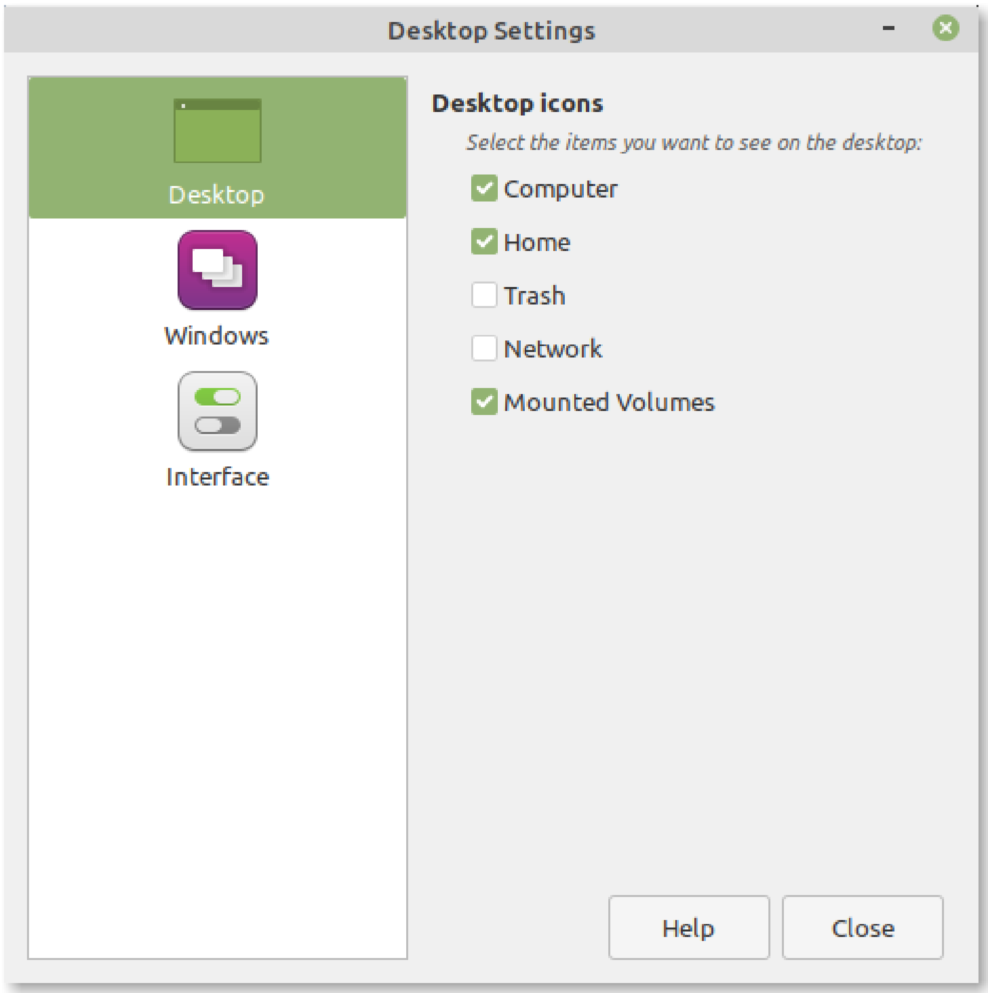
Fig. 1: Backup Tool

It also supports the ability to save the list of installed packages, so they can be reinstalled later.

This project is developed in Python and its source code is available on Github.

4.3 mintdesktop

This is a tool which provides some additional settings for the MATE desktop environment and the ability to switch window managers.



Desktop icons

Select the items you want to see on the desktop:

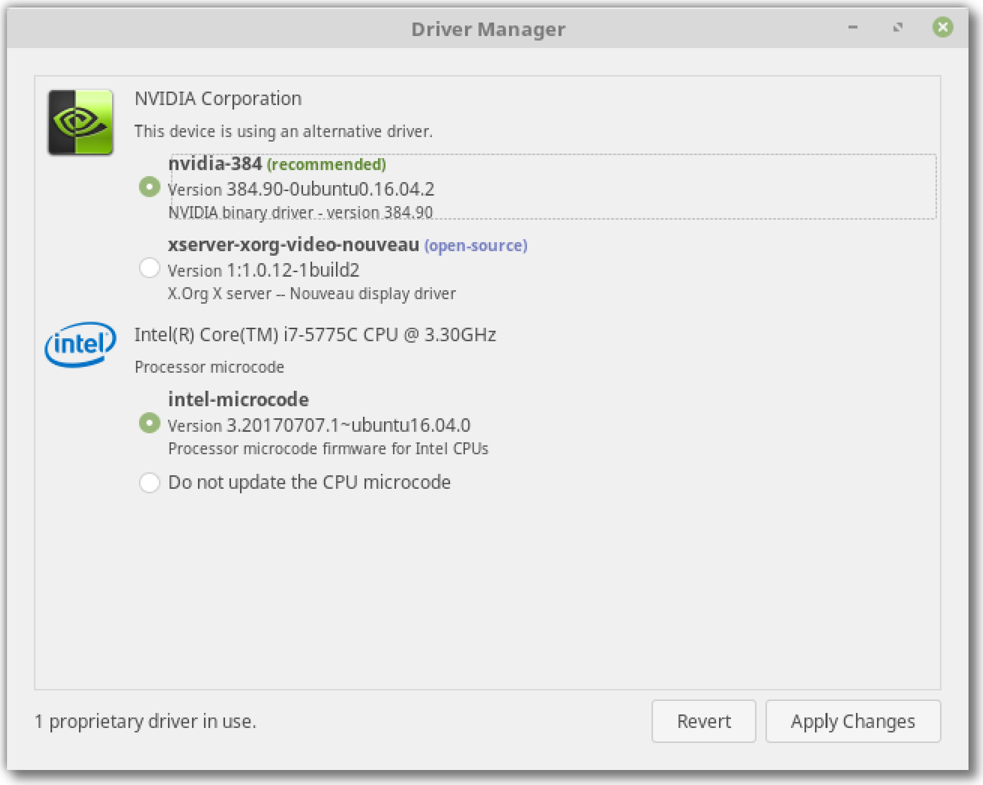
- ✓ Computer
- ✓ Home
- ☐ Trash
- ☐ Network
- ✓ Mounted Volumes

Fig. 2: Desktop Settings

This project is developed in Python and its source code is available on Github.

4.4 mintdrivers

The *Driver Manager*, mintdrivers, makes it easy to install proprietary drivers when applicable.


Driver Manager

NVIDIA Corporation
This device is using an alternative driver.

[✓ **nvidia-384** (recommended)
Version 384.90-0ubuntu0.16.04.2
NVIDIA binary driver - version 384.90]

xserver-xorg-video-nouveau (open-source)
Version 1:1.0.12-1build2
X.Org X server -- Nouveau display driver

Intel(R) Core(TM) i7-5775C CPU @ 3.30GHz
Processor microcode

[✓ **intel-microcode**
Version 3.20170707.1~ubuntu16.04.0
Processor microcode firmware for Intel CPUs]

Do not update the CPU microcode

1 proprietary driver in use.

Fig. 3: Driver Manager

It relies on the *ubuntu-drivers* backend and isn't available in LMDE.

This project is developed in Python and its source code is available on Github.

4.5 mintinstall

The *Software Manager*, mintinstall, is an App store for Free Software. It provides access to popular applications from within the repository.

It's also compatible with Flatpak and able to list flatpaks from multiple flatpak repositories.

This project is developed in Python and its source code is available on Github.

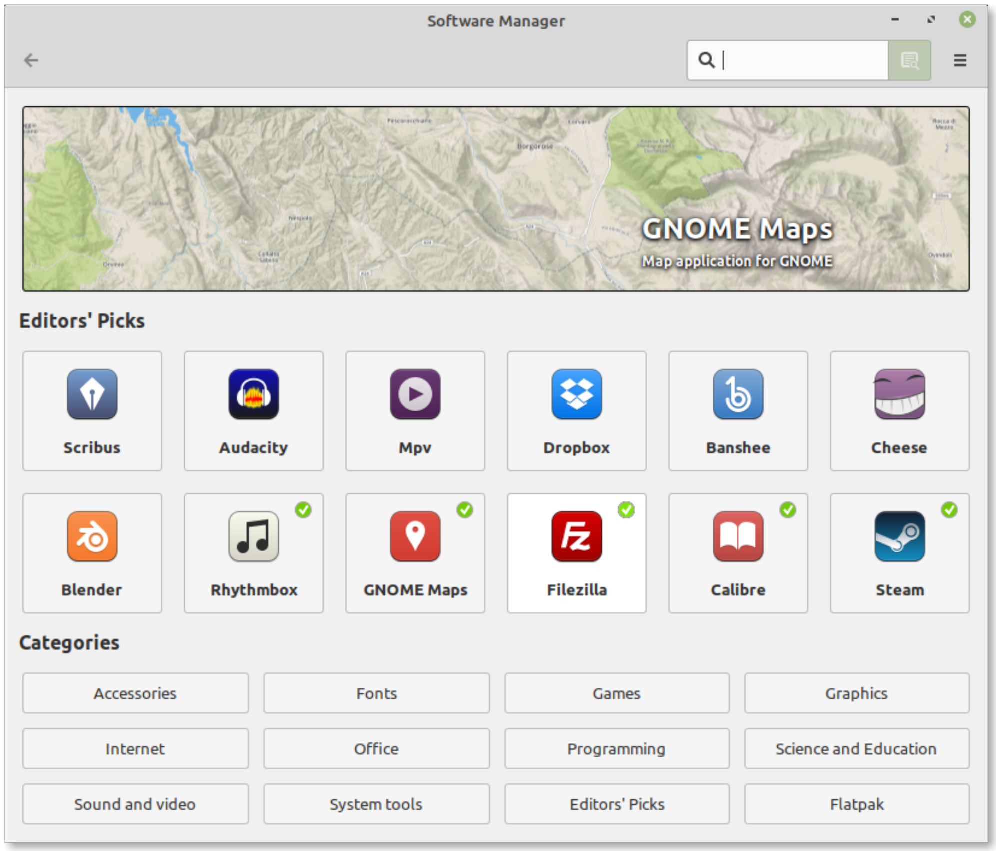
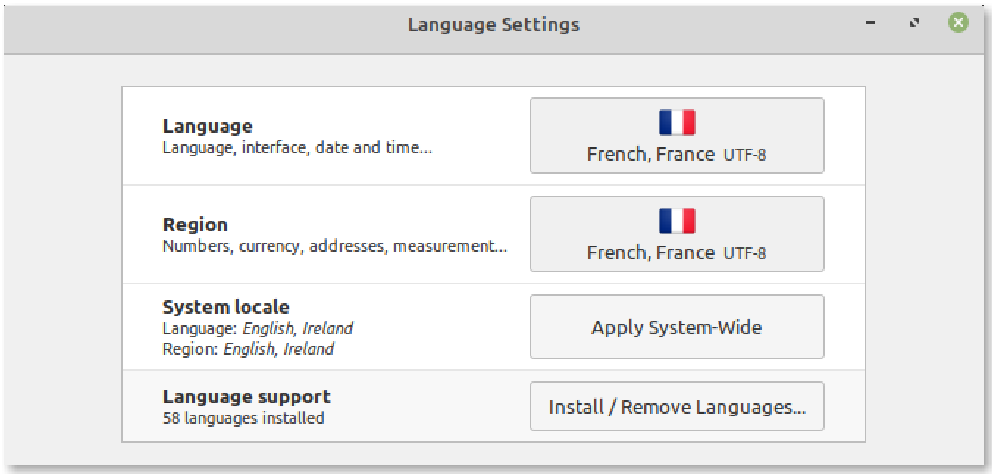


