GSoC 2026 Proposal: Improve the new git repo command

Author: K Jayatheerth <jayatheerthkulkarni2005@gmail.com>

1. About Me

I am a junior Computer Science undergraduate at Geethanjali College of Engineering and Technology with a focus on systems programming.

My interest in the Git project stems from a desire to understand the internals of version control and contribute to a tool that is fundamental to the global software development ecosystem.

I am applying to this project specifically because I have followed the development of `builtin/repo.c` since its beginning in GSoC 2025. I have closely tracked its evolution through Lucas Oshiro's development blogs (https://lucasoshiro.github.io/gsoc-en/) and have studied the mailing list discussions.

2. Contact Information

- **Email:** jayatheerthkulkarni2005@gmail.com
- **GitHub:** https://github.com/jayatheerthkulkarni
- **Website:** https://jayatheerth.com/
- **LinkedIn:** https://www.linkedin.com/in/jayatheerth/
- **Timezone:** UTC+05:30 (IST)
- **Tech Stack:** C, Shell, Rust, Go

3. Previous Experience with Git

I have been an active contributor to the Git mailing list since February 2025. I have navigated the full patch lifecycle that includes submission and review to integration into `seen`, `next` and `master`.

My work has focused on areas directly relevant to this proposal: path matching, submodule interactions, and safe memory handling.

I have ensured to not pick any micro-project that would directly affect other GSoC applicants.

3.1 Selected Contributions (By Relevance)

(1) [PATCH v3] dir.c: literal match with wildcard in pathspec should still glob
Date: April 22, 2025
Status: Merged into master
Link: LINK
Impact: Fixed a logic error in `dir.c` where wildcard pattern matching terminated prematurely.

(2) [PATCH v3] stash: fix incorrect branch name in stash message
Date: June 30, 2025 (Featured in Git Rev News #124)
Status: Merged into master
Link: git.github.io/rev_news/2025/06/30/edition-124/
Impact: Fixed a memory safety issue (static buffer reuse) in `refs_resolve_ref_unsafe` that corrupted stash messages in submodule environments.

(3) [PATCH v3 0/3] clean up a few things
- **[PATCH v3 1/3] path: remove unused header**
- **[PATCH v3 2/3] path: use size_t for dir_prefix length**
- **[PATCH v3 3/3] path: remove redundant function calls**

Date: March 2, 2026
Status: Merged into master
Link: LINK
Impact: Improves readability and type safety in `path.c`.

(4) [PATCH v8 0/2] Avoid submodule overwritten and skip redundant active entries
- **[PATCH v8 1/2] submodule: prevent overwriting .gitmodules entry on path reuse**
- **[PATCH v8 2/2] submodule: skip redundant active entries when pattern covers path**

Date: July 25, 2025
Status: Merged into master
Link: LINK
Impact: Prevented accidental overwrites of `.gitmodules` configuration when paths are reused.

(5) [PATCH] repo: Remove unnecessary variable shadow
Date: Febuary 23, 2026
Status: Merged into master
Link: LINK
Impact: A targeted cleanup in `builtin/repo.c`.

(6) [PATCH v4 0/3] Update MyFirstContribution.adoc to follow modern practices
- **[PATCH v4 1/3] docs: remove unused mentoring mailing list reference**
- **[PATCH v4 2/3] docs: clarify cmd_psuh signature and explain UNUSED macro**
- **[PATCH v4 3/3] docs: replace git_config to repo_config**

Date: May 18, 2025

Status: Merged into master
Link: LINK
Impact: Modernized documentation to reflect current API usage (e.g., `repo_config`).

(7) [GSoC] t7101: modernize test path checks (Microproject)
Date: January 09, 2026
Status: Merged into master
Link: LINK
Impact: Modernized legacy `test -f` checks to `test_path_is_file`.

3.2 Community Engagement & Design Discussions

Beyond my own patch series, I actively participate in technical discussions to shape the future of the codebase.

(1) [Discussion] Regarding 'repo info' architecture
Link: LINK
Context: Participated in discussions regarding the path normalization strategies for the new command.

(2) [Review] Reviewing other contributors' patches
Link 1: LINK
Link 2: LINK
Link 3: LINK
Link 4: LINK
Context: I review patches from time to time of other contributors to help reduce the burden on maintainers and ensure code quality.

(3) [Discussion] Regarding 'git pull –rebase' fork-point behavior
Link: LINK
Context: Wanted clarification on the fork-point heuristic and identified the core internal functions (`get_rebase_fork_point()` and `get_rebase_newbase_and_upstream()`) responsible for the behavior to begin exploring potential approaches for a fix.

