Troubleshooting Guide

Release

Linux Mint

Jan 27, 2018

General concepts

This guide will help you troubleshoot issues on your computers, find help within the community and report bugs to the development team.

CHAPTER 1

Observation

You're using your computer and all of a sudden you notice something wrong. Something doesn't seem to work correctly… it doesn't do what you expect it to.

Is there something wrong with the hardware or with the software? Is there something wrong in your configuration or the way you're using the computer? Are you expecting the wrong thing to happen?

You don't know yet, but one thing is sure: What you saw didn't match what you expected.

At this stage, this is an **observation**. Something looks wrong, so you start to troubleshoot.

Note:

Here's an example of an observation: "While walking, I felt a pain in my right foot".

Important: Do not report observations as bugs. Observations are not bugs, they're observations. If you think your observation might lead to finding a bug, troubleshoot to make sure.

CHAPTER 2

Expectation

Observations are mismatches between what the computer does and what you're expecting it to do.

Before troubleshooting, make sure your expectation is correct.

Read the relevant documentation, search the web for related issues, go through the release notes and if you're still unsure ask around in the community.

You cannot troubleshoot without doing this.

If your expectation is indeed correct, continue to troubleshoot.

Note:

Here's an example of a valid expectation: "My shoe should not hurt my foot".

CHAPTER 3

Reproducibility

An important step in troubleshooting is **reproducibility**.

The goal is to identify how to trigger the issue.

If you can perform a series of steps over and over again and **always** observe the same issue, then the issue is **reproducible** and you have a good chance of being able to troubleshoot it.

Note:

Here's an example of a reproducible issue: "Every time I walk, I feel a pain in my right foot".

Reproducibility is key in troubleshooting. It's very hard to analyze an issue if it's not reproducible. You're unlikely to troubleshoot an issue successfully if you don't manage to identify the steps to trigger it. When you seek help or report a bug, people will always ask you if your issue is reproducible and what are the steps to perform to reproduce it.

If the issue only happened once and isn't happening again, or if it happens sometimes but you're not sure why... you haven't identified these steps.

Important: Do not report bugs for issues you cannot reproduce. When reporting bugs, also always specify the steps to perform to reproduce the issue.

CHAPTER 4

Responsibility

An important step in troubleshooting is **responsibility**.

At this stage you've already established the following:

- Observation: The computer doesn't behave the way you expect.
- Expectation: Your expectation is correct.
- Reproducibility: You're able to reproduce the issue and you have identified the steps to do that.

The goal now is to identify **which component is responsible** for the issue. The best way to find this out is to proceed by elimination.

During the **reproducibility** phase you identified a series of steps necessary to reproduce the issue. You should now try to reduce the number of steps and remove components which might play a role in the issue in order to identify which component actually does.

Here are some examples:

- An application crashes when you run it in Russian... does it also crash in English?
- The computer is slow to boot... is it slow when offline? are there USB devices plugged in?
- The desktop freezes... does it happen with the default configuration? or without 3rd party applets?

Another strategy to identify which component is responsible is to try and reproduce the issue in different environments.

Here are some examples:

- You see an issue in Cinnamon, does it also happen in MATE?
- You see an issue in a particular release, does it also happen in previous releases?
- You see an issue in Linux Mint, does it also happen in Ubuntu?

By identifying what's common and what's not, you have a better chance of identifying where the issue lies.

Note:

Here's an example of an issue where the responsible component was identified: "My Nike shoes hurt my right foot every time I walk. I feel no pain with other pairs of shoes, the same socks, the same places, the same amount of time."

At this stage, you still don't know enough to understand the **cause** of the issue, but you're able to successfully identify the component responsible for the issue.

Important: Linux Mint is made up of thousands of software components, many of them developed by very different teams. Identifying the right component is key to knowing which development team might be able to help and where to report an issue.

CHAPTER 5

Change

When something worked **before** and no longer works **anymore**, an important aspect to troubleshoot is **change**.

What changed recently? Did you install something new? Did you apply updates? Did you change some settings? Are you connecting from a different place?

Reviewing the differences in context between when things worked and when things no longer do is key and can help you identify **responsibility** (i.e. the component responsible for the issue) very rapidly.

