Git Virtual Contributor's Summit

September 26 & 27th, 2023

Day 1

Welcome / Conservancy Update

Presenter: Taylor Blau, Notetaker: Keanen Wold

- Software Freedom Conservancy status report
- We have about $89k in the Git project account (up ~$20k from last year)
    - Biggest expense is Heroku - Fusion has been covering the bill
        - There's on and off work on porting Git homepage from a Rails app to a static site: https://github.com/git/git-scm.com/issues/942
    - Dan Moore from FusionAuth has been providing donations
    - Ideally we are able to move away from using Heroku, but in the meantime we'll have coverage either from (a) FusionAuth, or (b) Heroku's new open-source credit system
- We have more money than we have plans for, we're looking for ideas on how to spend this money such as funding people to visit our conferences and sponsoring students to learn more about Git
    - Outreachy, etc.
    - travel to Git Merge
- Trademark considerations for people using "Git" in their product names
    - We do have general council and are trying to think more about what the Git trademark means
    - Question - are there other conservancy products who have trademark issues (Inkscape, Samba,...)
        - They hold all trademarks for their projects
        - Git has had the most problems with people/products using Git in their name
        - They reach out with letters, etc. and have not had to take legal action in most cases
    - Question - how do we enforce the rules when we have GitHub and GitLab?
        - The trademark has exemptions for Hub and Lab
        - We need to hold the line for the trademark for new companies, etc. using the name otherwise we lose our leverage to protect the name (dilution of the mark)
    - Question - have the trademark 'offenses' been growing?
        - It's been pretty stable, 12-24 per year
        - We're looking to be fair, and to have cohesive strategy

- Additional questions can be sent to Pono Takamori (SFC)

Next-gen reference backends

Presenter: Patrick Steinhardt, Notetaker: Karthik Nayak

- Summary: There have been multiple proposals for reference backends on the mailing list. Trying to converge to one solution.
- Problem: At GitLab we have certain repos with large amounts of references. Some repos have multi-million refs which causes scalability issues.
    - Current files backend uses a combination of loose files and packed-refs.
    - Deletion performance is bad.
    - Reference lookups are slow.
    - Storage space is also large (around 1GB packed-refs file).
    - There are some patches which improved the situation. e.g. skip-list for packed-refs by Taylor Blau.
    - Atomic updates are currently not possible.
    - This is not an issue only faced by GitLab
- Two solutions proposed:
    - Reftables: Originally implemented by JGit (Shawn Pearce, 2017)
        - Google was storing the data in a table with one ref per row (in BigTable). This data was encrypted, which changes the ordering.
        - This led to realizing the ref storage itself was not optimal, so based on existing solutions at Google there was a proposal by Shawn and was implemented in JGit.
        - Design inspired by SSTable (Sorted Strings Table) data structure, uses prefix compression (like Trie), indexes, multiple levels, 64KB blocks
        - This solved the ref storage problem at Google; still a problem for Gerrit - because it uses lots of references.
        - The implementation in JGit by adoption was low because of a compatibility requirement with CGit.
        - New patch series submitted which swaps out the packed-refs with ref-tables, while keeping the existing file based loose-refs (layered approach).
            - *(Jeff Hostetler): Windows users might disagree on the different case semantics of loose and packed refs…*
            - *(brian m. carlson): I don't think we want case-insensitive refs. It's impossible to case fold correctly in a locale-insensitive way, so we should preserve the existing semantics of refs that we have.*
            - *(Han-Wen Nienhuys): @jh - Ah, yes if you mix loose refs and reftable, then the case folding becomes relevant again.*
        - There is API for using refs, it kind of works, but there are lot of edge cases, and lots of built-in assumptions that Git is using files on disk for references buried in the code, and avoiding the API

        - *(Phillip Wood): Han-Wen - are there many places left where we read/write files rather than using the ref API? I thought you'd done quite a lot of work to fix that.*
        - *(Han-Wen Nienhuys): worktrees is something I didn't figure out.*
    - Incremental take on reference backend (aka. packed-refs v2) by Derrick Stolee
        - Uses pre-existing infrastructure (chunk format API) in the git project. Makes it a more natural extension.
        - First part was to support a multi backend structure
            - Loose files + packed-refs (current implementation)
            - Loose files + reftable
            - Loose files + packed-refs v2
        - Second part was packed references v2 in the Git project
- Question: How do we take it forward from here?
    - (Emily Shaffer): What if the existing backend existed as a library (with a clear interface). Might be easier to replace and experiment with.
        - Jeff: A lot of work in that direction has already been landed. But there is still some bleed of the implementation in other parts of the code. Might be messy to clean up.
        - Patrick: Different implementations by different hosting providers with different requirements might cause issues for clients.
            - If some ref name is allowed for one git hosting provider, but not allowed for the other
            - *(Jeff Hostetler): Also different case folding and D/F collisions…*
            - *(Han-Wen Nienhuys): I added D/F conflict checks in the reftable code. Case folding not working is a feature, not a bug :-)*

    - Deletion performance is not the only issue faced (at GitLab) there are also deadlocks faced around this.
    - (brian m. carlson): If you have a large number of remote tracking refs you face the same perf issues, for example when renaming a remote, which involves mass rename of remote-tracking refs. Transactions for refs operation would be very useful here.
    - (Patrick): Any preference of which solution to go forward. GitLab is interested to pick this up and mostly going forward with reftables.
    - Derrick Stolee worked on deletion-specific solutions; but he is changing roles, and does not currently have time for git@
    - Reftables does support tombstoning, which should solve the problem with multiple deletions.
        - There is still a problem with refs being a prefix of other refs (ref/a and ref/a/b).
    - (Patrick): will pick work on reftables, looks like nobody else is working on this
    - Is there a world where loose refs are removed completely and replaced with reftables.

