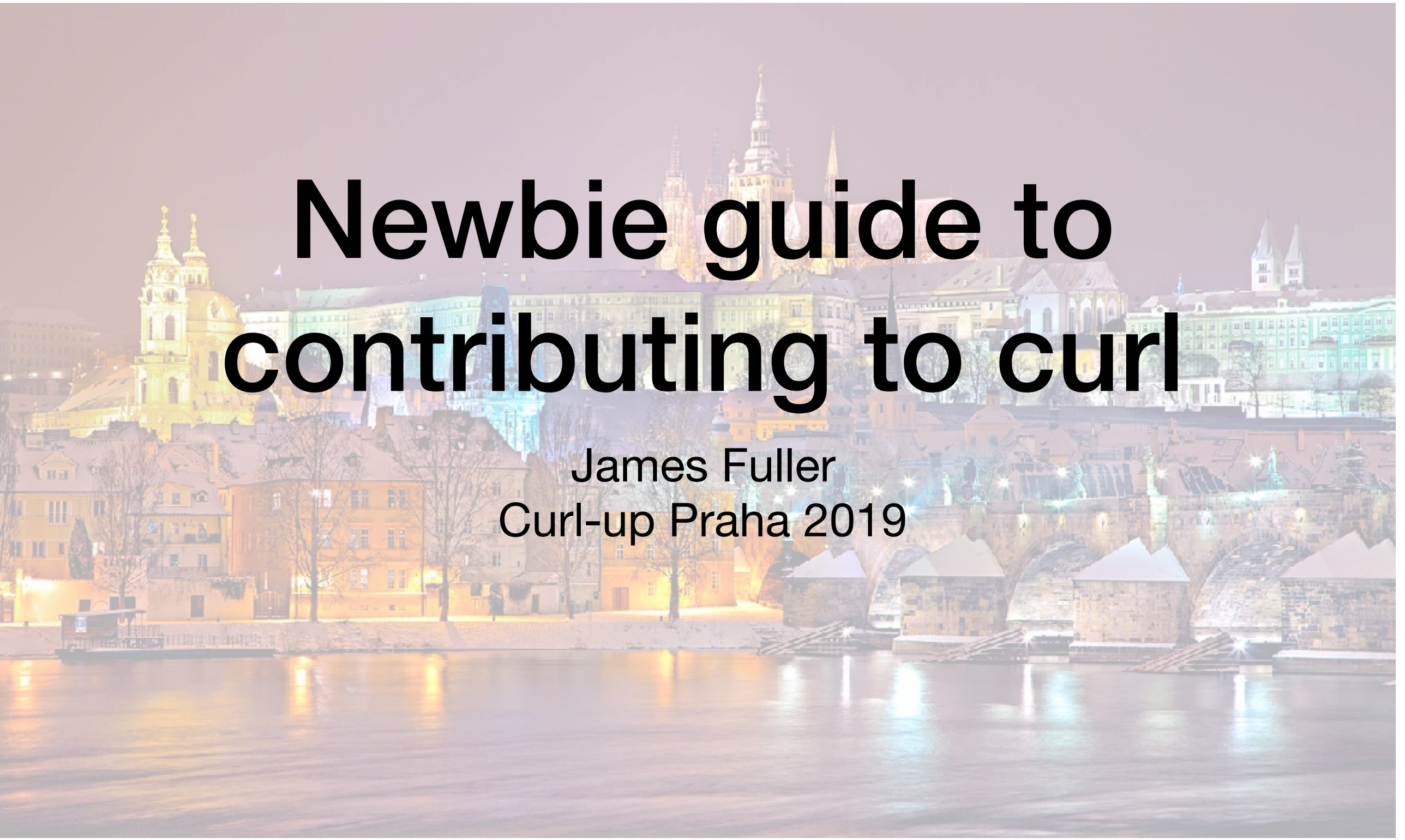


Newbie guide to contributing to curl

James Fuller

Curl-up Praha 2019

The W's

- ***Who*** was involved?
- ***What*** happened?
- ***When*** did it take place?
- ***Where*** did it take place?
- ***Why*** did that happen?

What is curl ?

- curl
- libcurl
- curl-config

What ...