Note:

Here's an example of an issue where change is key: "My Nike shoes hurt my right foot. They were fine until I put them in the washing machine."

CHAPTER 6

Errors

In many cases your computer detects the issue and reports an error message.

Always read error messages and try to understand them. They often explain the cause of the issue and sometimes even suggest workarounds or solutions for it.

6.1 System logs

System logs often contain clues for crashes, hardware, driver and networking issues.

You can use the following to go through system logs:

- The System Logs tool (from the application menu)
- The `dmesg` command
- The `journalctl` command
- The `/var/log/syslog` file

6.2 Session Errors

If you can't log in or if the issue is related to your desktop environment, check the `~/.xsession-errors` file, and log files from the `/var/log/lightdm/` directory.

6.3 Application output

If an application isn't working correctly, close it and run it from the terminal. It might output error messages in the terminal which will give you clues about the cause of the issue.

6.4 Crashes

After a crash, you can launch *Menu* → *System Reports* to see information about the crash and get a stack trace for it. This information is valuable for developers and helps tremendously when trying to understand causes of a crash.

Important: When reporting bugs, always look for errors and always report them.

CHAPTER 7

Environment

Your settings, your locale, the set of applications you've installed and everything that might make things on your computer unique, or somewhat different than on somebody else's is called your **environment**.

If the issue is always reproducible on your computer but the very same steps do not trigger it on other computers, then the environment is key and the differences between the environment of the two computers should be reviewed.

Important: When other people can't reproduce your issue with the steps you provided, give them more information about your environment.

Depending on the nature of the issue, you might look at different parts of the environment.

7.1 Locale

Your locale consists in your language and regional settings.

Here are a few common issues related to locales:

- Crashes due to a bad translation
- Missing translations
- Crashes due to unicode characters
- Wide widgets or applications due to long translations
- Failure to show special characters

When an issue is specific to your locale, set your language to English, log out and log back in and see if the issue is still happening.

To see or share details about your locale, open a terminal and type `locale`.

7.2 List of packages

An issue might happen because of a missing package or a missing library.

You can use the terminal to search. For instance, to search for the foobar library, type `apt search foobar`.

To see or share the list of installed packages on your system, type `dpkg -l`.

You can also check the version of a particular package. To check the installed and available versions of cinnamon for instance, you can type `apt policy cinnamon`.

Last but not least, you can list your repositories with `inxi -r`.

CHAPTER 8

Why

The reason why bugs are reported might sound obvious, but it's very important to properly understand it.

Bug reports are only valuable when they lead to a bug fix or an improvement. Only a small portion of open bug reports do. For software to improve quickly, development teams need to identify and work on bug reports which have the best chance of leading to a fix.

Ideally, the development team should be aware of all open bug reports and work on the ones which have the greatest chance of improving the quality of the software. In open-source though, where teams are small and communities are huge, the number of open bug reports is usually very big and teams cannot read, process or be aware of all the open bug reports. This is ok on small projects where there are 10 reports or so, but certainly not on large ones with thousands of open bug reports.

The author of a bug report suffers the issue and considers it to be real, whether or not the development team is able to reproduce it and no matter what the team thinks of it. When the bug report is closed, the author therefore might find it hard to accept. There can be feelings of frustration, legitimacy, ingratitude and ignorance at play. It's really important to set aside any personal feelings when it gets to that and to focus on being efficient.

When you report a bug, both you and the development team are focused on one thing and one thing only: Improving the software. Developers already know you enjoy their work, and you already know they value your feedback. The question isn't whether your issue is real or not, or how people feel, it's whether or not this bug report can be turned quickly into a successful bug fix or not. If it can't, it should be closed.

Important: The reason we report bugs is not to document or catalog anything that is (or might be) wrong with the software, it's to improve it. Bug reports which don't lead to successful bug fixes get in the way and prevent developers from quickly identifying bug reports which do. The goal of a bug report is to improve the software. When submitting a bug report, don't ask yourself whether it's legitimate, ask yourself if it's pertinent, simple and complete enough to lead to a bug fix.

CHAPTER 9

When

Do not open bug reports before troubleshooting is finished. A particular project might have 2 developers, 100 open bug reports, 1,000 active users on the forums and IRC, and a million users overall. You should troubleshoot the issue on your own first, as much as you can, and then rely on other users in the community for help, before interacting with the development team.

