Machine-Readable Repository Information Query Tool

Lucas Seiki Oshiro

8 de abril de 2025

1 Contact info

- Name: Lucas Seiki Oshiro
- Timezone: GMT-3 (America/São Paulo)
- IRC: lucasoshiro
- Personal page: https://lucasoshiro.github.io/en/
- GitHub: https://github.com/lucasoshiro
- LinkedIn: https://www.linkedin.com/in/lucasseikioshiro/
- GSoC blog: https://lucasoshiro.github.io/gsoc-en

2 About me

My name is Lucas Oshiro, I'm a developer and CS bachelor from São Paulo, Brazil. Currently I'm pursuing a master's degree in CS at University of São Paulo. My interest in Git dates from years ago and I even submitted a patch to its codebase in in 2019, though I couldn't complete it due to scheduling conflicts with my capstone project.

Having experience in the academia, industry and FLOSS, I highly value code quality, code legibility, well-maintained Git histories, unit tests and documentation.

2.1 Previous experience with Git

From 2020 to 2024, I haven't been involved directly with Git development community, however, I kept my interest in Git alive, for example:

- I translated the "Git Internals" chapter of Pro Git to Brazilian Portuguese: GitHub PR;
- I wrote some blog posts about Git, for example:
  - one explaining how Git can be used as a debugging tool: available here;
  - other explaining how Git merges submodules: available here;
- I wrote a compatible subset of Git in Haskell from scratch: available here;
- I helped organizing a Git Introductory Workshop at my University: more info here;

- I presented some lectures about Git in a company that I worked some years ago, covering the Git internals (objects, references, packfile) and debugging and archaeology related Git tools (blame, bisect, pickaxe, ls-files, etc).

2.2 Previous experience with C and open-source

I also have experience with C and some C++. During my CS course, C was one of the primary languages that I used. I also worked with C/C++, some examples are:

- I wrote an AMQP message broker from scratch: GitHub;
- I sent simple patches to the IIO subsystem of the Linux kernel: patches listed here;
- I contributed to the Marlin firmware for 3D printers: patches listed here;
- I'm writing a module for the ns-3 network simulator, dealing with both C and C++ codebases (currently under development, I plan to write a paper and make the code available soon);

During my CS course I also was member of FLUSP, a group in my university focused on FLOSS contributions and from Hardware Livre USP, another group that was focused on working with open-source hardware.

As a master's student, I'm one of the Open Science Ambassadors of my University (more information here, in Portuguese), an initiative for promoting Open Science principles (which include open-source software) in the department where I study.

I also contributed to some other free/open-source software, which I list in my personal page here.

2.3 Activity in the Git community in 2025

Since I decided to submit a proposal for GSoC, I sent some patches to the Git codebase and git.github.io where:

- As my GSoC microproject, I replaced some `test -f` by `test_path_is_file`: available here, merged to **master**;
- I added a paragraph to the merge-strategies documentation describing how Git merges submodules (based on the blog post that I mentioned before): available here, merged to **master**;
- I sent a patchset where I added a new `--subject-extra-prefix` flag for `git format-patch`, allowing the user to quickly prepend tags like [GSoC], [Newbie] or [Outreachy] to the beginning of the subject: available here. This patchset was rejected in favor of just using `--subject-prefix='GSoC PATCH'` or similar;
- After the feedback on the previous rejected patchset, I opened a Pull Request on git.github.io replacing the occurrences of `[GSoC][PATCH]` by `[GSoC PATCH]`, merged to **master**;
- I added a new userdiff driver for INI files, initially target for gitconfig files: available here, merged to **next**.

Beyond contributions, I also helped people on the mailing list that needed assistance on Git features and documentation. You can see my activity on Git mailing list here.

3 Project Proposal

Based on the description provided in in the GSoC 2025 ideas page, the goal of this project is to create a new Git command for querying information from a repository and returning it as a semi-structured data format as a JSON output.

A first idea of how this command would be named is `git repo-info`.

In the scope of this project, the JSON output will only include data that can currently be retrieved through existing Git commands. The main idea is to centralize data from the section Options for Files `git rev-parse`, which currently is overloaded with features that don't fit its main purpose.

Some of the data that we expect to retrieve and centralize are:

- The hashing algorithm (i.e. `sha1` or `sha256`), which currently can be retrieved using `git rev-parse --show-object-format`;
- The Git directory of the repository, currently retrieved by running `git rev-parse --git-dir`;
- The common Git directory, currently retrieved by running `--git-common-dir`;
- The top level directory of the repository, currently retrieved by using `git rev-parse --show-toplevel`;
- The reference database format (i.e. currently, `files` or `reftable`), currently retrieved by running `git rev-parse --show-ref-format`;
- The absolute path of the superproject, currently retrieved by running `git rev-parse --show-superproject-working-tree`;
- Whether this is a bare repository, currently retrieved by running `git --is-bare-repository`;
- Whether this is a shallow repository, currently retrieved by running `git --is-shallow-repository`.

The information that we want to compile is currently accessible with different sets of flags, so the user that wants to read them needs to have an advanced knowledge on Git. After having the repository details consolidated in a single command, the user will be able to quickly retrieve what it desires without navigating a complex combination of commands and flags.