- Daniel Stenberg
- mailing lists
- website
- IRC

How to Download

Method #1

- fork https://github.com/curl/curl
- setup dependencies

Method #2

- https://github.com/xquery/docker-libcurl-dev

- docker pull jamesfuller/libcurl-dev-dependencies-linux

How to Build

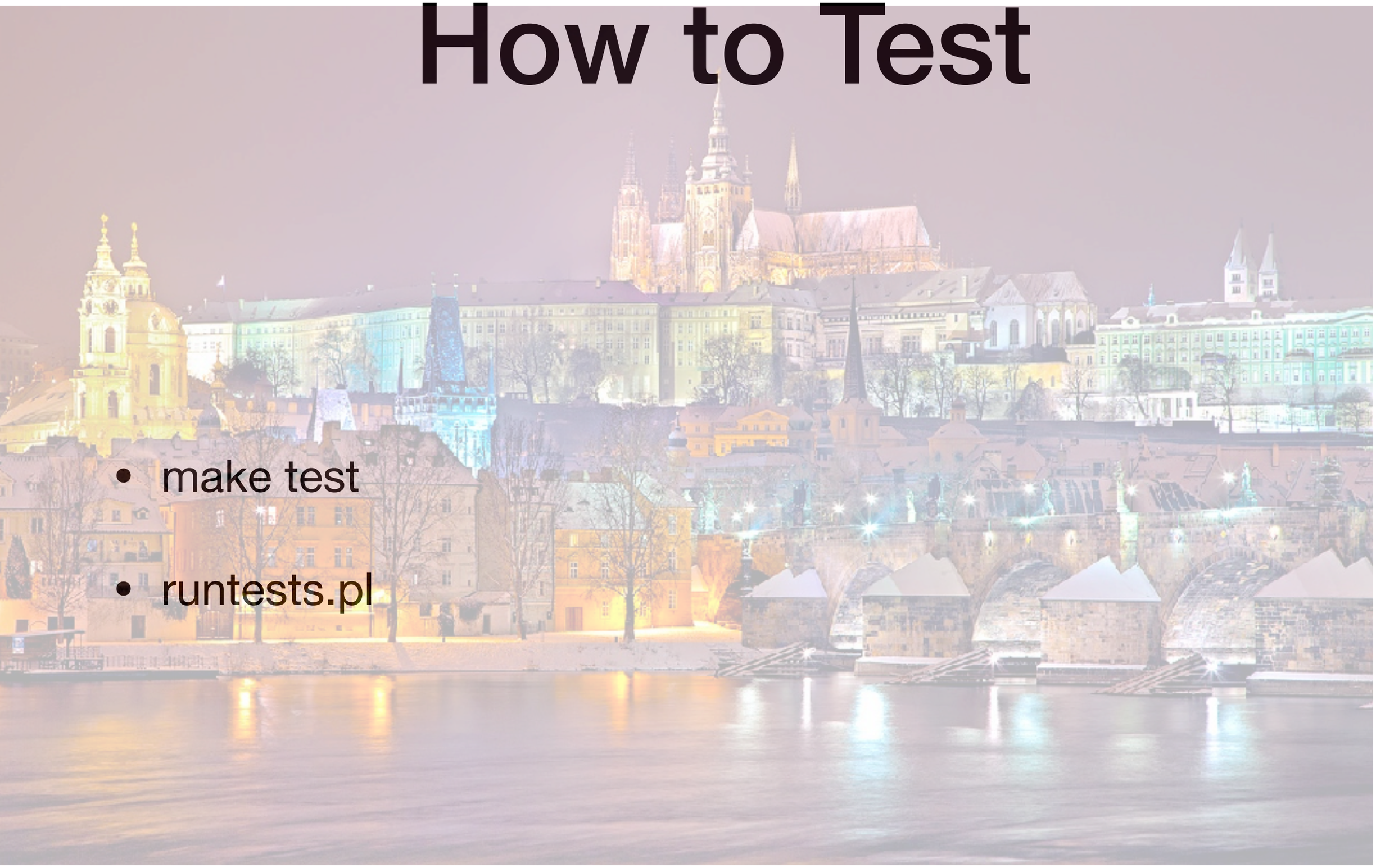
Method #1

- ./buildconf
- ./configure
- make

Method #2

- cmake

How to Test



- make test
- runtests.pl

Why contribute ?