You should report a bug when all these conditions are met:

- You're absolutely sure it's a bug (you've established reproducibility, responsibility and identified the cause).
- You've gathered enough information for the team to work on (error messages, steps to reproduce, etc.), and you're confident it will quickly lead to a fix.
- The issue hasn't already been reported.

Important: Although developers can help with questions, troubleshooting and analysis, there are very few of them and many other bug reports open. Use them for what they're best at: Fixing things. You should only open bug reports when everything is done already (observation, troubleshooting, analysis, gathering of information, understanding of the cause) and the only thing missing is the actual fix. Don't rely on developers to do the analysis/troubleshooting for you. A bug you don't understand is unlikely to be a bug they'll fix overnight.

CHAPTER 10

What

It's hard to generalize and there will always be special cases.

Ideally **what** you should report are actual bugs, i.e. issues which are deterministic, understood, analyzed and thus fixable.

Note: Do not report observations, they are rarely useful to developers, and when they are they usually require a large amount of time and efforts to be turned into improvements.

CHAPTER 11

Where

The Linux Mint operating system is made up of thousands of software components. Once you've identified the component responsible for your issue, you need to identify who's maintaining it.

11.1 Mint and upstream components

There are two types of components within Linux Mint:

- Mint components are maintained or patched by Linux Mint.
- Upstream components are maintained by Ubuntu (or Debian in the case of LMDE).

You can use the terminal to get information about a particular component.

Use `apt contains` to find the name of the package containing a particular file.

For instance, if you are chasing an issue in the file `/usr/share/polkit-1/actions/org.gnome.gnome-system-monitor.policy`, you can type:

```
apt contains /usr/share/polkit-1/actions/org.gnome.gnome-system-monitor.policy

gnome-system-monitor: /usr/share/polkit-1/actions/org.gnome.gnome-system-monitor.
↪policy
```

The output tells you that this file is part of the `gnome-system-monitor` package.

Once you know the name of the package, you can use `apt policy` to see who's maintaining it:

```
apt policy gnome-system-monitor

gnome-system-monitor:
  Installed: 3.18.2-1ubuntu1
  Candidate: 3.18.2-1ubuntu1
  Version table:
 *** 3.18.2-1ubuntu1 500
        500 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages
```

```
        100 /var/lib/dpkg/status
     3.18.2-1 500
        500 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages
```

As you can see above, gnome-system-monitor comes from the Ubuntu repositories and is therefore an upstream component. Bugs related to it should be reported to Ubuntu.

Here's another example:

```
apt policy gnome-calculator

gnome-calculator:
  Installed: 1:3.18.3-0ubuntu1.16.04.1.mint1
  Candidate: 1:3.18.3-0ubuntu1.16.04.1.mint1
  Version table:
 *** 1:3.18.3-0ubuntu1.16.04.1.mint1 700
        700 http://packages.linuxmint.com sylvia/upstream amd64 Packages
        100 /var/lib/dpkg/status
     1:3.18.3-0ubuntu1.16.04.1 500
        500 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages
     1:3.18.3-0ubuntu1 500
        500 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages
```

This time we're looking at gnome-calculator. As you can see above, Linux Mint provides version 1:3.18.3-0ubuntu1.16.04.1.mint1, and that's the version which is installed. In this example, our version of gnome-calculator is therefore a Mint component and bugs related to it should be reported to Linux Mint.

11.2 Upstream bugs: Launchpad and Debbugs

If the bug relates to an Ubuntu component, it should be reported on Launchpad.

If the bug relates to a Debian component, it should be reported on Debian's bug tracking system.

Important: Before reporting upstream bugs to Ubuntu and/or Debian, you should always make sure the issue can be reproduced in Ubuntu and/or Debian. Install Ubuntu (or Debian) in a virtual machine and try to reproduce the issue. If the issue is specific to Linux Mint, it should be reported to Linux Mint directly (whether the component is upstream or not).

The base used by LMDE 3 is Debian 9 'Stretch'. The base used by Linux Mint 18 is Ubuntu 16.04 'Xenial'. The base used by Linux Mint 19 is Ubuntu 18.04 'Bionic'.

