Reporting regressions

"*We don't cause regressions*" is the first rule of Linux kernel development; Linux founder and lead developer Linus Torvalds established it himself and ensures it's obeyed.

This document describes what the rule means for users and how the Linux kernel's development model ensures to address all reported regressions; aspects relevant for kernel developers are left to Documentation/process/handling-regressions.rst.

The important bits (aka "TL;DR")

1. It's a regression if something running fine with one Linux kernel works worse or not at all with a newer version. Note, the newer kernel has to be compiled using a similar configuration; the detailed explanations below describes this and other fine print in more detail.
2. Report your issue as outlined in Documentation/admin-guide/reporting-issues.rst, it already covers all aspects important for regressions and repeated below for convenience. Two of them are important: start your report's subject with "[REGRESSION]" and CC or forward it to the regression mailing list (regressions@lists.linux.dev).
3. Optional, but recommended: when sending or forwarding your report, make the Linux kernel regression tracking bot "regzbot" track the issue by specifying when the regression started like this:

   ```
   #regzbot introduced v5.13..v5.14-rc1
   ```

All the details on Linux kernel regressions relevant for users

The important basics

What is a "regression" and what is the "no regressions rule"?

It's a regression if some application or practical use case running fine with one Linux kernel works worse or not at all with a newer version compiled using a similar configuration. The "no regressions rule" forbids this to take place; if it happens by accident, developers that caused it are expected to quickly fix the issue.

It thus is a regression when a WiFi driver from Linux 5.13 works fine, but with 5.14 doesn't work at all, works significantly slower, or misbehaves somehow. It's also a regression if a perfectly working application suddenly shows erratic behavior with a newer kernel version; such issues can be caused by changes in procfs, sysfs, or one of the many other interfaces Linux provides to userland software. But keep in mind, as mentioned earlier: 5.14 in this example needs to be built from a configuration similar to the one from 5.13. This can be achieved using `make olddefconfig`, as explained in more detail below.

Note the "practical use case" in the first sentence of this section: developers despite the "no regressions" rule are free to change any aspect of the kernel and even APIs or ABIs to userland, as long as no existing application or use case breaks.

Also be aware the "no regressions" rule covers only interfaces the kernel provides to the userland. It thus does not apply to kernel-internal interfaces like the module API, which some externally developed drivers use to hook into the kernel.

How do I report a regression?

Just report the issue as outlined in Documentation/admin-guide/reporting-issues.rst, it already describes the important points. The following aspects outlined there are especially relevant for regressions:

- When checking for existing reports to join, also search the archives of the Linux regressions mailing list and regzbot's web-interface.
- Start your report's subject with "[REGRESSION]".
- In your report, clearly mention the last kernel version that worked fine and the first broken one. Ideally try to find the exact change causing the regression using a bisection, as explained below in more detail.
- Remember to let the Linux regressions mailing list (regressions@lists.linux.dev) know about your report:
  - If you report the regression by mail, CC the regressions list.
  - If you report your regression to some bug tracker, forward the submitted report by mail to the regressions list while CCing the maintainer and the mailing list for the subsystem in question.

  If it's a regression within a stable or longterm series (e.g. v5.15.3..v5.15.5), remember to CC the Linux stable mailing list (stable@vger.kernel.org).

  In case you performed a successful bisection, add everyone to the CC the culprit's commit message mentions in lines starting with "Signed-off-by:".

When CCing for forwarding your report to the list, consider directly telling the aforementioned Linux kernel regression tracking bot

about your report. To do that, include a paragraph like this in your mail:

```
#regzbot introduced: v5.13..v5.14-rc1
```

Regzbot will then consider your mail a report for a regression introduced in the specified version range. In above case Linux v5.13 still worked fine and Linux v5.14-rc1 was the first version where you encountered the issue. If you performed a bisection to find the commit that caused the regression, specify the culprit's commit-id instead:

```
#regzbot introduced: 1f2e3d4c5d
```

Placing such a "regzbot command" is in your interest, as it will ensure the report won't fall through the cracks unnoticed. If you omit this, the Linux kernel's regressions tracker will take care of telling regzbot about your regression, as long as you send a copy to the regressions mailing lists. But the regression tracker is just one human which sometimes has to rest or occasionally might even enjoy some time away from computers (as crazy as that might sound). Relying on this person thus will result in an unnecessary delay before the regressions becomes mentioned on the list of tracked and unresolved Linux kernel regressions and the weekly regression reports sent by regzbot. Such delays can result in Linus Torvalds being unaware of important regressions when deciding between "continue development or call this finished and release the final?".

Are really all regressions fixed?

Nearly all of them are, as long as the change causing the regression (the "culprit commit") is reliably identified. Some regressions can be fixed without this, but often it's required.

Who needs to find the root cause of a regression?

Developers of the affected code area should try to locate the culprit on their own. But for them that's often impossible to do with reasonable effort, as quite a lot of issues only occur in a particular environment outside the developer's reach -- for example, a specific hardware platform, firmware, Linux distro, system's configuration, or application. That's why in the end it's often up to the reporter to locate the culprit commit; sometimes users might even need to run additional tests afterwards to pinpoint the exact root cause. Developers should offer advice and reasonably help where they can, to make this process relatively easy and achievable for typical users.