4. Project Proposal

Taken from the ideas page of the GSoC 2026 projects list, the goal of this project is to improve the existing git repo command. To specify the objectives in detail, this proposal covers three main goals:

a. Add path.* keys into the repo info command.
b. Remove the USE_THE_REPOSITORY_VARIABLE.
c. Form categories for the keys for the git repo command.

I had a hard time picking which of these to keep out of scope, as I equally liked all the ideas in the proposal. However, taking timeline into consideration, I have moved Goal d to the stretch goal/post-GSoC commitment to ensure the core deliverables are highly stable.

d. Study and collaborate with Justin Tobler for git structure work.

Goal a: Add path.* keys into the repo info command

This is currently a high-activity area on the mailing list. To avoid redundant effort or conflicting patches, my strategy will be adaptive based on the state of the tree during the Community Bonding period.

I have reviewed the overlapping patch series submitted by Lucas Oshiro and Eslam Reda. Since discussions are already ongoing, my goal is to build upon their foundation rather than reinventing it. I am already actively participating in the core design discussions on the mailing list (LINK) regarding how to handle relative versus absolute path outputs. We are currently weighing several approaches: global `-path-format` flags, stateless virtual keys (e.g., `path.absolute.toplevel`), and `ref-filter` style format modifiers. I will prioritize finalizing this design consensus during the bonding period, and then focus on implementing the remaining core keys and cross-platform edge cases.

I will ensure the following keys are fully supported and tested by the end of the GSoC timeline:

- path.git-dir
- path.common-dir
- path.worktree
- path.objects
- path.hooks
- path.index
- path.grafts

If these keys are already added by the time I am selected, I would happily focus on the other goals or the stretch goals.

Goal b: Remove USE_THE_REPOSITORY_VARIABLE

Removing this macro is a priority to ensure the new command is library-safe. I have identified that `get_layout_bare(struct repository *repo UNUSED, ...)` is the primary inhibitor in `builtin/repo.c`.

Currently, this function ignores its argument and uses the global `is_bare_repository()` macro.

My implementation plan is as follows:
1. Refactor `get_layout_bare` to drop the `UNUSED` tag and explicitly use its `repo` argument.
2. Replace the global macro call with a check against `repo->worktree`. As defined in `repository.h`, a `NULL` value for `repo->worktree` indicates the absence of a working directory (i.e., a bare repository).
3. This removes the dependency on global state without requiring changes to the `struct repository` definition itself.

This change allows `builtin/repo.c` to drop the global state macro entirely, making the command fully reentrant and library-safe.

Goal c: Implement Category-Based Querying (Structured Discovery)

The current implementation of `repo info` suffers from an "all-or-nothing" usability problem. Users must either know the exact key name or dump the entire configuration.

My proposal is to implement Prefix-Based Querying.

The internal `repo_info_fields` array is already structured for this:

```
static const struct field repo_info_fields[] = {
    { "layout.bare", get_layout_bare },
    { "layout.shallow", get_layout_shallow },
    ...
};
```

Since this array is lexicographically sorted, I can implement efficient lookup without new data structures:

1. Use `bsearch` to find the first key matching the requested prefix (e.g., `git repo info layout`).
2. Iterate linearly from that point, printing keys as long as the prefix matches.
3. Stop immediately when the prefix no longer matches.

Note: the next goal is a stretch goal. I have only added these goals since I had already done the research and had to cut it off only because of timeline.

Goal d: Repository Health Diagnostics and Metric Distributions

While 'git repo structure' recently gained the ability to surface basic object counts and maximum sizes through Justin Tobler's recent patch series (now in 'next'), the command still presents data as raw, isolated extremes.

My goal is to evolve 'git repo structure' into a comprehensive diagnostic tool by introducing metric distributions, packfile analysis, and actionable health thresholds.

1. Object Size and Entry Distributions (Histograms):
As discussed recently on the mailing list by Patrick Steinhardt and Junio C Hamano, extreme maximums are useful, but distributions provide the real picture of repository health. I will implement a streaming bucketing system during the object walk to track size distributions (e.g., blob sizes) and entry distributions (e.g., tree entry counts). These will be formatted into an optional ASCII bar chart output to visualize repository shape.

2. Packfile and Pathological Path Metrics:
I will expand the traversal to capture metrics that directly impact performance and cross-platform compatibility:

- **Longest Delta Chains:** Excessively long delta chains degrade packfile performance and clone times.
- **Maximum Path Depth & Length:** Critical for identifying paths that silently break checkouts on systems with path-length limits (e.g., Windows).