- build the feature you want now !
- ‘reciprocate’ back to the curl community
- learn more about network programming and c
- ‘Fortuna audaces iuvat’

Get involved

- https://curl.haxx.se/docs/help-us.html
- help wanted issue tag
  - https://github.com/curl/curl/issues?q=is%3Aissue+is%3Aopen+label%3A%22help+wanted%22
- https://curl.haxx.se/dev/contribute.html
  - report bugs
  - code
  - docs
  - testing
  - sponsorship
- mailing lists
- IRC
- https://curl.haxx.se/book.html

jim’s seven laws of reporting bugs

Be precise and succinct in describing the bug

Spend an extra second when defining a summary title for a bug. Bug titles should be an assertion about behaviour, avoiding noun-phrase titles, for example:

Good: "Clicking Save button results in server exception"

Bad: "Save button functionality"

Provide the steps to reproduce the bug as well as describing observed and expected behaviour.

Optimally if you can include a small test that **consistently reproduces** the bug this will help the engineer 'teleport' precisely to the code causing the problem.

Bug reports should not rely on ephemeral context eg. a bug report viewed one hour/day/week/month from now should not contain references to 'yesterday' or 'that build' or non specific language like 'too slow' that will require the engineer to 'swap in context'.

Avoid the use of pronouns (like 'it' or 'the window') and specifically name things.

One should strive to write bug reports that are unambiguous and can stand alone.

By all means be **minimally verbose**.

Gather up relevant information on the environment

What versions/branches of software are in use ?

Is this bug specific to an environment (cloud, linux, osx, windows) ?

When did the bug start happening ?

Knowing the exact commit when a bug started nails down its time/space coordinates eg. we know the likely author and piece of code that caused the problem.

Optimally we should bisect until we know the exact commit but often we only know a range of time (eg. many people use nightly builds). In this case we can still review the commit logs and narrow down the range of commits and authors creating a smaller 'problem space' for the engineer to search.

Identify candidate commits

If you have a few authors from a range of commits - sometimes it is easy to match up breakage with an author based on their areas of code responsibility and where in the codebase they are committing to - all without really understanding the code itself.

This range of commits are usually fixing other bugs - go ahead and include links to those bugs to provide as much context as possible.

Search first

Search github and mailing lists

It is not a sin to write a duplicate bug report but synthesising information across several bug reports can make managing bugs more complicated (and slower to address).

Enriching an existing bug report is often more helpful then having a duplicate bug report.

Provide useful context

Relevant log sections, stack traces, error codes, configuration, etc anything to reduce the receiving engineer's need to synthesise additional context.

Do not forget to include other logs (like js console logs).

If the bug involves a REST API then best to include a curl command with relevant payload.

Try to bring as much detail to ambiguous characteristics of the bug.

For example, if a bug is failing intermittently try to identify if the timing of failure is 'random' vs failing over a long period of time or periodically (ex. 'once a day').

Similarly if something is 'too slow' try to characterise with a minimal set of observations illustrating poor performance.

Getting the balance of 'enough' context is an art - focus should be on relevance rather then comprehensive auditing (ex. provide the relevant section of a log or stack trace vs the entire log/stacktrace).

Avoid conflating issues

Try to create a new bug for each different issue.

It is a **red flag** when a single bug becomes a 'container' for several different issues - typically requiring more communication. That entire communication does not necessarily have to be included in the bug report.

In other words, try to keep a bug report and related comments focused on a single 'issue' ... by all means have the conversation to unpick the issues, but do not force engineers in the future (who may have to fix the bug) have to parse irrelevant conversation.

Avoid the pitfalls, traps, quicksand, bugbears and how not to run with scissors when developing for libcurl

- ask for help *considerately*, with minimal code examples and clear framing
  - are you asking a ‘curl’ question ?
  - did you search first …
- fetch and rebase often
- when in doubt run ./buildconf … it is your friend
- run travis on your fork before you raise PR
- do not forget - tests, docs, examples etc …
- if a new feature, then have an ‘incremental discussion’