Fig. 4: Software Manager

4.6 mintlocale

The mintlocale project provides two configuration tools.

The first one is dedicated to locale selection and installation.



Language
Language, interface, date and time...
French, France UTF-8

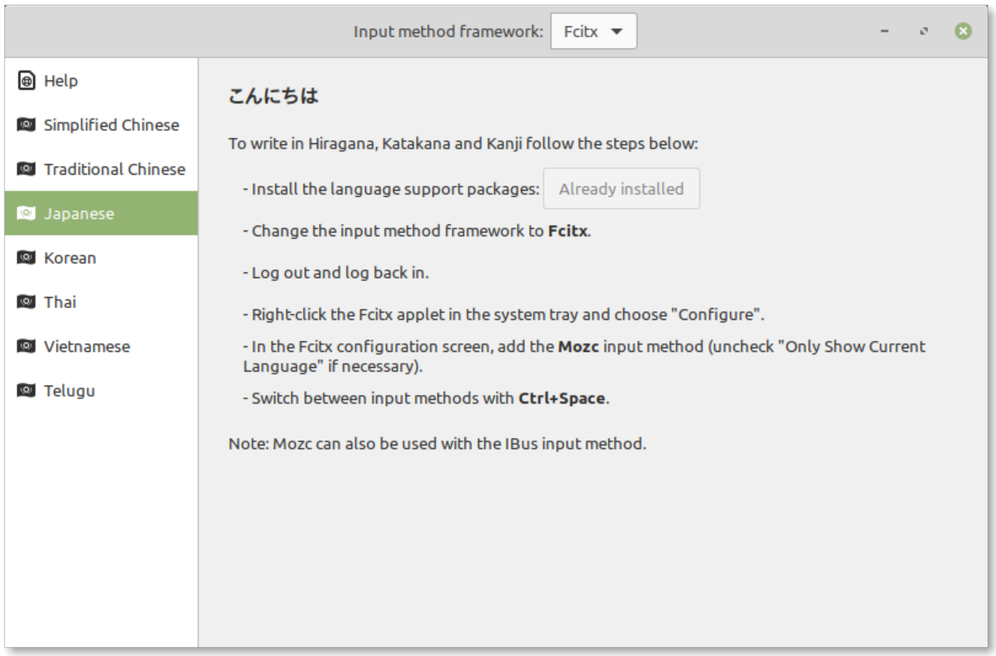
Region
Numbers, currency, addresses, measurement...
French, France UTF-8

System locale
Language: *English, Ireland*
Region: *English, Ireland*
Apply System-Wide

Language support
58 languages installed
Install / Remove Languages...

Fig. 5: Language Settings

The second one is dedicated to input methods:



こんにちは

To write in Hiragana, Katakana and Kanji follow the steps below:

- Install the language support packages: Already installed
- Change the input method framework to **Fcitx**.
- Log out and log back in.
- Right-click the Fcitx applet in the system tray and choose "Configure".
- In the Fcitx configuration screen, add the **Mozc** input method (uncheck "Only Show Current Language" if necessary).
- Switch between input methods with **Ctrl+Space**.

Note: Mozc can also be used with the IBus input method.

Fig. 6: Input Methods

This project is developed in Python and its source code is available on Github.

4.7 mintmenu

This is the main application menu for the MATE edition of Linux Mint.

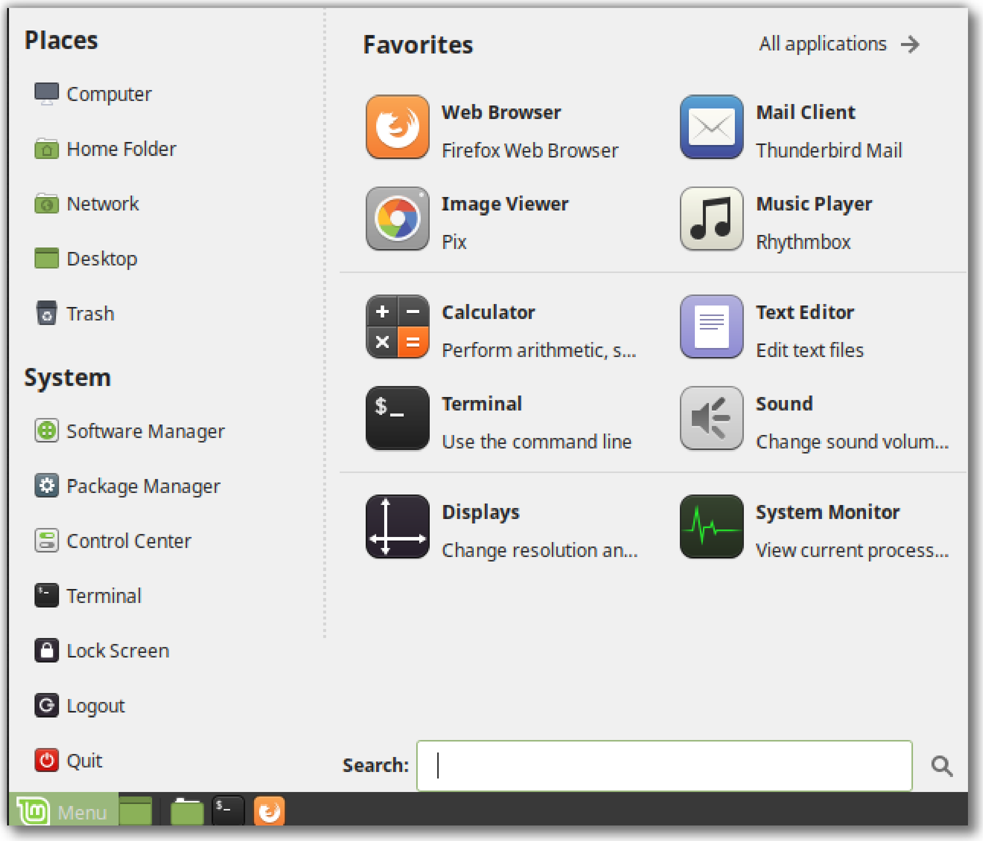


Fig. 7: MintMenu

This project is developed in Python and its source code is available on Github.

4.8 mintnanny

The *Domain Blocker*, mintnanny, blocks outgoing traffic towards chosen domain names using /etc/hosts.

This project is developed in Python and its source code is available on Github.

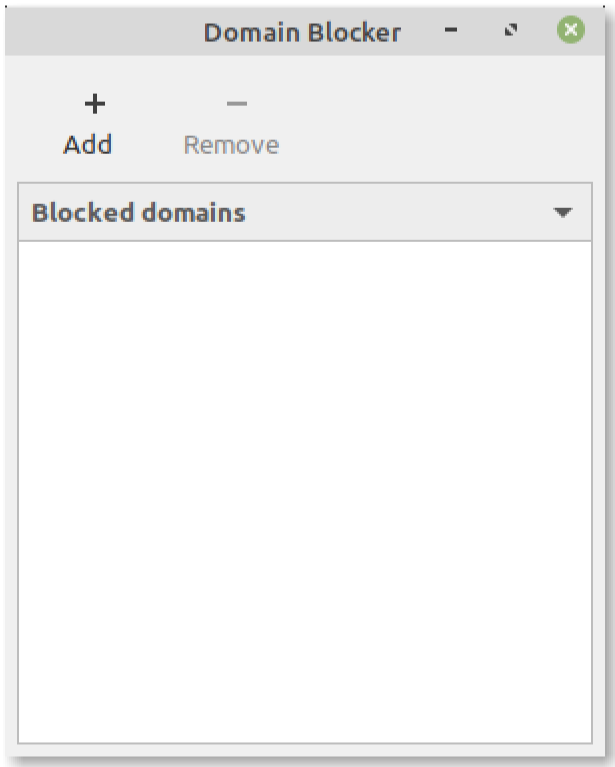


Fig. 8: Domain Blocker

4.9 mintreport

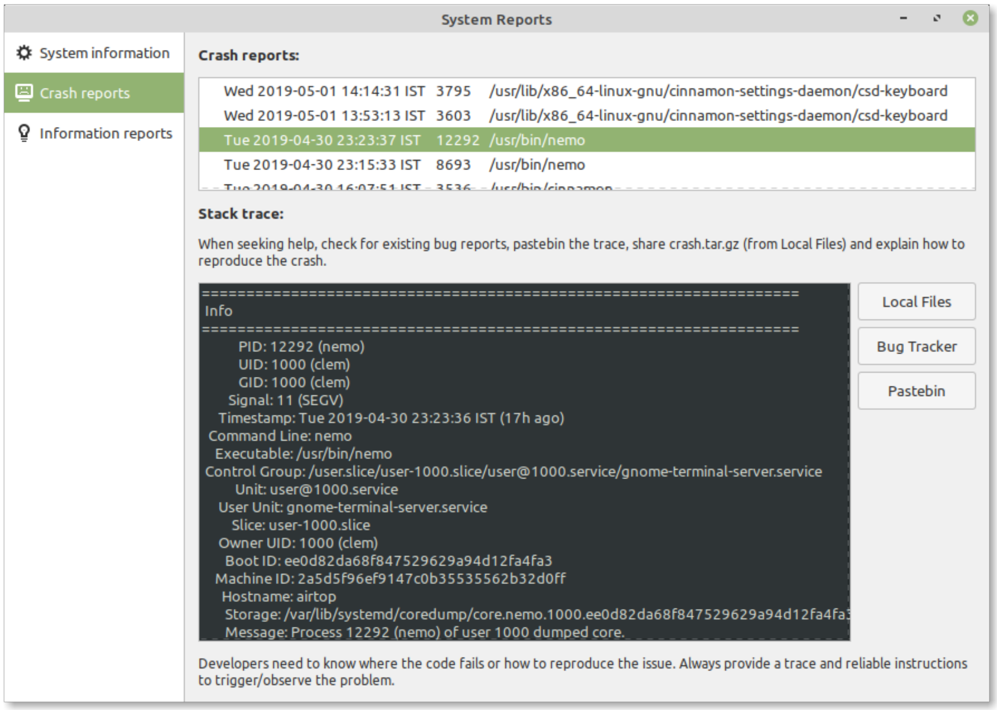
The *System Reports*, mintreport, provides system information and helps the user collect information about application crashes.