How can I find the culprit?

Perform a bisection, as roughly outlined in Documentation/admin-guide/reporting-issues.rst and described in more detail by Documentation/admin-guide/bug-bisect.rst. It might sound like a lot of work, but in many cases finds the culprit relatively quickly. If it's hard or time-consuming to reliably reproduce the issue, consider teaming up with other affected users to narrow down the search range together.

Who can I ask for advice when it comes to regressions?

Send a mail to the regressions mailing list (regressions@lists.linux.dev) while CCing the Linux kernel's regression tracker (regressions@leemhuis.info); if the issue might better be dealt with in private, feel free to omit the list.

Additional details about regressions

What is the goal of the "no regressions rule"?

Users should feel safe when updating kernel versions and not have to worry something might break. This is in the interest of the kernel developers to make updating attractive: they don't want users to stay on stable or longterm Linux series that are either abandoned or more than one and a half years old. That's in everybody's interest, as those series might have known bugs, security issues, or other problematic aspects already fixed in later versions. Additionally, the kernel developers want to make it simple and appealing for users to test the latest pre-release or regular release. That's also in everybody's interest, as it's a lot easier to track down and fix problems, if they are reported shortly after being introduced.

Is the "no regressions" rule really adhered in practice?

It's taken really seriously, as can be seen by many mailing list posts from Linux creator and lead developer Linus Torvalds, some of which are quoted in Documentation/process/handling-regressions.rst.

Exceptions to this rule are extremely rare; in the past developers almost always turned out to be wrong when they assumed a particular situation was warranting an exception.

Who ensures the "no regressions" is actually followed?

The subsystem maintainers should take care of that, which are watched and supported by the tree maintainers -- e.g. Linus Torvalds for mainline and Greg Kroah-Hartman et al. for various stable/longterm series.

All of them are helped by people trying to ensure no regression report falls through the cracks. One of them is Thorsten Leemhuis, who's currently acting as the Linux kernel's "regressions tracker"; to facilitate this work he relies on regzbot, the Linux kernel regression tracking bot. That's why you want to bring your report on the radar of these people by CCing or forwarding each report to the regressions mailing list, ideally with a "regzbot command" in your mail to get it tracked immediately.

How quickly are regressions normally fixed?

Developers should fix any reported regression as quickly as possible, to provide affected users with a solution in a timely manner and prevent more users from running into the issue; nevertheless developers need to take enough time and care to ensure regression fixes do not cause additional damage.

The answer thus depends on various factors like the impact of a regression, its age, or the Linux series in which it occurs. In the end though, most regressions should be fixed within two weeks.

Is it a regression, if the issue can be avoided by updating some software?

Almost always: yes. If a developer tells you otherwise, ask the regression tracker for advice as outlined above.

Is it a regression, if a newer kernel works slower or consumes more energy?

Yes, but the difference has to be significant. A five percent slow-down in a micro-benchmark thus is unlikely to qualify as regression, unless it also influences the results of a broad benchmark by more than one percent. If in doubt, ask for advice.

Is it a regression, if an external kernel module breaks when updating Linux?

No, as the "no regression" rule is about interfaces and services the Linux kernel provides to the userland. It thus does not cover building or running externally developed kernel modules, as they run in kernel-space and hook into the kernel using internal interfaces occasionally changed.

How are regressions handled that are caused by security fixes?

In extremely rare situations security issues can't be fixed without causing regressions; those fixes are given way, as they are the lesser evil in the end. Luckily this middling almost always can be avoided, as key developers for the affected area and often Linus Torvalds himself try very hard to fix security issues without causing regressions.

If you nevertheless face such a case, check the mailing list archives if people tried their best to avoid the regression. If not, report it; if in doubt, ask for advice as outlined above.

What happens if fixing a regression is impossible without causing another?

Sadly these things happen, but luckily not very often; if they occur, expert developers of the affected code area should look into the issue to find a fix that avoids regressions or at least their impact. If you run into such a situation, do what was outlined already for regressions caused by security fixes: check earlier discussions if people already tried their best and ask for advice if in doubt.

A quick note while at it: these situations could be avoided, if people would regularly give mainline pre-releases (say v5.15-rc1 or -rc3) from each development cycle a test run. This is best explained by imagining a change integrated between Linux v5.14 and v5.15-rc1 which causes a regression, but at the same time is a hard requirement for some other improvement applied for 5.15-rc1. All these changes often can simply be reverted and the regression thus solved, if someone finds and reports it before 5.15 is released. A few days or weeks later this solution can become impossible, as some software might have started to rely on aspects introduced by one of the follow-up changes: reverting all changes would then cause a regression for users of said software and thus is out of the question.

Is it a regression, if some feature I relied on was removed months ago?