11.3 MATE bugs: Github/mate-desktop

MATE is co-developed by Linux Mint.

Bugs which affect MATE or one of its components should be reported on the MATE Github site.

One exception to this are bugs which are specific to Linux Mint. For instance, if a bug relates to mintmenu, mintdesktop or a mint theme in MATE, please report it to Linux Mint directly.

11.4 Cinnamon, X-Apps and Linux Mint bugs

Linux Mint has three development teams:

- The Cinnamon development team maintains all Cinnamon components, including nemo and muffin.
- The X-App development teams maintains all cross-distribution projects such as the X-App applications (pix, xed, xreader, xplayer, xviewer), libraries, and slick-greeter, blueberry, etc.
- The Linux Mint development team maintains all the Mint tools and other components distributed via the Mint repositories.

When reporting a bug to one of these teams, try to find the compoment on the Linux Mint Github site.

For instance, a nemo bug should be reported on Nemo, a mintmenu bug should be reported on Mintmenu, an xplayer bug on Xplayer, etc.

If you want to report a general issue about Cinnamon, you can use Cinnamon.

If you want to report a general issue about Linux Mint, an issue about an upstream component which is patched by Linux Mint, or an issue about an upstream component which is specific to Linux Mint, you can use Linux Mint.

CHAPTER 12

How

12.1 Provide precise and relevant information

The first thing developers try to do is understand how to reproduce the issue.

Always give them the relevant information:

- Steps you perform to always reproduce the issue on your computer (be as precise as possible).
- Particularities in your environment which might be relevant or different than the developers' environment (your Mint edition and release, your locale, your drivers, etc..).
- Any relevant error messages or software output.
- Screenshots which show the issue.
- Anything that might make it easier for developers to understand and/or reproduce the issue.

Note: A lot of information about the environment can be listed with the command `inxi -Fxxrzc0`.

12.2 Provide stack traces for crashes

When reporting a crash, provide the following:

- The `dmesg` line for the crash.
- A stack trace for the crash (you can get one using *Menu* → *System Reports*).
- As for other bugs, steps to reproduce the issue.

12.3 Explain your expectation

It might sound silly, but sometimes a bug report is clear on what the software does but not on what the author expects it to do. This can lead to a misunderstanding.

If possible, briefly explain why you think the software behavior is wrong and what you would expect it to do instead.

12.4 Be patient and pleasant

Most people are. It's easy to be grumpy (or sometimes just to "sound" grumpy) on the Internet. Developers and users alike should do their best to make the interaction as pleasant as possible.

This is open-source after all, we're all in it for the fun (even though, let's face it.. bugs aren't exactly the funniest aspect of it).

12.5 Feel like a hero

When people see their reports closed or no answers to their comments, they might assume they wasted their time. That assumption is completely wrong. There are very few developers and many many users. It is not possible to interact with everybody and to fix some bug reports without closing many. Linux Mint is getting better and better though, every day, thanks to people like us, and people like you. It is fueled by effort and feedback. If you have read until here and/or you have contributed feedback or bug reports, you should feel like a hero. It takes time to pat each other on the back and we surely don't do it enough.

CHAPTER 13

Can ideas or feature requests be submitted?

Yes, absolutely. Github issues aren't necessarily bug reports. They can also be ideas or feature requests.

When submitting an idea or a feature request, try to indicate it in the issue title, so that it's immediately clear to developers they're not looking at a bug report.

CHAPTER 14

Why are ideas and feature requests closed?

In the past, issues for "good" ideas (this is highly subjective) used to remain open until the idea was implemented, while "bad" ideas (or at least ideas which weren't deemed urgent, feasible or important enough) were closed.

This was a mistake, because it focused on legitimacy and perception and it tried to give issue authors something we could not afford: traceability and reporting of their ideas' implementation.

The process is much simpler now and much more efficient. When you submit an idea, it is processed and then closed. It might lead to an implementation, it might lead to a design discussion within the team, it might be planned for a particular release and put on the roadmap, it might lead to nothing at all or stick in people's mind as something they could improve later on. Either way, your role is done when the idea is reported and once it's understood by the development team.

When the process of gathering information about your idea and understanding it is finished, your issue can be closed. The eventual and possible implementation of that idea is separate and shouldn't keep the issue open.