This project is developed in Python and its source code is available on Github.

4.10 mintsources

The *Software Sources* configuration tool, mintsources, helps the user configure software repositories, choose a mirror, add PPAs and perform maintenance tasks related to package management.

This project is developed in Python and its source code is available on Github.



Crash reports:

```
Wed 2019-05-01 14:14:31 IST  3795   /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-keyboard
Wed 2019-05-01 13:53:13 IST  3603   /usr/lib/x86_64-linux-gnu/cinnamon-settings-daemon/csd-keyboard
Tue 2019-04-30 23:23:37 IST  12292  /usr/bin/nemo
Tue 2019-04-30 23:15:33 IST  8693   /usr/bin/nemo
```

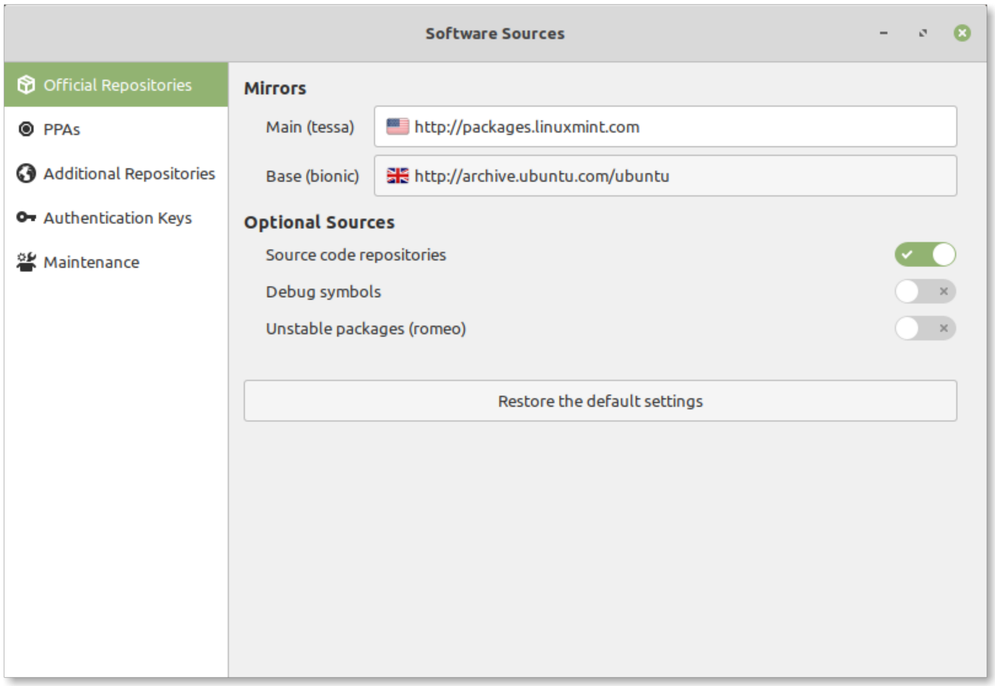
Stack trace:

When seeking help, check for existing bug reports, pastebin the trace, share crash.tar.gz (from Local Files) and explain how to reproduce the crash.

```
===================================================================
Info
===================================================================
           PID: 12292 (nemo)
           UID: 1000 (clem)
           GID: 1000 (clem)
        Signal: 11 (SEGV)
     Timestamp: Tue 2019-04-30 23:23:36 IST (17h ago)
  Command Line: nemo
    Executable: /usr/bin/nemo
 Control Group: /user.slice/user-1000.slice/user@1000.service/gnome-terminal-server.service
          Unit: user@1000.service
     User Unit: gnome-terminal-server.service
         Slice: user-1000.slice
     Owner UID: 1000 (clem)
       Boot ID: ee0d82da68f847529629a94d12fa4fa3
    Machine ID: 2a5d5f96ef9147c0b35535562b32d0ff
      Hostname: airtop
       Storage: /var/lib/systemd/coredump/core.nemo.1000.ee0d82da68f847529629a94d12fa4fa3
       Message: Process 12292 (nemo) of user 1000 dumped core.
```

Developers need to know where the code fails or how to reproduce the issue. Always provide a trace and reliable instructions to trigger/observe the problem.

Fig. 9: System Reports



Mirrors

Main (tessa): http://packages.linuxmint.com

Base (bionic): http://archive.ubuntu.com/ubuntu

Optional Sources

Source code repositories ✓

Debug symbols ✗

Unstable packages (romeo) ✗

Fig. 10: Software Sources Configuration Tool

4.11 mintstick

The mintstick project provides two utilities.

The first one is dedicated to formatting USB sticks.

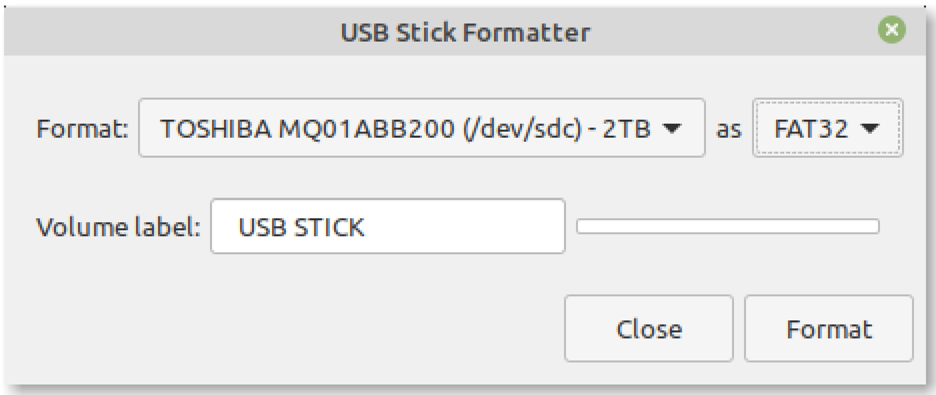


Fig. 11: USB Stick Formatter

The second one is used to make live USB sticks from ISO images:

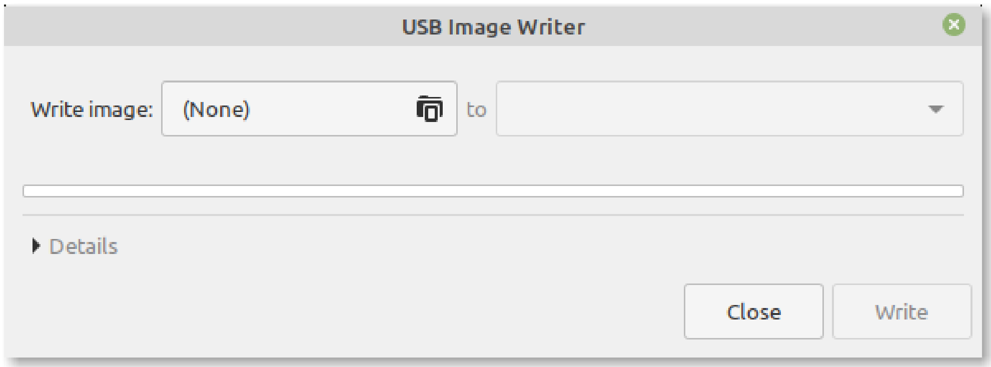


Fig. 12: USB Image Writer

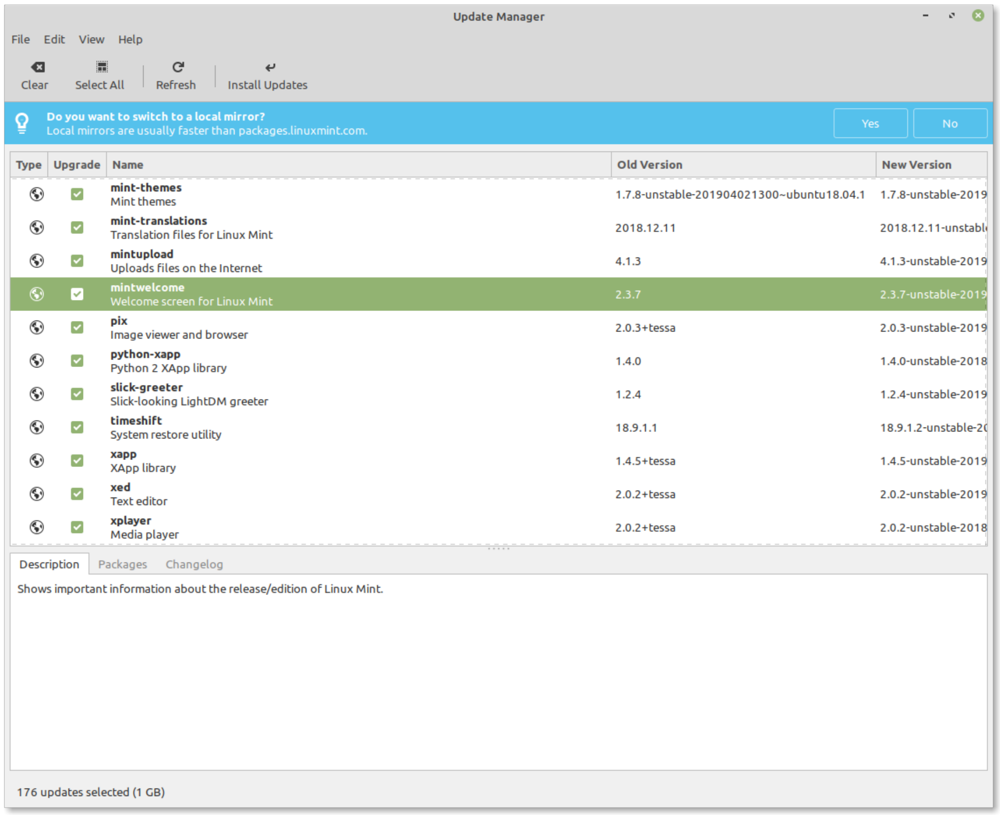
This project is developed in Python and its source code is available on Github.

