# Contributing Guidelines

<!-- vim-markdown-toc GFM -->

\* [Example Law: The Law of Leaky Abstractions](#example-law-the-law-of-leaky-abstractions)

\* [Localisation](#localisation)

<!-- vim-markdown-toc -->

The goal of this project is to have a set of \_concise\_ definitions to laws, principles, methodologies and patterns which hackers will find useful. They should be:

1. Short - one or two paragraphs.

2. Include the original source.

3. Quote the law if possible, with the author's name.

4. Link to related laws in the 'See also' section.

5. Include real-world examples if possible in the 'Real-world examples' section.

Some other tips:

- It is fine to include laws which are humorous or not serious.

- If a law does not obviously apply to development or coding, include a paragraph explaining the relevance to technologists.

- Don't worry about managing the table of contents, I can generate it.

- Feel free to include images, but aim to keep it down to one image per law.

- Be careful not to copy-and-paste content (unless it is explicitly quoted), as it might violate copyright.

- Include hyperlinks to referenced material.

- Do not advocate for the law, or aim to be opinionated on the correctness or incorrectness of the law, as this repository is simply the descriptions and links.

- Avoid 'you' when writing. For example, prefer "This law suggests refactoring should be avoided when..." rather than "you should avoid refactoring when...". This keeps the style slightly more formal and avoids seeming like advocation of a law.

An example law is shown below, which covers most of the key points:

---

### Example Law: The Law of Leaky Abstractions

[The Law of Leaky Abstractions on Joel on Software](https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/)

> All non-trivial abstractions, to some degree, are leaky.

>

> (Joel Spolsky)

This law states that abstractions, which are generally used in computing to simplify working with complicated systems, will in certain situations 'leak' elements of the underlying system, this making the abstraction behave in an unexpected way.

An example might be loading a file and reading its contents. The file system APIs are an \_abstraction\_ of the lower level kernel systems, which are themselves an abstraction over the physical processes relating to changing data on a magnetic platter (or flash memory for an SSD). In most cases, the abstraction of treating a file like a stream of binary data will work. However, for a magnetic drive, reading data sequentially will be \*significantly\* faster than random access (due to increased overhead of page faults), but for an SSD drive, this overhead will not be present. Underlying details will need to be understood to deal with this case (for example, database index files are structured to reduce the overhead of random access), the abstraction 'leaks' implementation details the developer may need to be aware of.

The example above can become more complex when \_more\_ abstractions are introduced. The Linux operating system allows files to be accessed over a network, but represented locally as 'normal' files. This abstraction will 'leak' if there are network failures. If a developer treats these files as 'normal' files, without considering the fact that they may be subject to network latency and failures, the solutions will be buggy.

The article describing the law suggests that an over-reliance on abstractions, combined with a poor understanding of the underlying processes, actually makes dealing with the problem at hand \_more\_ complex in some cases.

See also:

- [Hyrum's Law](#hyrums-law-the-law-of-implicit-interfaces)

Real-world examples:

- [Photoshop Slow Startup](https://forums.adobe.com/thread/376152) - an issue I encountered in the past. Photoshop would be slow to startup, sometimes taking minutes. It seems the issue was that on startup it reads some information about the current default printer. However, if that printer is actually a network printer, this could take an extremely long time. The \_abstraction\_ of a network printer being presented to the system similar to a local printer caused an issue for users in poor connectivity situations.

### Localisation

We are currently using [GitLocalize](https://gitlocalize.com) to handle translations. This provides features to make it easier for people to manage translations as changes come in:

![GitLocalize Screenshot](../images/gitlocalize.png)

This is still work in progress - if you would like to be a maintainer for a language just open an issue to get in touch!