        - Debugging is much easier with loose refs, reftables is binary formatting. Might need additional tooling here. This is already proven to be working at Google.
    -

Libification Goals and Progress

Presenter: Emily Shaffer, Notetaker: Taylor Blau

- The effort is to isolate some parts of Git into smaller, independently buildable libraries.
    - Can unit test it,
    - swap out implementations, etc.
- Calvin Wan has been working on extracting a common set of interfaces, refining the types, etc. This is in pursuit of a “standard library” implementation for Git. Close to being shippable.
- Josh Steadmon spent some time in the second half of the year suggesting a unit testing framework in order to test the library interfaces beyond our standard shell tests.
- Goals:
    - Google has a couple of ways to proceed with their libification effort. Community input is solicited:
        - Interfaces for VFS / callable by IDE integration to avoid shelling out
        - Target libification for the sake of Git itself. Code clean-up, making the code more predictable / testable. Example being submodules, which are messy and difficult to reason about. References backend, etc.
- Is there an appetite for libification? Some particular component that would especially benefit from clean-up, being made more test-able, hot-swappable, etc.
    - Libification would make it easier to reason about submodules
- (From Emily’s comment above) If others are implementing the basic references backend via a different implementation, how do we make sure that we are building compatible parts? Goal would be to have Git’s unit tests pass against a different API.
- (Patrick Steinhardt) For reference backends especially: would like to be able to split between “policy” and “mechanism”. This would avoid the issue discussed in the last session where different e.g. refs backend implementations have different behavior.
    - Emily: white-box tests for the API to make sure that different implementations meet the policy
- (Jonathan Nieder) For reference backends in particular, the current implementation has an odd “layering” scheme - packed-refs today is an incomplete backend using the same interface as the complete “loose and packed refs” backend, serves as a mechanism without fulfilling the policy requirements. The approach above seems like a positive change.
- (Emily) Are also looking into a similar project around the object store, but have found that it is deeply intertwined throughout the rest of the code base. Difficult to reason about, even without a library interface. Can we make any given change safely?

    - Hunch is that it is still useful to target that sort of thing, even if there aren't clear boundaries.
    - In the interim, can still be part of the same compilation unit, just creating a clearer boundary.
- (Emily) For hosting providers and others building things on top of git, are there parts of git functionality that you'd like to have libified so you can get benefits without having to wait for feature lag?
- (brian) not interested in using Git as a library in GitHub's codebase because of license incompatibility. Would like to experiment with different algorithms for packing and delta-fication in Rust as a highly parallel system. Would be nice to be able to swap out something that is C-compatible. Have been able to make changes in libgit2 while causing libgit2 to segfault, doesn't want to write more segfaults.
- (Taylor) There's an effort going on in GitHub to reduce our dependency on libgit2, precisely for the feature lag reason Emily mentions. I don't think we're planning on using it as a library soon, but we rely on the Git command-line interface through fork/exec
- (Emily) Is licensing the only obstacle to using Git as a library, or are there other practical concerns?
- (Jeff Hostetler) Pulled libgit2sharp out of Visual Studio. Issues with crashing, but also running into classical issues with large repositories. Memory consumption was a real issue at the time. Safer to have memory segmented across multiple processes so that processes can't touch other processes memory space; crashing git / libgit2 does not crash Visual Studio.
- (Emily) Interesting: thinks that performance overhead would outweigh the memory issues.
- (Patrick) To reiterate from GitLab's point of view: we are in the same boat as Microsoft and GitHub. Have used libgit2 extensively in the past, but was able to drop support last month. No plans to use Git as a library in the future. Having a process boundary is useful, avoids memory leaks, bugs in Git spilling out to GitLab. Still have an "upstream-first" policy. Benefits everybody by spreading the maintenance burden and ensuring that others can benefit from such functionality.
- (Emily) If we had the capacity to write portions of Git's code in Rust (memory safety, performance, use it as a library), would we want to use it?
    - (Junio) I notice in the participant list people like Randall who work on NonStop. I'd worry about the effect on minority stakeholders, portability.
    - (Junio) Not fundamentally opposed to the direction.
    - Rust is portable, but does it support all platforms that Git currently supports and should support?
        - *(brian m. carlson): ia64 and alpha are problems because LLVM doesn't support them. They're obsolete and can't be bought new, but ia64 is a NonStop target and some older systems do still have those chips.*
        - *(Vincenzo Palazzo): For not supported architecture there is the possibility also to consider looking at the gcc rust frontend (?)*
        - *(Phillip Wood): Æver tests git on HP-UX and AIX, I'm not sure if they have rust support or if we have any users on HP-UX*