4.12 mintsystem

This project provides small utilities, as well as files, scripts and resources used by the OS.

4.13 mintupdate

The *Update Manager*, mintupdate, provides users with software and security updates.



Do you want to switch to a local mirror?
Local mirrors are usually faster than packages.linuxmint.com.

Type	Upgrade	Name	Old Version	New Version
	✓	**mint-themes** Mint themes	1.7.8-unstable-201904021300~ubuntu18.04.1	1.7.8-unstable-2019
	✓	**mint-translations** Translation files for Linux Mint	2018.12.11	2018.12.11-unstabl
	✓	**mintupload** Uploads files on the Internet	4.1.3	4.1.3-unstable-2019
	✓	**mintwelcome** Welcome screen for Linux Mint	2.3.7	2.3.7-unstable-2019
	✓	**pix** Image viewer and browser	2.0.3+tessa	2.0.3-unstable-2019
	✓	**python-xapp** Python 2 XApp library	1.4.0	1.4.0-unstable-2018
	✓	**slick-greeter** Slick-looking LightDM greeter	1.2.4	1.2.4-unstable-2019
	✓	**timeshift** System restore utility	18.9.1.1	18.9.1.2-unstable-2
	✓	**xapp** XApp library	1.4.5+tessa	1.4.5-unstable-2019
	✓	**xed** Text editor	2.0.2+tessa	2.0.2-unstable-2019
	✓	**xplayer** Media player	2.0.2+tessa	2.0.2-unstable-2018

Shows important information about the release/edition of Linux Mint.

176 updates selected (1 GB)

Fig. 13: Update Manager

This project is developed in Python and its source code is available on Github.

4.14 mintupload

The *Upload Manager*, mintupload, allows the user to upload files to a particular location, without browsing it, just by dropping the files with the mouse.

This project is developed in Python and its source code is available on Github.

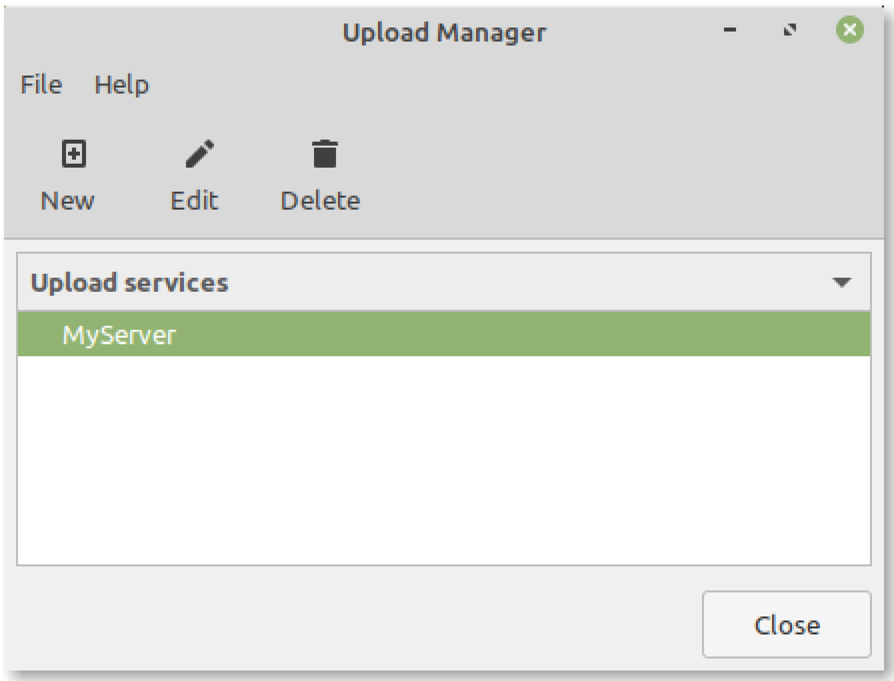
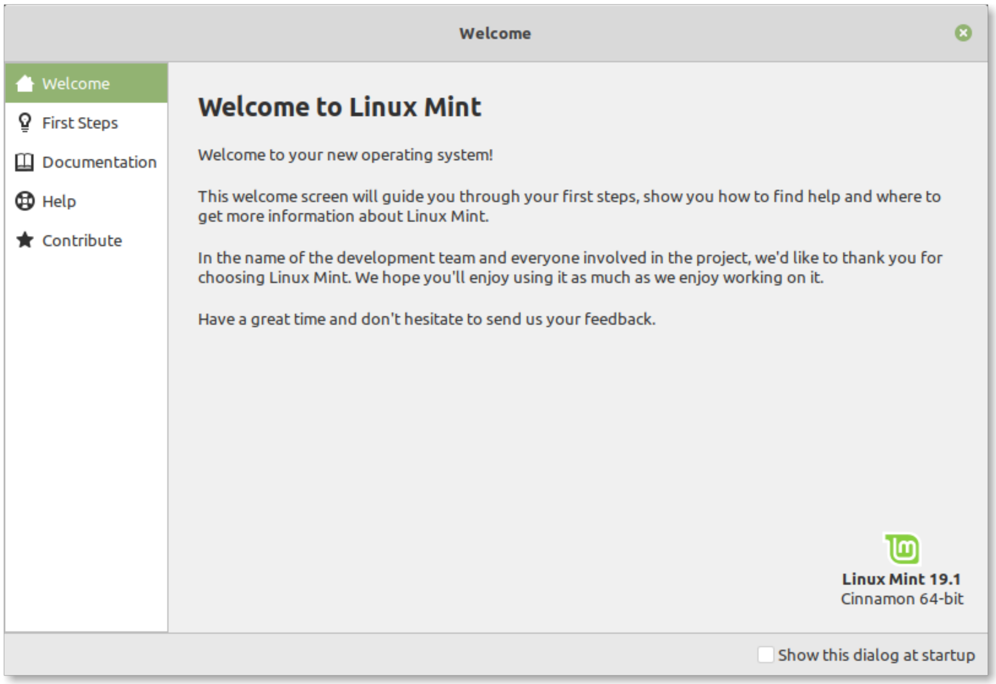


Fig. 14: Welcome Screen

4.15 mintwelcome

The *Welcome Screen*, mintwelcome, welcomes new users into Linux Mint and guides them through their first steps.

This project is developed in Python and its source code is available on Github.



Welcome to Linux Mint

Welcome to your new operating system!

This welcome screen will guide you through your first steps, show you how to find help and where to get more information about Linux Mint.

In the name of the development team and everyone involved in the project, we'd like to thank you for choosing Linux Mint. We hope you'll enjoy using it as much as we enjoy working on it.

Have a great time and don't hesitate to send us your feedback.

Fig. 15: Welcome Screen

CHAPTER

FIVE

CINNAMON

The Cinnamon desktop environment is a very large development project.

Between 2006 and 2010 the main desktop environment for Linux Mint was GNOME 2. It was very stable and very popular.

In 2011, Linux Mint 12 was unable to ship with GNOME 2. The upstream GNOME team had released a brand new desktop (GNOME 3 aka "Gnome Shell") which was using new technologies (Clutter, GTK3), which had a completely different design and implemented a radically different paradigm than its predecessor but which used the same namespaces and thus it couldn't be installed alongside GNOME 2. Following the decision from Debian to upgrade GNOME to version 3, GNOME 2 was no longer available in Linux Mint.

To tackle this issue two new projects were started:

- A project called "MATE" was started by a developer called Perberos. Its goal was to rename and repackage GNOME 2 so that it could be just as it was before.
- A project called "MGSE" was started by Linux Mint. Its goal was to develop extensions for GNOME 3 to give it back some of the functionality it had lost and which was available in GNOME 2 (a panel, a systray, an application menu, a window-centric alt-tab selector, a window-list..etc).

Linux Mint 12 shipped with both MATE and GNOME3+MGSE.

6 months later and after a huge amount of work, MATE was becoming stable, and from a set of extensions MGSE became a fork of GNOME 3 called Cinnamon.

Linux Mint 13 was the first Linux release to ship with the Cinnamon desktop. Since then Linux Mint has a MATE and a Cinnamon edition, both providing users with a conservative desktop paradigm, one forked from GNOME 2 and the other forked and derived from GNOME 3.

5.1 Processes

Fig. 1: Binary view of the various processes within a Cinnamon session

The figure above shows the various processes at play within a Cinnamon session.

After you log in, the following processes are automatically started:

- cinnamon-session (the session manager which starts all the other processes)
- cinnamon (which is the visual part of the cinnamon desktop)
- nemo-desktop (which handles the desktop icons and desktop context menu)
- cinnamon-screensaver (the screensaver)

- various csd-* processes (which are settings daemon plugins and run in the background)

The nemo process starts when you browse files and directories. It remains open as long as at least one file manager window is open.

The cinnamon-settings process starts when you launch the *System Settings* and remains open as long as at least one configuration module is open.

5.2 Libraries

5.2.1 cinnamon-menus

The *cinnamon-menus* library provides utility functions to read and monitor the set of desktop applications installed on the computer. Thanks to *cinnamon-menus*, Cinnamon can quickly list installed applications within the application menu, fetch application icons for the menu, the alt-tab selector and the window-list and keep this data in sync whenever applications are installed or removed from the computer.

The cinnamon-menus library is developed in C and the source code is available on Github.