3. Threshold and Concern Levels:
Currently, the command dumps data agnostically. Inspired by the 'level of concern' logic in tools like 'git-sizer', I will introduce a mechanism to visually flag metrics that exceed typical Git "sweet spots" (e.g., flagging trees with $>$1000 entries, or $>$10 octopus merge parents). This transforms the command into an actionable CI/CD health-check tool.

4. Integration into the Query Architecture:
I will ensure that both these new diagnostic metrics and the recently introduced ODB extremes (max parents, max tree entries) are seamlessly integrated into the prefix-based querying system proposed in Goal c, respecting the context-aware libification from Goal b.

Approach

I understand the review cycle of Git cannot always be quick, therefore my timeline is designed to buffer patch series and pivot between tasks while waiting for reviews.

To maximize productivity while respecting Git's review cycles and preventing merge conflicts, I am splitting the core work into two tracks:

i. Track - a (The Foundation): This will include Goal a (path.* keys) and Goal b (Libification). Completing path resolution and safely removing global state provides the clean foundation required

for building a new querying API.
ii. Track - b (The API): This will include Goal c (Query Architecture). Development here will fully spin up once Track a is stabilized.

5. Timeline

I will dedicate 35-40 hours per week to this project. Because Git uses an asynchronous mailing list review process, my timeline is designed to build patch series with ample buffer room for design iterations.

Community Bonding Period (May 1 - May 24, 2026)

- Synchronize with Lucas Oshiro and mentors on the current state of 'path.*' architecture in the master branch.
- Finalize the exact list of remaining path keys needed.
- Publish my introductory GSoC blog post on jayatheerth.com/blogs.
- Pick up a small RFC question if we need globs in querying keys.

Phase 1: Track a Execution (May 25 – July 10, 2026)
This phase focuses strictly on stabilizing Track a: Goal a (path.* keys) and Goal b (Libification).

- **Weeks 1 - 4 (May 25 – June 21):**
  - Work concurrently on Track a deliverables to build the command's foundation.
  - Implement the core missing `path.*` keys locally (git-dir, common-dir, worktree, objects, hooks) and write comprehensive, OS-agnostic tests in t/.
  - Refactor `get_layout_bare` to drop the UNUSED macro and completely remove `USE_THE_REPOSITORY_VARIABLE` from `builtin/repo.c`.
  - Milestone: Submit [PATCH v1] series for both Goal a (Paths) and Goal b (Libification).
- **Weeks 5 - 7 (June 22 – July 10):**
  - Buffer time for asynchronous review cycles. Address mailing list feedback for Track a patches and submit subsequent versions ([PATCH v2], etc.).
  - Midterm Evaluation: Ensure Track a patch series are stabilized and queued in `next`.
  - Publish a detailed halfway-point blog post.

Phase 2: Track b Execution (July 11 – August 21, 2026)
With Track a stabilizing, development shifts entirely to Goal c (Query Architecture).

- **Weeks 8 - 9 (July 11 – July 24):**
  - Using the feedback from the community bonding RFC, implement the query system.
- **Weeks 10 - 11 (July 25 – August 7):**

  - Run the full test suite and ensure querying handles all edge cases.
  - Milestone: Submit [PATCH v1] encompassing the Goal c query architecture.

- **Weeks 12 - 13 (August 8 – August 21) [Buffer & Review Weeks]:**
  - Dedicated time for addressing potentially complex architectural reviews for the querying patch series.
  - Submit subsequent patch versions ([PATCH v2], [PATCH v3]).
  - Catch up on any unforeseen edge cases from previous weeks.

Phase 3: Final Handoff (August 22 – August 31, 2026)

- Write the final project report and publish the concluding blog post.
- Submit the Final GSoC Evaluation.

5.1 Stretch Goals: Repository Structure Diagnostics

If the core `repo info` deliverables (Paths, Libification, and Querying) are completed and merged ahead of schedule, I will pivot to extending the `git repo structure` command (Goal d). I will prioritize implementing the streaming bucketing system during the object walk to output ASCII histograms for Object Size and Tree Entry distributions.

6. Availability

This timeline aligns perfectly with my schedule. The project kicks off in May and June during which I will be on summer vacation and can dedicate 35-50 hours a week. During July, I will transition into my final year of university. My academic schedule during this period is highly flexible.

7. Blogging

I have a domain setup at jayatheerth.com. As patches flow and the project progresses, I will host a dedicated endpoint at `/blogs` to provide comprehensive, weekly and a phase wise coverage of my project.

8. Post GSoC

I actively follow the mailing list and intend to continue contributing bug fixes and enhancements. I would absolutely complete Goal d in case no one has any reservations. I have been a part of the Git community since 2025 and hopefully will continue to be one for a long time.

Probably mentoring the next year GSoC applicants if that is a possibility.