        - *(brian m. carlson): AIX has Rust. HP-UX, I don't think so yet.*
    - Cargo, Rust package manager, would also be useful (for libification efforts)
- (Elijah) did not parallelize the C implementation of the new ORT backend. Wanted to rewrite it in Rust, cleaned up headers as a side-effect, and looked at other bits. Merge backends are already pluggable, could have a “normal” one in addition to a Rust backend.
- (Emily) If we already have something in C that establishes an existing API boundary, that makes it more tenable to rewrite it in Rust. Could say that the C version is deprecated and make future changes to Rust.
- (brian) Thinks they would be in favor of that; is personally happy to say that operating systems need to accept support for bottom languages eventually. All of the main Debian architectures in use have Rust ports. They are portable to all of the main architectures. Would make it easier to do unit testing. Could add parallelization and optimization without worrying about race conditions, which would be a benefit. Is happy to implement unit tests with Rust’s nicer ecosystem.
- (Taylor) Is it just NonStop?
- (Elijah) Randall mentioned that they have a contractual agreement that is supposed to expire at some point (https://lore.kernel.org/git/004601d8ed6b$13a2f580$3ae8e080$@nexbridge.com/). Could we have a transition plan that:
    - Keeps NonStop users happy until their contract expires.
    - Allows the rest of us to get up to speed with Rust.
- (Jonathan Nieder) doing this in a “self-contained module” mode with fallback C implementation gives us the opportunity to back out in the future (at least in the early periods while we’re still learning).
- (Jonathan Tan) back to process isolation: is the short lifetime of the process important?
- (Taylor Blau) seems like an impossible goal to be able to do multi-command executions in a single process, the code is just not designed for it.
- (Junio) is anybody using the `` `git cat-file --batch-command` `` mode that switches between batch and batch-check.
    - *(brian m. carlson): I think we use it in some cases in Git LFS as well. `git cat-file --batch-check`, that is.*
- (Patrick Steinhardt) they are longer lived, but only “middle” long-lived. GitLab limits the maximum runtime, on the order of ~minutes, at which point they are reaped.
- (Taylor Blau) lots of issues besides memory leaks that would become an issue
- (Jeff Hostetler) would be nice to keep memory-hungry components pinned across multiple command-equivalents.
- (Taylor Blau): would long lived process respond correctly to changes in repository, for example re-read on the index refresh; much state is kept
- (Taylor Blau): same issue as reading configuration.

Designing a Makefile for multiple libraries

Presenter: Calvin Wan, Notetaker: Keanen Wold

- Looking for help with makefile use and how he's making libraries
- Wants to have rules to leverage makefiles repeatable for future libraries
- Wants a fail fast for breaking libraries
- Current approach that isn't working so well
    - Each library to have its own section - using directives to section off the libraries
    - There would more libraries, do not want to use more directives, to avoid clutter
- Request
    - Are there makefile experts who can help?
        - *(Taylor Blau): Are we assuming there exist Makefile experts in the first place? ;-)*
- (Jonathan) do you have an example?
    - (Calvin) using 'ifdef GIT_STD_LIBRARY' to tweak what goes in LIB_OBJS. This approach doesn't scale.
- (Peff) for every C file you have two copies?
    - No, for every reference they are using the same file
- (Junio) libgit.a will we need something different, if so, why?
    - Stubs, how do they come into play?
    - If we had a makefile for a library, we're trying to understand how we have a subset
- (Jonathan) Do I end up with two different .o files?
    - Yes, there is a subset of shared and not shared files
    - Some of the objects are the same, the stubs are different.
    - The problems are the stubs which are shared
    - Example: with and without support for gettext
- (Calvin?) ideally we want the .o files to be the same
    - Yes
- (Peff) if you are worried about writing the same rules again and again, there should be a solution
    - Yes, it will likely have to be a custom pattern
    - Git uses GNU make - can generate parts with eval code, but it might be unreadable solution
    - Does anyone have a solution that has worked before? A simple solution? OR our own custom templating
- (Phillip) Can we build the file in standard git so we're not creating the file for two different libraries?
- (Emily) if we are changing the behavior using standard git builds and library builds……
- (Jonathan) in the past other projects used recursive "make" to reflect module structure in the build structure, which has lots of downsides (Peter Miller, *Recursive make considered harmful*).
    - We can define our own structure of what we want the Makefile to look like. Linux kernel files are perhaps a good example. There's not necessarily one standard everyone follows, it tends to be very codebase specific
    - For better or worse, "make" is a "build your own" kind of build system

- (Emily) why are we not using a different build system? Such as CMake
    - What are the technical reasons for make?
        - inertia
- (Junio) How do the libraries and make relate to each other? Avoiding compile-time conditional behavior seems desirable anyway - git as a consumer of the library can also benefit from cleaner error handling.
    - (Emily) cleanup related to the library might mean moving `exit()`/`die()` closer to the built-in. Do we consider that potentially valuable instead of useless churn?
        - (Junio) yes
- (Jakub) It's easier to die when something is wrong at the top level
    - (Peff) It depends on what level of error handling we want to get to. The reality of C is every single malloc can fail. Do we need to check every error?
    - (brian) standard error handling mechanism would be helpful.
    - (brian) Rust and Go also die on allocation error, but have better error handling support than C, relying on `errno` as Git does is brittle; perhaps in-out parameter for error handling?
- (Emily) for libgit2 does the caller handle the memory?
    - (brian) a dummy value (`git_buf_oom`) pointing to pre-allocated value where you can check if it's out of memory
        - (Jakub): does it affect performance due to branching? This needs checking.

Scaling Git from a forge's perspective

Presenter: Taylor Blau, Notetaker: Karthik Nayak

- Things on my mind!
- There's been a bunch of work from the forges: GitHub, GitLab, Google, over the last few years - bitmaps, commit-graphs, multi-pack indexes (midx). etc.
    - Improving performance on server side
- Q: What should we do next? Curious to hear from everyone. Including Keanen's team
- **Idea (1)** Boundary-based bitmap traversals, already spoke about it last year. If you have lots of tips that you're excluding from the rev-list query. Backlog to check the perf of this.
    - Patrick: still have not activated it in production. Faced some issues (performance regressions) the last time it was activated. We do plan to experiment with this (https://gitlab.com/gitlab-org/gitaly/-/issues/5537)
        - Move to Roaring Bitmaps (https://roaringbitmap.org), *one of not quite finished GSoC projects (did cleanup, not had time for full Roaring)* have not improved things, EWAH might be best for us
    - Taylor: Curious of the impact.
    - In almost all cases they perform better, in some equal and very few worse.
- (Jonathan Nieder) Two open-ended questions:

        - Different forges run into the same problems. Maybe its worth comparing notes. Do we have a good way to do this? In Git Discord there is a server operator channel, but only two messages.
            - Taylor and Patrick have conversations over this via email exchange.
            - Keanen: Used to have a quarterly meeting. Attendance is low.
            - From an opportunistic perspective, when people want to do this, currently seems like 1:1 conversations take place, but there hasn't been a wider-group forum
            - Server operator monthly might be fun to revive
            - Git Contributor Summit is where this generally happens. :)
        - At the last Git Merge there was a talk by Stolee about Git as a database and how as a user that can guide you in scaling. Potential roadmap for how a git server could do some of that automatically. Potential idea? For example, sharding by time? Like gc automatically generating a pack to serve shallow clones for recent history.
            - Extending cruft-pack implementation to more organically have a threshold on the number of bytes. The current scheme of rewriting the entire cruft-pack might not be the best for big repos - big I/O hit.
            - Patrick: We currently have such a mechanism for geometric repacking.
- (Taylor Blau) **idea (2)**. Geometric repacking was done a number of years ago, to more gradually compress the repository from many to few packfiles. We still have periodic cases where the repository is reduced to 2 packs, one cruft and one of the objects. If you had some set of packs which contained disjoint objects (no duplicates), could we extend the verbatim packs to work with these multiple packs. Anyone had similar issues?
    - (Jonathan): One problem is whether to know if a pack has a non-redundant reachable object or not without worrying about things like TTL. In git, there is "push quarantine" code, if the hook rejects it, it doesn't get added to the repo. In JGit there is nothing similar yet, so someone could push a bunch of objects, which get stored even though they're rejected by a pre-receive hook. Which could end up with packs with unreachable objects. With history rewriting we also run into complexity about knowing what packs are "live".
        - (Patrick): Deterministically pruning objects from the repository is hard to solve. In GitLab it's a problem where replicas of the repository contain objects which probably need to be deleted.
        - (Jeff H): Can we have a classification of refs which makes classification possible wherein some refs are transient and some are long term (like 'main' vs feature branches).
            - (Jeff King): There are a bunch of heuristic inputs which can help with this. Like how older objects have a lesser chance of change vs newer. Area for exploration.
            - (Taylor): Order by recency, so older ones are in one bitmap and newer changeable ones could be one clump of bitmaps.

- (Minh): I have a question about Taylor's proposal of a single pack composed of multiple disjoint packs. Midx can notice duplicate objects. Does that help with knowing what can be streamed through?
    - (Taylor): The pack reuse code is a bit too naive at this point, but conceptually this would work. We already have tools for working with packs like this. But this does give more flexibility.
- (Taylor): **idea (3)**. GitHub recently switched to merge-ort for test merges *(see Optimizing Git's Merge Machinery series of blog posts)*, tremendous improvements, but sometimes creates a bunch of loose objects. Option to have merge-ort to side step loose objects (write to fast-import or write a pack directly)?
    - Things slow down when writing to the filesystem so much.
    - (Jonathan Tan): one thing we've discussed is having support in git for a pack handle representing a still-open pack file that you can append to and read from in the context of an operation.
    - (Dscho): that sounds like the sanest thing to do. There's a robust invariant of needing an idx for the pack file that you need for working with it efficiently, which requires the pack file to be closed. So there are some things to figure out there, I'm interested to follow it.
    - (Junio): There was a patch sent to list to restrict the streaming interface. I wonder if that moves in the opposite direction of what we're describing
    - (brian): In sha256 work I noticed it only *currently* works on blobs. But I don't think adapting it to other object types would be a major departure. As long as we don't make the interop harder, I don't see a big problem with doing that. Conversion happens at the pack-indexing time.
        - Enhance streaming interface to write trees and commits
        - Smaller amount of non-blob objects
    - (Elijah): Did I understand correctly that this produces a lot of cruft objects?
    - (Dscho): Yes. We perform test merges and then no ref points to them.
    - (Elijah): Nice. "`git log --remerge-diff`" similarly produces objects that don't need to be stored when it performs test merges; that code path is careful not to commit them to the object store. You might be able to reuse some of that code.
    - (Dscho): Thanks! I'll take a look.

Replacing git LFS using multiple promisor remotes

Presenter: Christian Couder, Notetaker: Jonathan Nieder

- Idea: Git LFS has some downsides
    - Not integrated into Git, that's a problem in itself
    - Not easy to change decisions after the affect about what blobs to offload into LFS storage
- So I started work some years ago on multiple promisor remotes as an alternative to Git LFS
- Works! Requires some pieces

    - Filtering objects when repacking (`git repack --filter`, due to be merged hopefully soon)
- I'm curious about issues related to Git LFS - what leads people not to use Git LFS and to do things in other, less efficient ways?
- Choices
    - We can discuss details of a demo I worked on a few years ago
    - We can discuss Git LFS, how it works, and how we can do better
- (brian): Sounds like this is a mostly server-side improvement. How does this work on the client side for avoiding to need old versions of huge files?
    - (Christian): On the client side, you can get those files when you need them (using partial clone), and `repack --filter` allows you to remove your local copy when you don't need them any more
    - There could be more options and commands to manage that kind of removal
    - Need some tool to manage multiple promisor remotes and check what they have
- (Terry): with multiple promisor remotes, does gc write the large files as their own separate packfiles? What does the setup look like in practice?
    - (Christian): You can do that. But you can also use a remote helper to access the remotes where the large files live. Such a cache server can be a plain http server hosting the large files, and the remote helper can know how to do a basic HTTP GET or RANGE request to get that file.
    - It can also work if the separate remote can be a git remote, specialized in handling large files.
    - (Terry): So it can behave more like an LFS server, but as a native part of the git protocol. How flexible is it?
    - (Christian): yes. Remote helpers can be scripts, they don't need to know a lot of things when they're just being used to get a few objects.
- (Jonathan Tan): Is it important for this use case that the server serves regular files instead of git packfiles?
    - (Christian): not so important, but it can be useful because some people may want to access their large objects in different ways. As they're large, it's expensive to store them; using the same server to store them for all purposes can make things less expensive. E.g. "just stick the file on Google Drive".
- (Taylor): In concept, this seems like a sensible direction. My concern would be immaturity of partial clone client behavior in these multiple-promisor scenarios
    - I don't think we have a lot of these users at GitHub. Have others had heavy use of partial clone? Have there been heavy issues on the client side?
    - (Terry): Within the Android world, partial clone is heavily used by users and CI/CD and it's working well.
    - (jrnieder): Two qualifications to add, we've been using it with blob filters and not tree filters. Haven't been using multiple promisor remotes.
    - (Patrick): What's nice about LFS is that it's able to easily offload objects to a CDN. Reduce strain on the Git server itself. We might need a protocol addition here to redirect to a CDN.