5.2.2 cinnamon-desktop

cinnamon-desktop is a set of utility libraries and settings used by other Cinnamon components.

Whenever multiple desktop components need to access the same resource (whether this is a setting or a utility function), we place this resource in cinnamon-desktop.

Here's an overview of some of the resources currently in *cinnamon-desktop*:

cinnamon.desktop	dconf settings schemas used by several Cinnamon components
libcvc	A PulseAudio utility library used to control sound volume and devices
gnomerr	An Xrandr utility library to detect, load and save monitor configurations
gnome-xkb	A keyboard layout utility library
gnome-bg	A wallpaper utility library
gnome-installer	A cross-distribution library used to install software applications

The cinnamon-desktop library is developed in C and the source code is available on Github.

5.2.3 muffin

Muffin, or *libmuffin* to be more precise is a window management library.

Within the Cinnamon desktop environment, the Window Manager isn't running in a separate process. The main *cinnamon* process implements the *libmuffin* library and therefore runs both the visible components (panel, applets..etc) and the window manager.

Note: The *muffin* package also provides a *muffin* binary. This binary is a small program which implements *libmuffin* and provides a minimal window manager, sometimes used by the developers as a troubleshooting tool. Note that whether or not *muffin* is installed by default in Linux Mint, it doesn't run by default in a Cinnamon session. The *cinnamon* process, which also implements *libmuffin*, is the default window manager.

The clutter and cogl libraries are also part of the muffin package now. Clutter is a library for creating and displaying both 2d and 3d graphical elements. It is used both by muffin itself (eg. for compositing and setting up the stage), and also by St in cinnamon (all St widgets are clutter actors). Cogl is a library that clutter uses for 3d rendering.

Muffin is developed in C and the source code is available on Github.

5.2.4 cjs

CJS is Cinnamon's Javascript interpreter. It uses MozJS (Mozilla's SpiderMonkey) and makes it possible to work with GObject and interact with GIR, GNOME and Cinnamon libraries using that language.

CJS is run by and within the main *cinnamon* process and the parts of the desktop written in Javascript are contained in the main Cinnamon component.

CJS is developed in C++ and Javascript and the source code is available on Github.

5.3 Core components

5.3.1 cinnamon-session

The Cinnamon session manager is responsible for launching all the components needed by the session after you log in, and closing the session properly when you want to log out.

Among other things, the session manager launches the core components required by the session (such as the desktop itself and its components), as well as applications which are configured to start automatically.

Cinnamon-session also provides a DBus interface called the Presence interface, which makes it easy for applications such as media players to set the sessions as busy and inhibit power management (suspend, hibernate, etc…) and the screensaver during video playback.

Last but not least, the session management lets applications register so they can be closed cleanly. The text editor for instance is registered to the session when launched and interacts with it on logout. If a document isn't saved, the session is aware of it and lets you save your work before proceeding to log out.

5.3.2 cinnamon-settings-daemon

cinnamon-settings-daemon is a collection of processes which run in the background during your Cinnamon session.

Here's a description of some of these processes.

csd-automount	Automatically mounts hardware devices when they are plugged in
csd-clipboard	Manages the additional copy-paste buffer available via Ctrl+C/Ctrl+V
csd-housekeeping	Handles the thumbnail cache and keeps an eye on the space available on the disk
csd-keyboard	Handles keyboard layouts and configuration
csd-media-keys	Handles media keys
csd-mouse	Handles mice and touch devices
csd-orientation	Handles accelerometers and screen orientation
csd-power	Handles battery and power management
csd-print-notifications	Handles printer notifications
csd-wacom	Handles wacom devices
csd-xrandr	Handles screen resolution and monitors configuration
csd-xsettings	Handles X11 and GTK configuration

Cinnamon-settings-daemon is developed in C and the source code is available on Github.

5.4 Visible desktop layer

5.4.1 cinnamon-screensaver

The Cinnamon screensaver is responsible for locking the screen and to a lesser extent for handling some power management functions (although most of these are handled by csd-power within the Cinnamon Settings Daemon).

Cinnamon-screensaver is developed in Python and the source code is available on Github.

5.4.2 cinnamon

The Cinnamon github project is the biggest and most active project within the overall project.

It contains various subcomponents written in C:

St	Cinnamon's widget toolkit written on top of Clutter
Appsys	An abstraction of Gio.AppInfo and cinnamon-menus, providing metadata on installed applications
DocInfo	An abstraction of recently opened documents
Tray	A small library for managing status icons

The visible layer of the desktop is written in Javascript:

Cinnamon JS	The panels, window management, HUD, effects and most of what you see…
Applets	The applets within the panel
Desklets	The desklets on top of the desktop

The System Settings, its configuration modules and utility scripts are written in Python.

Cinnamon is developed in C, Python and Javascript and the source code is available on Github.

5.4.3 nemo

Nemo is Cinnamon's file manager. When you open up your home directory or browse files you're running Nemo.

Another little part of Nemo is *nemo-desktop*. Its role is to handle desktop icons and the desktop context menu.

When you log in, *nemo-desktop* is started automatically by cinnamon-session. The *nemo* process itself only starts when you're browsing through the directories and stops when you close the last opened file manager window.

Nemo is developed in C and the source code is available on Github.

5.4.4 nemo-extensions

Nemo provides a set of APIs and is very easy to extend, both in C and in Python. *nemo-extensions* is the Github project where common extensions are stored.

Some Nemo extensions are developed in C and some in Python. Their source code is available on Github.

5.4.5 cinnamon-control-center

Although *cinnamon-settings* (which is part of the Cinnamon project itself) and most of its modules are written in Python. A few configuration modules are still written in C.

Note: Historically, when Cinnamon was forked from GNOME 3, all configuration modules were written in C, as part of gnome-control-center. At the beginning of the Cinnamon project, all configurations modules were thus written in C and were part of cinnamon-control-center. Since then the vast majority of modules were rewritten from scratch in Python and moved to the Cinnamon project itself.

Nowadays, only a few modules are still in cinnamon-control-center:

color	Color profiles
datetime	Date and Time configuration
display	Display and monitors configuration
network	Network configuration
online-accounts	Online Accounts configuration
wacom	Wacom devices configuration

Cinnamon-control-center is developed in C and the source code is available on Github.

CHAPTER

SIX

XAPPS

A project called “X-Apps” was started in 2016 to produce generic applications for traditional GTK desktop environments.

The idea behind this project is to replace applications which no longer integrate properly outside of a particular environment (this is the case for a growing number of GNOME applications) and to give our desktop environments the same set of core applications, so that each change, each new feature being developed, each little improvement made in one of them will benefit not just one environment, but all of them.

The core ideas for X-Apps are:

- To use modern toolkits and technologies (GTK3 for HiDPI support, gsettings etc..)
- To use traditional user interfaces (titlebars, menubars)
- To work everywhere (to be generic, desktop-agnostic and distro-agnostic)
- To provide the functionality users already enjoy (or enjoyed in the past for distributions which already lost some functionality)
- To be backward-compatible (in order to work on as many distributions as possible)

Within Linux Mint, users didn’t need to adapt to X-Apps, because in many cases, they were very similar or exactly the same as the applications people were already using. For instance, Totem 3.18 was radically different than Totem 3.10 which shipped with Linux Mint 17, but Xplayer 1.0 (which was the default media player in Linux Mint 18) was exactly the same. The goal of the X-Apps is not to reinvent the wheel. Quite the opposite in fact, it’s to guarantee the maintenance of applications we already enjoyed and to steer their development in a direction that benefits multiple desktop environments.

It makes no sense to develop 3 different text editors, 5 different calculators and so on. When we work on projects like these, we want to make it count. An improvement in the text editor shouldn’t benefit only one edition, it should benefit all of them.

All three editions of Linux Mint come with the same XApps libraries and applications. When working on XApps, our development efforts are focused on improving all desktops.

6.1 libxapp

This is the XApps library. Anything that is cross-desktop goes in there.

It’s available in Python and JS as well, through GObject Introspection.

This project is developed in C and its source code is available on Github.

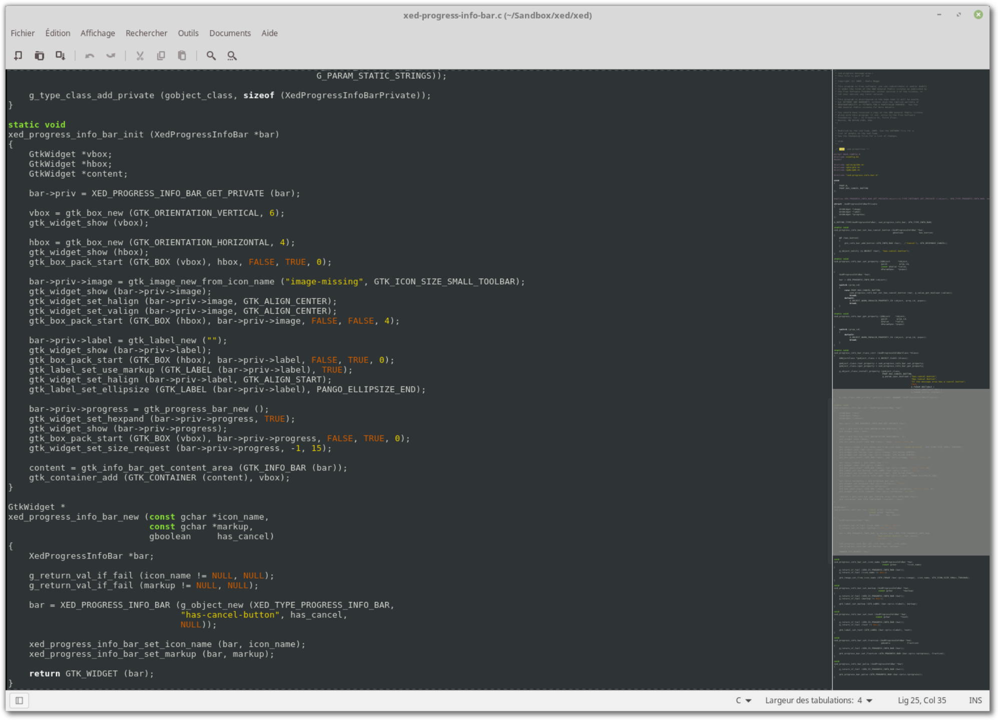
6.2 python-xapp

This is a small Python library providing extra functionality.