It is, but often it's hard to fix such regressions due to the aspects outlined in the previous section. It hence needs to be dealt with on a case-by-case basis. This is another reason why it's in everybody's interest to regularly test mainline pre-releases.

Does the "no regression" rule apply if I seem to be the only affected person?

It does, but only for practical usage: the Linux developers want to be free to remove support for hardware only to be found in attics and museums anymore.

Note, sometimes regressions can't be avoided to make progress -- and the latter is needed to prevent Linux from stagnation. Hence, if only very few users seem to be affected by a regression, it for the greater good might be in their and everyone else's interest to lettings things pass. Especially if there is an easy way to circumvent the regression somehow, for example by updating some software or using a kernel parameter created just for this purpose.

Does the regression rule apply for code in the staging tree as well?

Not according to the help text for the configuration option covering all staging code, which since its early days states:

```
Please note that these drivers are under heavy development, may or
may not work, and may contain userspace interfaces that most likely
will be changed in the near future.
```

The staging developers nevertheless often adhere to the "no regressions" rule, but sometimes bend it to make progress. That's for example why some users had to deal with (often negligible) regressions when a WiFi driver from the staging tree was replaced by a totally different one written from scratch.

Why do later versions have to be "compiled with a similar configuration"?

Because the Linux kernel developers sometimes integrate changes known to cause regressions, but make them optional and disable them in the kernel's default configuration. This trick allows progress, as the "no regressions" rule otherwise would lead to stagnation.

Consider for example a new security feature blocking access to some kernel interfaces often abused by malware, which at the same time are required to run a few rarely used applications. The outlined approach makes both camps happy: people using these applications can leave the new security feature off, while everyone else can enable it without running into trouble.

How to create a configuration similar to the one of an older kernel?

Start your machine with a known-good kernel and configure the newer Linux version with `make olddefconfig`. This makes the kernel's build scripts pick up the configuration file (the ".config" file) from the running kernel as base for the new one you are about to compile; afterwards they set all new configuration options to their default value, which should disable new features that might cause regressions.

Can I report a regression I found with pre-compiled vanilla kernels?

You need to ensure the newer kernel was compiled with a similar configuration file as the older one (see above), as those that built them might have enabled some known-to-be incompatible feature for the newer kernel. If in doubt, report the matter to the kernel's provider and ask for advice.

More about regression tracking with "regzbot"

What is regression tracking and why should I care about it?

Rules like "no regressions" need someone to ensure they are followed, otherwise they are broken either accidentally or on purpose. History has shown this to be true for Linux kernel development as well. That's why Thorsten Leemhuis, the Linux Kernel's regression tracker, and some people try to ensure all regression are fixed by keeping an eye on them until they are resolved. Neither of them are paid for this, that's why the work is done on a best effort basis.

Why and how are Linux kernel regressions tracked using a bot?

Tracking regressions completely manually has proven to be quite hard due to the distributed and loosely structured nature of Linux kernel development process. That's why the Linux kernel's regression tracker developed regzbot to facilitate the work, with the long term goal to automate regression tracking as much as possible for everyone involved.

Regzbot works by watching for replies to reports of tracked regressions. Additionally, it's looking out for posted or committed patches referencing such reports with "Link:" tags; replies to such patch postings are tracked as well. Combined this data provides good insights into the current state of the fixing process.

How to see which regressions regzbot tracks currently?

Check out regzbot's web-interface.

What kind of issues are supposed to be tracked by regzbot?

The bot is meant to track regressions, hence please don't involve regzbot for regular issues. But it's okay for the Linux kernel's regression tracker if you involve regzbot to track severe issues, like reports about hangs, corrupted data, or internal errors (Panic, Oops, BUG(), warning, ...).

How to change aspects of a tracked regression?

By using a 'regzbot command' in a direct or indirect reply to the mail with the report. The easiest way to do that: find the report in your "Sent" folder or the mailing list archive and reply to it using your mailer's "Reply-all" function. In that mail, use one of the following commands in a stand-alone paragraph (IOW: use blank lines to separate one or multiple of these commands from the rest of the mail's text).

- Update when the regression started to happen, for example after performing a bisection:

  ```
  #regzbot introduced: 1f2e3d4c5d
  ```

- Set or update the title:

  ```
  #regzbot title: foo
  ```

- Monitor a discussion or bugzilla.kernel.org ticket where additions aspects of the issue or a fix are discussed::

  ```
  #regzbot monitor: https://lore.kernel.org/r/30th.anniversary.repost@klaava.Helsinki.FI/
  #regzbot monitor: https://bugzilla.kernel.org/show_bug.cgi?id=123456789
  ```

- Point to a place with further details of interest, like a mailing list post or a ticket in a bug tracker that are slightly related, but about a different topic:

```
#regzbot link: https://bugzilla.kernel.org/show_bug.cgi?id=123456789
```

- Mark a regression as invalid:

```
#regzbot invalid: wasn't a regression, problem has always existed
```

Regzbot supports a few other commands primarily used by developers or people tracking regressions. They and more details about the aforementioned regzbot commands can be found in the getting started guide and the reference documentation for regzbot.