- (Jonathan Tan): if we have a protocol addition (server-side option for blob-only fetch or something), we can use a remote helper to do the appropriate logic, not necessarily involving a git server
    - The issue, though, is that Git expects packfiles, as the way it stores things in its object store.
    - As long as the CDN supports serving packfiles, this would all be doable using current Git (lazy fetch).
    - If the file format differs, it may need more work.
- (jrnieder): Going back to Terry's question on the distinction between this and using an LFS server. One key difference is that with git LFS, is that the identifier is not the object ID, it's some other hash. Are there any other fundamental differences?
    - (Christian): With git LFS if you want some blobs to be stored with LFS and they're not stored in LFS anymore you have to rewrite the history. Inflexible in that area.
    - Using the git object ID gives you that flexibility
- (brian): One thing Git LFS has that Git doesn't is deduping
    - On macOS and Windows and btrfs on Linux (COW, Copy-on-Write filesystems), having only one underlying copy of the file
    - That's possible because we store the file uncompressed
    - That's a feature some people would like to have some time. Not out of the question to do in Git, would require a change to how objects are stored in the git object store
- (jrnieder): Is anyone using the demonstrated setup?
    - Christian: Doesn't seem so. It was considered interesting when demoed in GitLab.
- (Jonathan Tan): Is the COW thing brian mentioned part of what this would be intended to support?
    - (Christian): Ultimately that would be possible.
    - (brian): To replace Git LFS, you need the ability to store uncompressed objects in the git object store. E.g. game textures. Avoids waste of CPU and lets you use reflinks (ioctl to share extents).
    - (Patrick): objects need the header prefix to denote the object type.
    - (brian): Yes, you'd need the blobs + metadata. That's part of what Git LFS gives us within GitHub, avoiding having to spend CPU on compressing these large objects to serve to the user.
- (jrnieder): Going back to the discussion with multiple promisors. When people turn on multiple promisors by mistake, the level of flexibility has been a problem. This causes a lot of failed/slow requests - git is very optimistic and tries to fetch objects from everywhere. This suggests the approach that Jonathan suggested, where the helper is responsible for choosing where to get objects from, it might help mitigate these issues.
    - (Christian): yes
- (Minh): can the server say "here are most of the objects you asked for, but these other objects I'd encourage you to get from elsewhere"?
    - (Christian): you can configure the same promisor remote on the server. If the client doesn't use the promisor remote and only contacts the main server, the

server will contact the promisor remote, get the object, and send it to the client. It's not very efficient, but it works. Another downside is that if this happens, that object from the promisor remote is now also on the server, so you need to remove it if you don't want to keep it there.
- (Minh): it seems someone has to pack the object with the header and compute the git blob id for it, which is itself expensive
- (Christian): if the promisor remote is a regular git server, then yes, the objects will be compressed in git packfile format. But if it's a plain HTTP server and you access it with a helper, it doesn't need to. But of course, if the objects are ever fetched by the main server, then it's in packfile or loose object format there.

Day 2

Clarifying backwards compatibility and when we break it

Presenter: Emily Shaffer, Notetaker: Taylor Blau

- (Emily) In the last year, there were a handful of scenarios where we had issues with backwards compatibility.
    - E.g. config based hooks. As a result of this change, text coloration was broken from user-provided hooks. Missing tests, but still viewed it as backwards-compatibility-breaking.
    - E.g. deleted internal "`git submodule--helper`" that looked like a plumbing command, which other projects depended on. Was being used in the wild, even though we didn't expect it.
    - E.g. bare repository embedding. Interested in fixing that as a security concern (https://offensi.com/2019/12/16/4-google-cloud-shell-bugs-explained-bug-3/, https://github.com/justinsteven/advisories/blob/main/2022_git_buried_bare_repos_and_fsmonitor_various_abuses.md). Weren't able to do so in a widespread fashion, since many projects are using it for testing setups (i.e. test fixtures).
- (Emily) When do we consider odd behavior as bugs, versus when do we consider them as something that's part of our backwards compatibility guarantee.
- (Emily) What do we want backwards compatibility to look like for libraries? How do we want to handle this in the future?
- (Minh) Is there documentation on how this should behave?
    - (Emily): Typically try to guarantee backwards compatibility via integration tests. Have changed documented behavior in the past when it seems "just wrong". Is the documentation the source of truth?
    - (Jonathan Nieder): In the case of browsers, using a specification for compatibility is a useful tool. What will work at the same time across different implementations? When there is a single implementation (e.g. git) it is easier to capture your intention with the implementation instead of a specification.