This project is developed in Python and its source code is available on Github.

6.3 xed

Xed is based on Pluma and acts as the default text editor.


xed-progress-info-bar.c (~/Sandbox/xed/xed)
Fichier Édition Affichage Rechercher Outils Documents Aide
C
Largeur des tabulations: 4
Lig 25, Col 35
INS

```c
                                                         G_PARAM_STATIC_STRINGS));

    g_type_class_add_private (gobject_class, sizeof (XedProgressInfoBarPrivate));
}

static void
xed_progress_info_bar_init (XedProgressInfoBar *bar)
{
    GtkWidget *vbox;
    GtkWidget *hbox;
    GtkWidget *content;

    bar->priv = XED_PROGRESS_INFO_BAR_GET_PRIVATE (bar);

    vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 6);
    gtk_widget_show (vbox);

    hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 4);
    gtk_widget_show (hbox);
    gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, TRUE, 0);

    bar->priv->image = gtk_image_new_from_icon_name ("image-missing", GTK_ICON_SIZE_SMALL_TOOLBAR);
    gtk_widget_show (bar->priv->image);
    gtk_widget_set_halign (bar->priv->image, GTK_ALIGN_CENTER);
    gtk_widget_set_valign (bar->priv->image, GTK_ALIGN_CENTER);
    gtk_box_pack_start (GTK_BOX (hbox), bar->priv->image, FALSE, FALSE, 4);

    bar->priv->label = gtk_label_new ("");
    gtk_widget_show (bar->priv->label);
    gtk_box_pack_start (GTK_BOX (hbox), bar->priv->label, FALSE, TRUE, 0);
    gtk_label_set_use_markup (GTK_LABEL (bar->priv->label), TRUE);
    gtk_widget_set_halign (bar->priv->label, GTK_ALIGN_START);
    gtk_label_set_ellipsize (GTK_LABEL (bar->priv->label), PANGO_ELLIPSIZE_END);

    bar->priv->progress = gtk_progress_bar_new ();
    gtk_widget_set_hexpand (bar->priv->progress, TRUE);
    gtk_widget_show (bar->priv->progress);
    gtk_box_pack_start (GTK_BOX (vbox), bar->priv->progress, FALSE, TRUE, 0);
    gtk_widget_set_size_request (bar->priv->progress, -1, 15);

    content = gtk_info_bar_get_content_area (GTK_INFO_BAR (bar));
    gtk_container_add (GTK_CONTAINER (content), vbox);
}

GtkWidget *
xed_progress_info_bar_new (const gchar *icon_name,
                           const gchar *markup,
                           gboolean     has_cancel)
{
    XedProgressInfoBar *bar;

    g_return_val_if_fail (icon_name != NULL, NULL);
    g_return_val_if_fail (markup != NULL, NULL);

    bar = XED_PROGRESS_INFO_BAR (g_object_new (XED_TYPE_PROGRESS_INFO_BAR,
                                 "has-cancel-button", has_cancel,
                                 NULL));

    xed_progress_info_bar_set_icon_name (bar, icon_name);
    xed_progress_info_bar_set_markup (bar, markup);

    return GTK_WIDGET (bar);
}
```

Fig. 1: Text Editor

This project is developed in C and its source code is available on Github.

6.4 xviewer

Xviewer is based on Eye of GNOME and acts as the default image viewer.

This project is developed in C and its source code is available on Github.

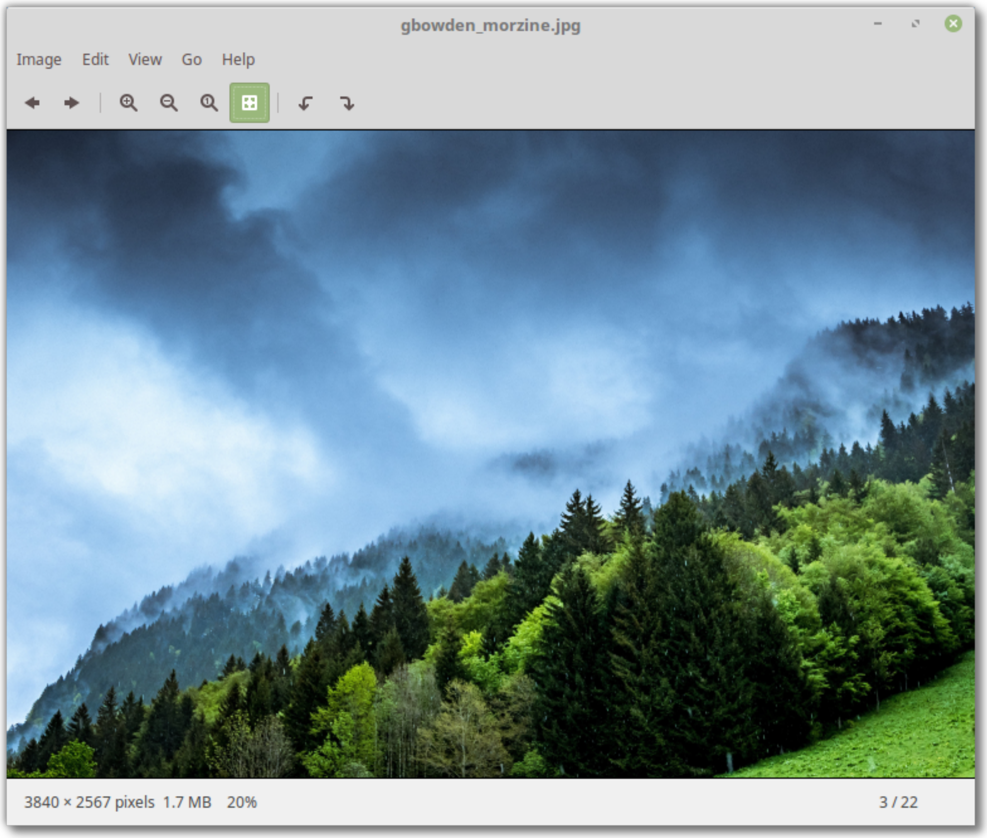


Fig. 2: Image Viewer

6.5 xplayer

Xplayer is based on Totem and acts as the default media player for music and videos.

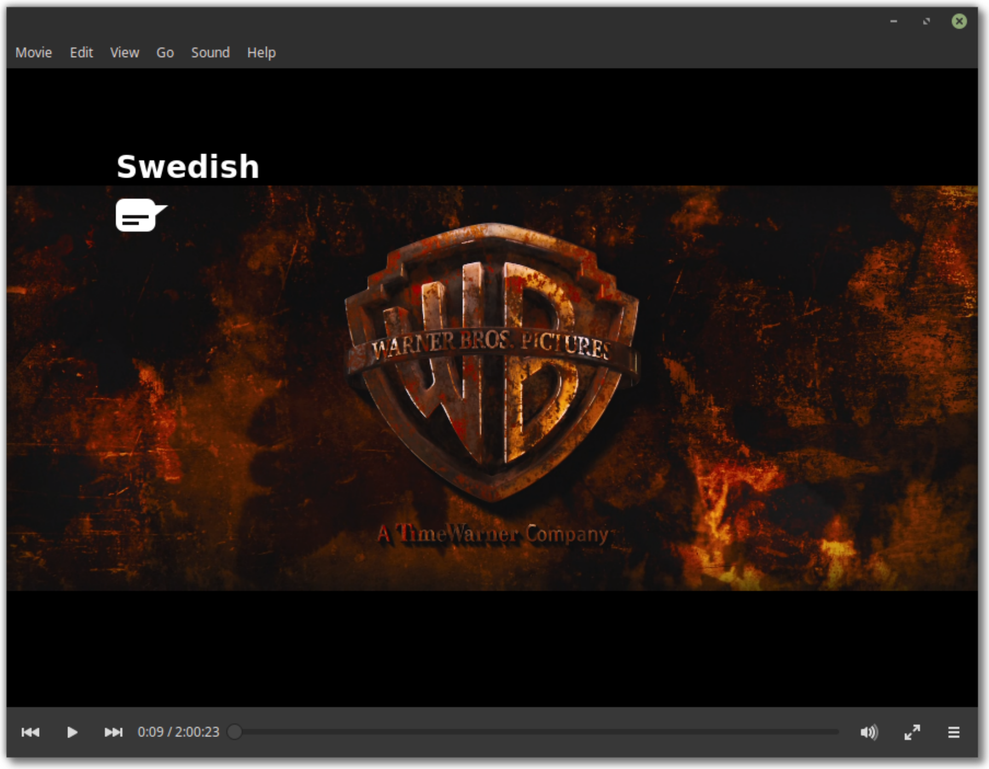


Fig. 3: Multimedia Player

This project is developed in C and its source code is available on Github.