A side effect is decreasing the reponsibility of `rev-parse`, like `git switch` and `git restore` did for `git checkout`.

3.1 Use cases

Use cases that will be benefited of `git repo-info` will be:

- CLI tools that display formatted information about a Git repository, for example, OneFetch;
- Text editors, IDEs and plugins that have front-ends for Git, such as Magit or GitLens;
- FLOSS repository tracking software, for example, kworkflow, ctracker;
- Tools that integrate with Git and currently rely on `rev-parse` to get information about the repository;

3.2 Planned features

`git repo-info` consists of one big feature: produce the JSON with repository metadata. At first, this should be populated with the data that currently can only be retrieved through `git rev-parse`, like the ones listed before.

Other data may be added depending on the demands of the Git community and the user base.

By now, it's not possible to decide precisely how this command would work without it being more discussed. But a first draft on how it would be invoked and the output that it will produce is:

```
$ git repo-info

{
    "object-format": "sha1",
    "git-dir": ".git",
    "common-dir": ".",
    "toplevel": "/home/user/my_repo",
    "ref-format": "files",
    "superproject-working-tree": "/home/user/my_super_repo",
    "bare-repository": true,
    "shallow-repository": false
}
```

This first draft will be sent to the mailing list in order to get feedback from the Git community.

3.3 Development plan

Since this is a new command that is not directly related to any specific existent command, its main code will probably be placed in a new file `builtin/repo-info.c`.

This project will use JSON-related functionality to Git currently provided by `json-writer.c` (currently used by `trace2` features), extending it if needed.

The functionality of `git repo-info` can be divided into two categories:

1. **Data gathering**: retrieving data from different sources, calling existent functions and reading data structures declared in other files;
2. **Data serialization**: formatting the gathered data in a JSON format. This represents two challenges: generating the JSON itself and designing the schema for how the desired data will be presented.

Since the exported data is already provided by other Git commands, it won't be difficult to implement this side of the functionality. The main task for gathering data will be inspect the existing codebase and find the functions and data structures that will feed our output. For example, the already mentioned data:

- Hashing algorithm: `git rev-parse` reads from the field `the_repository->hash_algo->name` (aka `the_hash_algo->name`);
- The Git directory, the common Git directory and top level directory: `git rev-parse` formats those paths using `print_path`, based on the `prefix` parameter and the fields `commondir` and from `the_repository`;

- Reference database format: `git rev-parse` retrieves it from `ref_storage_format_to_name`;
- Absolute path of the superproject: `git rev-parse` retrieves it from the function `get_superproject_working_tree`;
- Whether the repository is bare: `git rev-parse` retrieves it from the function `is_bare_repository`;
- Whether the repository is shallow: `git rev-parse` retrieves it from the function `is_repository_shallow`.

Designing the schema, however, requires special planning, as the flexibility of semi-structured data like JSON may lead to early bad decisions. A solution may emerge by analysing other software that export JSON as metadata.

3.4 Schedule

1. **Now – May 5th**: Requirements gathering
   - Present to the Git community the proposal, asking what are the features that it demands;
   - Inspect codebases that uses Git (specially `rev-parse`) as data source;
   - Studying more deeply the codebase, specially the `rev-parse` source code;
2. **May 6th – June 1st**: Community bonding
   - Get in touch with the mentors;
   - Present to the community a first proposal of the JSON schema;
   - Receive feedback from the community about the schema;
   - Present a first proposal on the command line interface;
   - Receive feedback from the community about the command line interface;
3. **June 2nd – June 31th**: First coding round
   - Decide how the CLI should behave, that is, what should be exported by default, what should be outputted only by using flags and what should not be outputted by using disabling flags;
   - Write a minimal JSON serializer, focusing on export only the top-level object with only string values;
   - Define a simple data structure that holds only one field of the data that we want to export;
   - Introduce a first version of the command, outputting the first data structure using the first JSON serializer;
   - Write test cases in different scenarios comparing the output of the new command with the outputs of the existing commands;
   - Write a minimal documentation file for the new command, making it explicitly that the command is experimental and it's still under development;

4. **July 1st – July 14th**: Second coding round
    - Fill the data structure with all other string fields that should be exported;
    - Add flags for filtering the data output;
    - Write tests for each one of the new fields and flags;
    - Improve the documentation file, explaining what the command does, its output format and what the flags that were implemented so far do;
5. **July 15th – August 15th**: Third coding round
    - Improve the JSON serializer, allowing other data types: array, number, boolean, null and nested objects;
    - Fill the data structure with the remaining non-string fields;
    - Write flags, tests and documentation for the types that were implemented in this round;
6. **August 16th – August 25th**: Fourth coding round
    - Polish the documentation, providing examples on how to use it and notes on why this command was created;
    - Finish remaining work from the previous rounds that still need to be concluded;

3.5 Availability

2025 is my last year in my master's degree. Currently, I'm not attending any classes and I am more focused on developing the software of my research, performing experiments and writing scientific articles and my thesis. Since my advisor is aware that I'm proposing a GSoC project, it will be possible to work on Git while working on my master's tasks.