    - (Jonathan Nieder): Converting that documentation into a specification can hurt readability or inhibit its other uses.
    - (Junio): Tend to ensure that observable behavior is locked in via integration tests. Tests are the source of truth, along with implementation. Documentation is often lying. Unlike the browser example, we don't have an external specification to rely on. Intention from developers is captured in the log proposed message.
- (Minh) Should we be testing more, then?
    - (Emily) That's part of it, but some older behavior (e.g. from the original implementation) has less information in the commit message as a result of project culture at the time.
    - (Junio) Working-as-designed, but the design outlived its usefulness.
    - (Jonathan Nieder) E.g. dashed commands, i.e. '`git-add`' versus '`git add`'. Outcry after we changed behavior, so we rolled it back. Much later we got to a place where people weren't relying on this behavior as much. Removing it simplifies installer, and takes less space on MS Windows.
- (Jonathan Nieder) There is another kind of documentation besides specification. E.g. the kernel has a documented guarantee about compatibility: "the kernel never breaks userspace". This doesn't mean that we can't have observable behavior changes. Only that they maintain "depended-upon" behavior that the kernel is reasonably responsible for providing. Can only determine this surface area by rolling out changes and seeing if folks complain.
- (Jonathan Nieder) It sometimes feels like we have adopted a similar philosophy, but the kernel has an easier job since POSIX, System V, etc have defined the overall shape of the syscall interface.
- (Elijah) Difficult to distinguish between bug fixes and breaking backwards compatibility. When we break existing test cases, document a rationale for doing so in the proposed patch message. Cases where documentation was just wrong. Often comes down to a judgment call.
- (Minh) Is the consensus to keep tests up-to-date, and add more tests when behavior is unclear?
- (Jonathan Nieder) Problem is that there can be differences of opinion on what are safe compatibility guarantees to break.
- (Emily) Also the case that there are tests that are in the integration suite that are enforcing things that weren't meant to be compatibility guarantees. E.g. enforcing error messages. How do we cope with legacy behavior and legacy tests when making a judgment call? There is some documentation in general about backwards compatibility, plumbing commands are frozen, porcelain commands are not. Should we expand that documentation to clarify how to decide?
- (brian) This would be helpful, but not sure what it would look like. Kernel's approach may be too rigid for Git. Sometimes useful to break backwards compatibility. E.g. "we have it, but it isn't a good choice." Users depend on those error messages. When we make a change that is overwhelmingly beneficial, we can't please everybody all of the time.
- (Jonathan Nieder) Back to guarantees for library code. Kind of view the plumbing/porcelain decision as a failed experiment. Of course scripters are going to use

the plumbing. Want a better backwards compatibility guarantee. People are going to want to add more functionality there and lock in additional behavior. When people write scripts, they write using the commands that they understand how to use. End-user commands gain for-scripting functionality, like git log's `--format`.

- (Junio) Worse is when new features are added only to porcelain, and plumbing code is left behind.
- (Jonathan Nieder) In a way, we made it harder on ourselves. If porcelains are written as scripts, you need plumbing commands to expose the functionality they need. Now porcelains use function calls, so the well maintained interface is more on the (internal) library side
- Libification moves us in a good direction, since it provides an alternative to the CLI as a well-defined programmatic access method.
- (Jonathan Nieder) If we succeed at this, the command-line backwards compatibility guarantee for porcelain commands can break down a bit to the extent that users start to adopt the library code as their interface to Git.
- (Emily) If we have suitable replacements in the library, can we deprecate the plumbing variant of that functionality eventually? Freeze a particular plumbing command instead of adding to it
- (Taylor) Can't break existing behavior, shouldn't have to force users to upgrade to library code for existing behavior. Apologies if this is what you were saying.
- (Jakub) Auto-generated CLI shim, like cloud providers often provide for their APIs?
    - *(Jakub): A comment about library -> plumbing helpers; we have a lot of test helpers that expose internal functions for testing (some due to no having unit test)*
    - *(nasamuffin): @Jakub - take a look at https://lore.kernel.org/git/CAJoAoZmC-Ku80aSp5xkxfirJitoi_ai6ryK3cJ7Jkx_2fUjENw@mail.gmail.com/ - i think we can move most of these to use the unit test framework Josh Steadmon is working on, when that goes in*
- (Jonathan Tan) Might be hard to create scriptable interfaces for library commands. Library allows us to pass pointers and function callbacks, neither of these we can accomplish via the shell.
- (Minh) Is there an understanding that the library has to implement 100% of the functionality of plumbing commands?
- (Emily) Not convinced that we need a one-to-one match between the library and command-line interface. Want to expose the same intent, not necessarily exact incantations.
- (Jonathan Nieder) Let me try and summarize. Question resonates with people, no one has a silver bullet. Maybe some agreement for using more tests, but the general approach to figuring out our compatibility guarantees remains an open discussion.
- *(brian, via chat) One final thought: maybe we could look at what Semantic Versioning defines a breaking change as, since they've defined this in a very public way.*
- *(Phillip, via chat) Thinking back to yesterday there were people saying that they chose the cli over a library because of concerns about memory leaks and the library*

crashing/dying as well as licensing concerns. If we were to add new functionality only in libraries we'd need to make sure that they were robust.

-

Authentication to new hosts without setup

Presenter: M Hickford, Notetaker: Lessley Dennington
Slides: Authentication to new hosts without setup