6.6 xreader

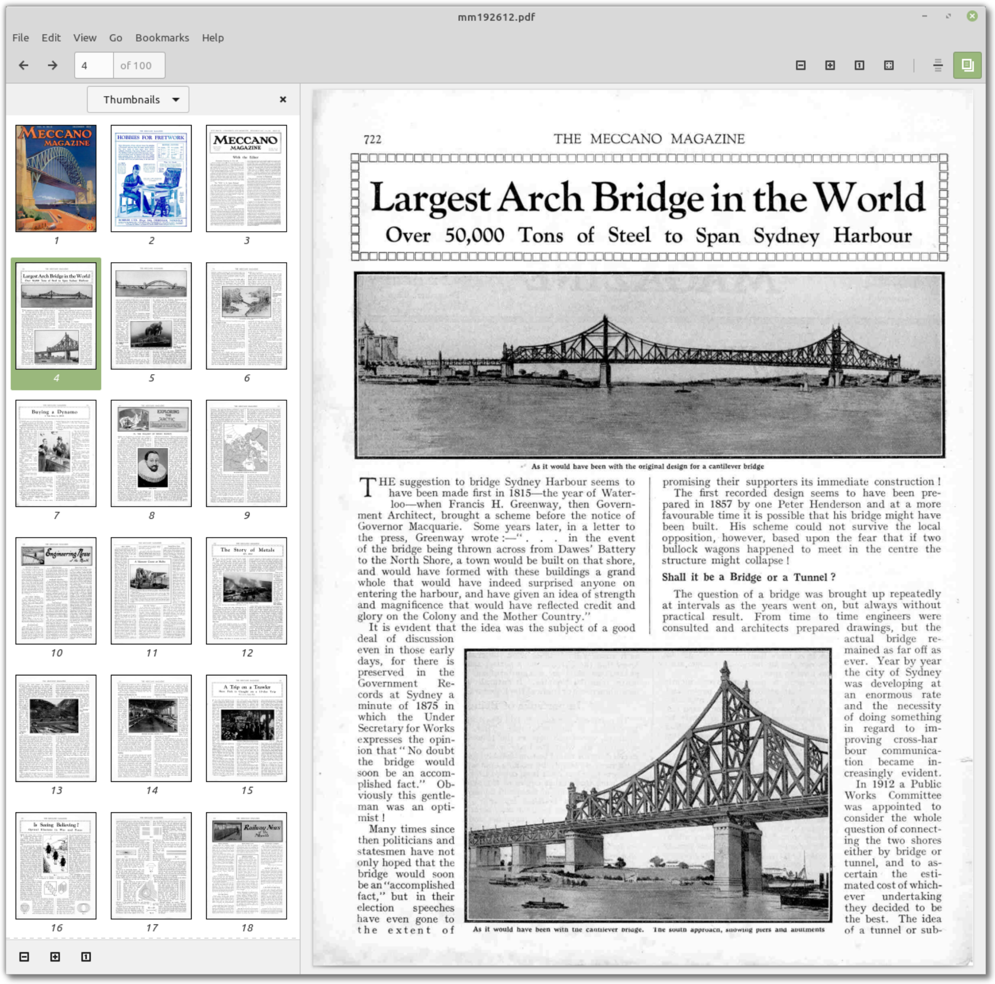
Xreader is based on Atril and acts as the default document and PDF reader.

This project is developed in C and its source code is available on Github.

6.7 pix

Pix is based on gThumb, which is an application to organize your photos.

This project is developed in C and its source code is available on Github.



722 THE MECCANO MAGAZINE

Largest Arch Bridge in the World

Over 50,000 Tons of Steel to Span Sydney Harbour

As it would have been with the original design for a cantilever bridge

THE suggestion to bridge Sydney Harbour seems to have been made first in 1815—the year of Waterloo—when Francis H. Greenway, then Government Architect, brought a scheme before the notice of Governor Macquarie. Some years later, in a letter to the press, Greenway wrote :—" . . . in the event of the bridge being thrown across from Dawes' Battery to the North Shore, a town would be built on that shore, and would have formed with these buildings a grand whole that would have indeed surprised anyone on entering the harbour, and have given an idea of strength and magnificence that would have reflected credit and glory on the Colony and the Mother Country."

It is evident that the idea was the subject of a good deal of discussion even in those early days, for there is preserved in the Government Records at Sydney a minute of 1875 in which the Under Secretary for Works expresses the opinion that " No doubt the bridge would soon be an accomplished fact." Obviously this gentleman was an optimist !

Many times since then politicians and statesmen have not only hoped that the bridge would soon be an "accomplished fact," but in their election speeches have even gone to the extent of promising their supporters its immediate construction !

The first recorded design seems to have been prepared in 1857 by one Peter Henderson and at a more favourable time it is possible that his bridge might have been built. His scheme could not survive the local opposition, however, based upon the fear that if two bullock wagons happened to meet in the centre the structure might collapse !

Shall it be a Bridge or a Tunnel ?

The question of a bridge was brought up repeatedly at intervals as the years went on, but always without practical result. From time to time engineers were consulted and architects prepared drawings, but the actual bridge remained as far off as ever. Year by year the city of Sydney was developing at an enormous rate and the necessity of doing something in regard to improving cross-harbour communication became increasingly evident.

In 1912 a Public Works Committee was appointed to consider the whole question of connecting the two shores either by bridge or tunnel, and to ascertain the estimated cost of whichever undertaking they decided to be the best. The idea of a tunnel or sub-

As it would have been with the cantilever bridge. The south approach, showing piers and abutments

Fig. 4: Document Viewer

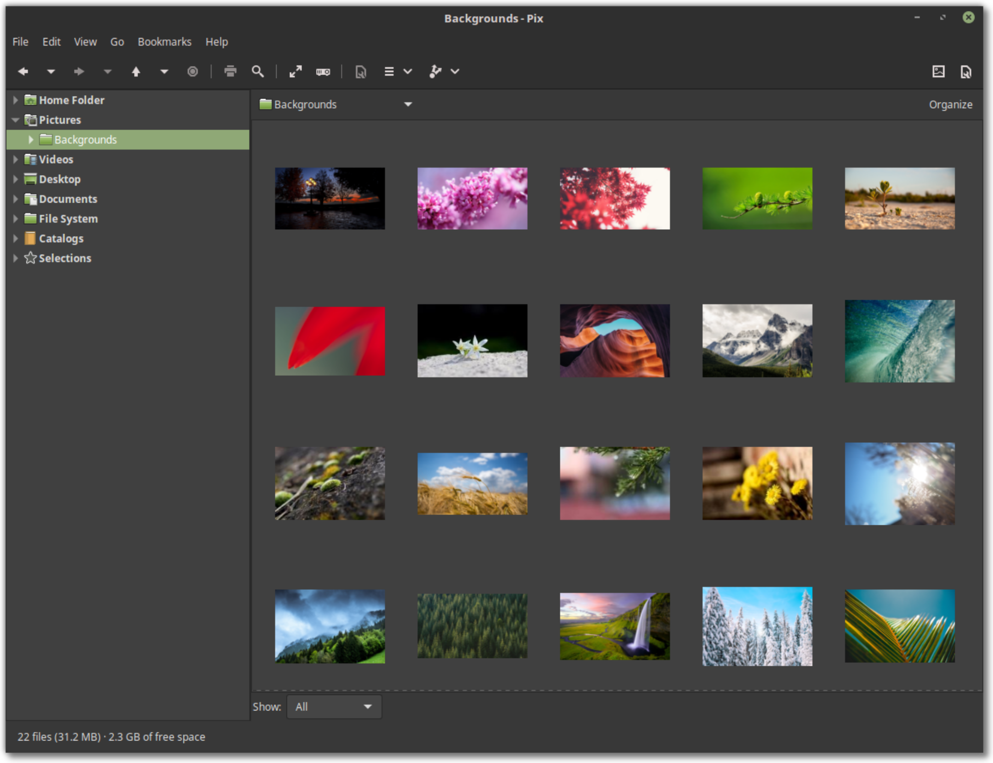


Fig. 5: Pix

6.8 blueberry

The *Bluetooth* tool, blueberry, is a frontend to gnome-bluetooth with systray support.

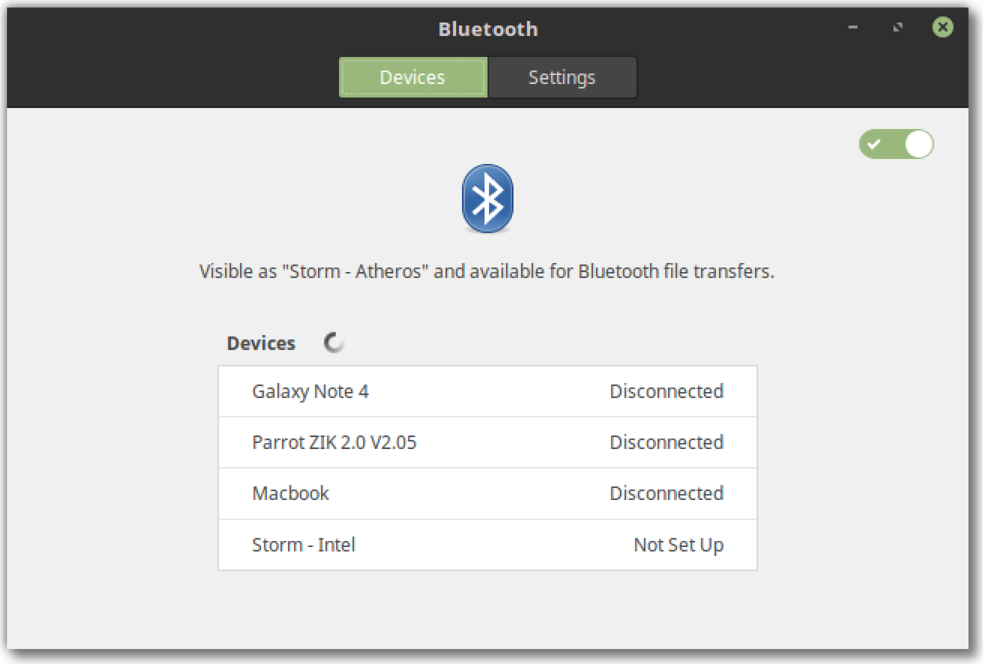
The GNOME Bluetooth frontend was removed from gnome-bluetooth and made part of gnome-control-center, essentially making gnome-bluetooth useless outside of GNOME. Blueberry provides that missing frontend and makes it easy for other GTK desktops to use gnome-bluetooth.

This project is developed in Python and its source code is available on Github.

6.9 slick-greeter

Slick-greeter is the default login screen, it's a LightDM greeter originally forked from *unity-greeter* and modified to work on its own (without gnome-settings-daemon, gnome or unity).

This project is developed in Vala and its source code is available on Github.



Visible as "Storm - Atheros" and available for Bluetooth file transfers.