- (Hickford) I interact with many Git "hosts" (GitHub, GitLab, gitlab.freedesktop.org, etc.). I had 15 Personal Access Tokens (PATs) around, which was tedious. I was using Git Credential Manager, which has an option to authenticate via web browser which creates a token. I released git-credential-oauth with this feature which you can use with a storage helper. I'm going to show an example of authenticating to a host I've never used before (Gitea). *Demonstrates signing into Gitea via web browser and cloning his fork of project xorm/xorm. Since the repo is public, no authentication is necessary. Makes a commit and pushes. Auth flow is triggered, provides consent. Authentication was successful.* There was no need for PATs or shell keys. Git-credential-oauth supports GitHub, GitLab, Gitea, and Gitee out of the box. Works using new(ish) `password_expiry_utc` attribute and `wwwauth[]` headers.
- (Brian) Thinks it's a great idea because it's convenient. github.com/github requires SAML/SSO and the browser, and this should work just fine. It wouldn't be great to have in C, but as a helper it's super convenient.
- (Hickford) Ruled out a C implementation due to the challenges. Goal was to remove a barrier to entry for contributors to OSS trying to make bug fixes and having to set up/deal with PATs/SSH keys.
- (Jakub) Still work to do with creating a fork, pushing.
- ~~(Brian) GCM does this but represents a greater barrier to entry for less Git literate users. Less beneficial for Git power users.~~
    - *Edit: Lessley and Brian spoke after the meeting, and Lessley realized the above was not recorded correctly.* `git-credential-oauth` *and GCM both remove the need for users to manually set up PATs/SSH keys (which was what was being considered as the high barrier to entry).*

Update on jj, including at Google

Presenter: Martin von Zweigbergk, Notetaker: Glen Choo

- (Martin) jj team at Google has been growing. The support for different commit "backends" has been expanding - we can now store "commits in the cloud" using the Google-internal backend.
    - "Revset" engine. Revset is a language for selecting commits (e.g. "select all commits by me"). We now have an implementation that scales to Google's

millions of commits. Commit id prefixes are resolved against the “local” commits (not the full Google mainline).
    - Conflicts are now stored tree-level, instead of per-file level. Conflict detection is much faster since jj doesn’t need to traverse the tree.
    - Exporting jj commits to internal code review tool (Critique).
- (Martin) What’s left?
    - Renames: do we track renames? Do we detect them?
    - No support for detecting renames and copies yet, is needed especially for large rebases with very large number of commits
- (Elijah) If conflicts are tree-level, can you store partially-resolved conflicts?
    - (Martin) Yes, we store trees for each side of the conflict and resolve the conflicts only when needed.
- (Jrnieder) Are there lessons from jj development that Git would benefit from? What can Git do to make jj’s life easier, and vice-versa?
    - (Martin) Conflicts-in-code work extremely well. I think Git could adopt that, but it would be very messy to migrate the UX to that. The operation log (a global view of all of the refs at a given “operation”) is also a big improvement over e.g. the reflog.
    - (Martin) jj uses libgit2 (with Rust bindings) under the hood, so we’re missing functionality like partial clone.
    - (Taylor) do you shell out to git, or only use libgit2? If you did shell out, are there other missing Git functions that you’d want?
        - (Martin) Only libgit2. Can’t think of other features jj would want.
        - Until merge-ort existed, worktree-less merge would be an example.
        - (Glen) When jj pushes things to a Git server, it loses information. If the server understood obsolescence markers, that would be a huge improvement for jj.
        - (Martin) Yes, jj uses a change-id to associate different amended versions of the same change, similar to Gerrit - it would be nice for Git to support the same thing.
- (Junio) Did you have to make any breaking changes that affect your users?
    - (Martin) We make many. We’re a small project, and people accept that it needs to break to get a nicer UX, which is a nice thing about being early in a project.
    - Format-wise, we try not to break the repo format - in terms of newer versions of jj being able to work with older versions of repositories. Older versions of jj are not expected to always be able to read repos written to by a newer version.
        - (Jonathan) “svn upgrade” style?
        - (Martin) Yes, except we immediately do the upgrade automatically.
        - (Jonathan) So the moment you use the new version of jj, you lose the ability to roll back.
        - (Martin) Yes. Errors out (crashes) when reading the format it doesn’t understand.
        - One of these was annoying for users, we may be at the point where we need something more formal.

    - (Junio) In 2005, we did two huge breaking changes in the repo format. There were lots of users, but we did it anyway. One was about object naming (used to compress first, then hash, which was a bad way of doing it - swapped the order to compress better and faster without changing object names).
- (Elijah) If we rewrote parts of Git in Rust, would we be able to share code?
    - (Martin) Possibly, but it would require a lot of rewriting to make that work.
- (Emily) Greenfield features in jj, e.g. preventing users from rewriting “public” commits/history. Are there other ideas would we like to try in jj that are harder to do in Git?
    - concept of https://wiki.mercurial-scm.org/Phases makes some things (like safe interactive rebase) easier
    - (Terry) The usual practice is to have policies on branches (some branches are more experimental, some have stringent quality requirements, etc), but those are implemented on the hosting provider, not the VCS.
- (Terry) jj has lots of glowing reviews! Power users are happy with it, using jj locally. If anything is not supported in jj, they can use Git instead. Is there a roadmap for simplifying the experience for non-power users, having it automatically take care of things like when to run gc, etc?
    - (Martin) Re: gc, jj doesn’t implement it yet.
    - (Terry) More general UX. If I’m a developer using git repositories and want to use jj, when do I get to a place where I have a nice end-to-end workflow?
    - (Martin) I already use jj, I don’t have the “colocated repo” so I only run jj commands, can’t run git commands. For blame I fall back to the hosting provider’s web UI. :) That’s something to add.
    - (jrnieder) My impression from the jj discord is that the UX is very dependent on their code review tool. Amending/rebasing and sending to GitHub seems to work OK. Losing the obsolescence information when pushing to Gerrit works quite poorly.
- (Minh) Does jj store commits in Git form? Can it translate between different commit representations?
    - (Martin) It can store commits in Git form. The demand for on-the-fly conversion has come up.
- (Taylor) How does jj represent non-Git concepts in Git format, like having multiple trees in a commit?
    - (Martin) It stores extra metadata outside of the Git commits in a separate storage, and also it stores its own shape in Git format, e.g. for multiple trees during conflict, each tree is its own directory `.<stage>`.
- (Minh) How do you optimize searches like “commits written by me”? Full text index?
    - (Martin) It’s implementation-specific. On local repos, it just iterates commits.
    - (Martin) The revset language is quite expressive, e.g. you can specify AND and OR. The language is also separate from implementation. Pickaxe search (searching in changes) not supported in jj yet, not indexed.
    - Git acquired commit-graph, bitmaps, multi-pack index, Bloom filters - which can be considered indexes to speed up search, but there is no generic index

- (Jakub) There are other tools that implement a query language like SQL for Git. It could be worth considering implementing one natively. (See Git Rev News archives.)