Devices

Galaxy Note 4	Disconnected
Parrot ZIK 2.0 V2.05	Disconnected
Macbook	Disconnected
Storm - Intel	Not Set Up

Fig. 6: Bluetooth

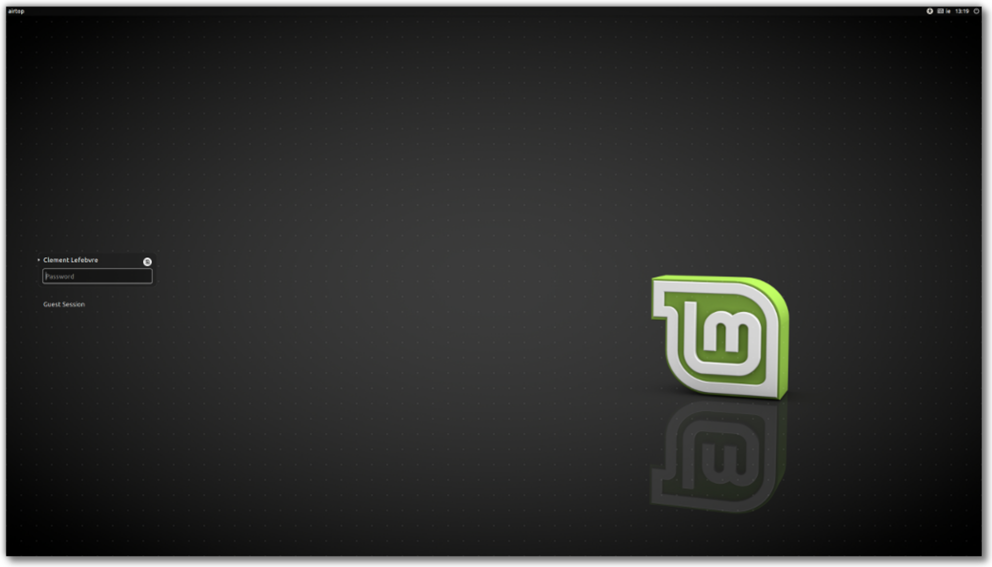
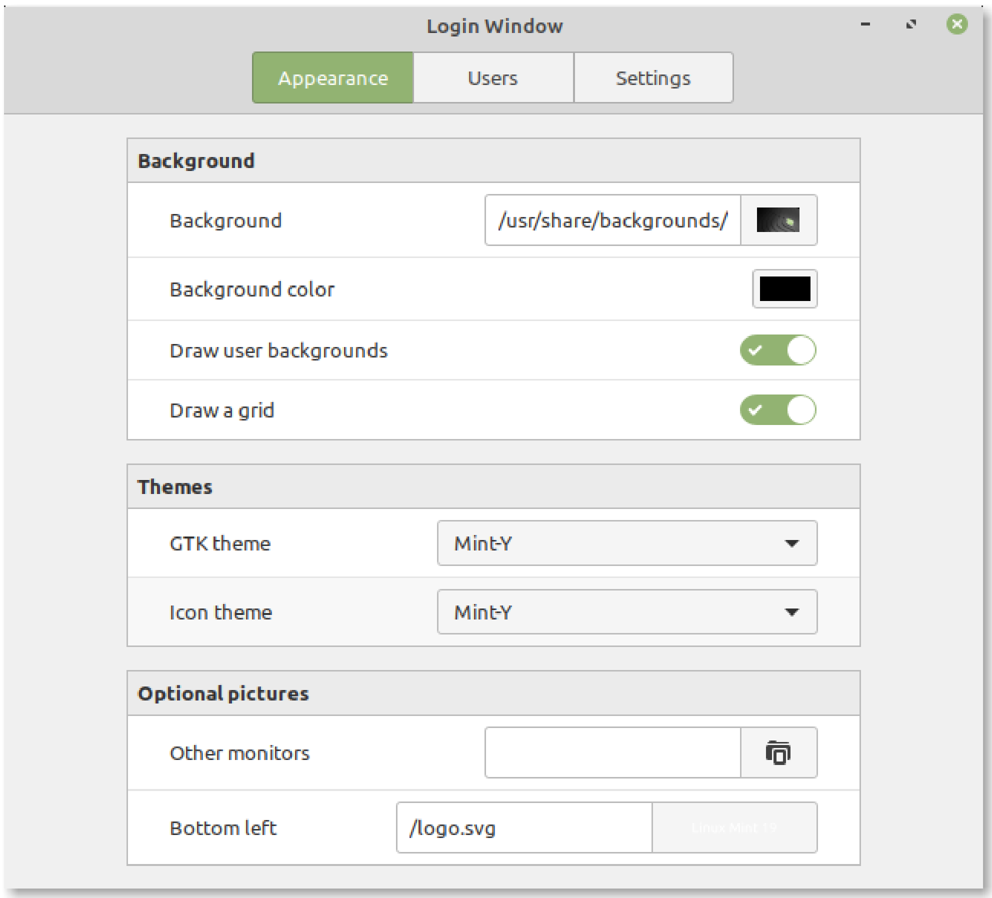


Fig. 7: Slick Greeter

6.10 lightdm-settings

The lightdm-settings project provides a configuration tool to set up LightDM and slick-greeter.



Background

Background	/usr/share/backgrounds/
Background color	■
Draw user backgrounds	✓
Draw a grid	✓

Themes

GTK theme	Mint-Y
Icon theme	Mint-Y

Optional pictures

Other monitors	
Bottom left	/logo.svg

Fig. 8: Login Window

This project is developed in Python and its source code is available on Github.

CHAPTER

SEVEN

BUILDING

Once you've installed *mint-dev-tools*, building Linux Mint projects from source is extremely easy.

7.1 Downloading the source code

Use *git clone* to get the source from github.

For instance, to get the source for mintinstall type:

```
cd ~/Sandbox
git clone https://github.com/linuxmint/mintinstall.git
```

7.2 Building a project for the first time

Use *mint-build* to build a project for the first time.

mint-build doesn't just build the project, it also fetches and installs the build dependencies (i.e. the packages which are required for the build to succeed).

To build mintinstall you would type:

```
cd ~/Sandbox/mintinstall
mint-build
```

When the build is complete, the resulting binary .deb packages are located in the parent directory (in this example in *~/Sandbox*).

7.3 Building a project

If all the build dependencies are already installed for a particular project (this is done by *mint-build* the first time you build a project), you can build faster by just calling *dpkg-buildpackage*.

To build mintinstall you would type:

```
cd ~/Sandbox/mintinstall
dpkg-buildpackage
```

7.4 Respecting the build order

If new changes in the project you're trying to build require new changes in another project you might need to build and install that other project first.

In general it's a good idea to build *mint-common* and *xapps* first.

CHAPTER

EIGHT

ROMEO

The development team uses Romeo to push new BETA/ALPHA features.

Important: Romeo contains unstable builds and recent changes pushed by the developers on github. By using it you'll upgrade from stable to unstable versions which might introduce regressions or not work at all.

8.1 Enabling Romeo

To enable Romeo on your computer, open the *Software Sources* and check the option "Unstable packages (romeo)".

8.2 Upgrading software to unstable versions

Use the *Update Manager* to refresh and upgrade your software to unstable versions.

8.3 Downgrading back to stable

To go back to stable versions, either restore a Timeshift snapshot, or disable Romeo (using the *Software Sources*), update the APT cache and use the *Maintenance -> Downgrade foreign packages* section of the *Software Sources* tool.

CHAPTER

NINE

CODING GUIDELINES

9.1 Coding Style

9.1.1 Simplicity

Prefer simple instructions over complicated ones, even if that means using more lines or duplicating code.

Avoid one-liners, complicated conditions, language specificities and abstract/generic patterns.

If your code needs to be explained, comment it.. or even better, rewrite it in a simpler way.

9.1.2 Consistency

Adopt the coding style used in the project you're contributing to.

This guarantees consistency between your new code and the existing code.

9.1.3 Indentation

In new projects or new files, do not use tabs. Use 4 space characters instead.

9.1.4 Trailing spaces

Do not leave trailing spaces in your code.

Hint: In Sublime, install the `TrailingSpaces` plugin to automatically highlight trailing spaces and give you the option to easily delete them.

9.1.5 Maximum Line length

Fit your code within 120 columns.

If a line of code is longer than 120 characters, break it into two or more lines.

Hint: In Sublime, select *View* → *Ruler* → *120* to show a ruler.

CHAPTER

TEN

CINNAMON JAVASCRIPT OPTIMIZATION TECHNIQUES

This is a temporary chapter about Javascript optimization techniques in Cinnamon. It will be part of the dev guide when the Cinnamon design is described and the content of this section will then fit into the right place.

10.1 Notes

Some of these optimization techniques don't make sense to us and we cannot explain them all, but they were tested methodically.

Jason is the only one in the team to see an impact from them. He's running a slow CPU with multi-monitors and a low-latency kernel with NVIDIA drivers. These were tested on Cinnamon 4.0. Performance boosts are witnessed in terms of input lag when moving windows and selecting text in Visual Studio Code.

These changes were tested in windowManager.js, an area of Cinnamon which is run constantly and which is prominent within the single execution thread.

10.2 Reducing the overal number of gsettings signal listeners

Here's an example: https://github.com/linuxmint/Cinnamon/commit/47bef00856e3b1f5a1e1a19e829dec498376d033

Reducing the number of listeners has a significant positive impact on performance.

10.3 Using declared functions in signal listeners

We found out that using:

```
settings.connect('changed::property', (s, k) => { this.property = s.get_int(k); });
```

was slower than:

```
settings.connect('changed::property', (s, k) => this.setProperty(s, k));
```

i.e. declaring a function and referring to that function within the callback was faster than using an anonymous block of instructions in the callback.

The impact was significant.

10.4 Factorizing callbacks

```
settings.connect('changed::int_property1', (s, k) => this.setProperty1(s, k));
settings.connect('changed::string_property2', (s, k) => this.setProperty2(s, k));
```

was slower than:

```
settings.connect('changed::int_property1', (s, k) => this.setProperty(s, k, 'int'));
settings.connect('changed::string_property2', (s, k) => this.setProperty(s, k, 'string
↪'));
```

It makes `setProperty()` slower of course, although that's usually not critical, but it makes the overal project faster.

The impact wasn't as significant and this optimization is probably only suited to critical paths such as windowmanager.js.

10.5 Grouping properties in smaller objects

Using `this.smallobject.property` is faster than `this.property`.

The idea is to avoid adding properties to large objects such as Main.wm.

So instead of using:

`Main.wm.desktop_effects_enabled`, we use `Main.wm.settings.desktop_effects_enabled`.

The impact is positive but subtle and this optimization is probably only suited to critical paths such as windowmanager.js.

10.6 Using hashmaps vs properties

Using `this.smallobject['property']` is faster than `this.smallobject.property`.

The idea is confirmed by https://www.freecodecamp.org/news/dot-notation-vs-square-brackets-javascript/.

So instead of using:

`Main.wm.desktop_effects_enabled`, we use `Main.wm.settings['desktop_effects_enabled']`.

The impact is positive but subtle and this optimization is probably only suited to critical paths such as windowmanager.js.