Code churn and cleanups

Presenter: Calvin Wan, Notetaker: Taylor Blau

- Question: When is refactoring worth the churn? The refactoring may or may not contribute to a different goal (e.g. libification). Other factors:
    - Should those refactor series be included with the feature?
    - Should they be split up?
    - Do they make sense as isolated units?
- Some examples: Elijah's `cache.h` cleanup series, which was obviously good. Others of dubious value.
- (Elijah) May have done the `cache.h` series a year or two earlier, but wasn't sure that it was obviously good.
- (Jonathan Tan) First have to define the churn. Two kinds:
    - Having reviewers look at it in the first place, since there are no objective user-facing improvements.
    - Causes additional toil in revision history.
- (Jonathan Tan) Let's start with reviewer churn. What constitutes "good" or "clean" code is subjective, so authors and reviewers may spend a large amount of time debating whether or not the refactoring meets that criteria. Can be avoided when the feature is on top in the same series.
    - (Junio) Speaking cynically: the new feature may be taking a subjective change or rejection of it hostage.
    - (Calvin) In other words, refactorings are of lower value than features?
    - (Junio) After you implement some features, you may discover opportunities for clean-up after the fact.
    - (Jonathan Nieder) In the course of solving a given problem, may come up with a lot of different changes that all help. If you generate a long patch series, you are over-constraining the maintainer in determining how to slot those changes in. Also makes applying to a maintenance branch, rolling back particular pieces harder, etc harder.
        - If I make a one-line bug fix and notice "this code was hard to understand, here's a refactoring that makes it more obvious", it's often more helpful to the project for the one-line bug fix to come *first* in the series and the refactoring to be a followup or later part of the series.
    - (Taylor) One thing that helps is *motivating* a refactoring. Saying "here's what this refactoring makes easier".
    - (Martin) What *is* "refactoring for its own sake"? For example, is removing global state something that we want without additional justification?

- (Emily) Can we split the difference? Can we send cleanup patches with less context? With more context? Should we be better about committing to a feature and presumptively merging clean-up patches along the way.
- (Junio) I rarely hear from reviewers the signals that would allow me to do this. “I have reviewed this series, and patches 1-4 look ready, I’d be happy with those landing and being built on top of”.
- (Emily) Could change our habits to add “LGTMs” part of the way through the series.
- (Jonathan Tan) We often need to add a feature to “sweeten the deal”. The feature proves that the refactoring is good. Doesn’t add to the overall value, but makes it cost less to review the refactoring. Perhaps that the presence of the feature is proof enough, even if it isn’t merged right away.
- (Terry) Sounds like the question is, “what is the value proposition for refactoring?” Usually to lower tech debt. Challenge: maybe every refactoring should stand on its own?
    - In implementing a feature, I might notice “the object database interface is causing problems in this way”. Then my cover letter can spell out those problems and how the refactoring addresses them.
    - It’s hard work to describe why something isn’t good, especially in a legacy codebase with some tech debt and some old changes missing clear commit messages. It’s work but I think it’s worthwhile. It builds an understanding in the community of how that subsystem should function.
- (Elijah) My series might be an example of that, didn’t have a feature associated with it. Helped with libification effort, and started with a smaller series to illustrate the direction. Guessing that there are certain types of refactoring that we already consider to be good.
- (Jonathan Nieder) Could having a wiki page that lists helpful refactorings that would be likely to be accepted on their own?
- (Jonathan Tan) I’d like to challenge Terry’s challenge. It’s a laudable goal, but a subsequent patch implementing the feature is worth 1,000 words.
- (Jonathan Nieder) If we want to be doing more refactoring, then we’re going to have to develop different skills as developers and reviewers. Reviewing refactoring is more like reviewing technical writing. Having examples to illustrate the idea can help, even if those examples are changes that aren’t changes we want to make right now to Git.
- (Terry) Some people are visual learners, some people are auditory learners, and so on. Having a change in place on top of a refactoring is worth 1,000 words. But if you write well, maybe you don’t need the latter patch.
- (Taylor) I think I agree with both these things - I like having the documentation and explanation, but I also see Jonathan Tan’s point about examples being helpful.
- We should become more comfortable with throwing away work. Suppose I’ve made a refactoring and we decide not to ship the change it was meant to support. Is it worth the reviewer’s time to take anyway?

        - We need to make the cover letters clearer, make the case for it being worth the time.
    - (Calvin) I think I agree with Taylor. To re-describe: our cost is code churn and reviewer time. Feature patches show that there is a 100% guarantee the preceding changes are worthwhile. There is a discount factor when you don't have a feature to illustrate the value. Authors need to be more clear when there doesn't exist a feature patch on what the value is.
        - Reviewers can encourage the author to give better examples of how the change will pay off.
    - (Glen) Are there things we could do to help newer contributors in this regard? Should we have a more opinionated style guide?
        - (Taylor) Separate CodingGuidelines into semantic requirements and more subjective "here are kinds of refactorings we like"
    - (Jonathan Nieder) For newer contributors: better/more worked examples of how experienced contributors justify their refactoring changes. E.g. "here are some series in the past that were harder to review because of the lack of this change". If people had examples to emulate, they would be doing it more.
    - (Emily) Difficult to synthesize commit messages without examples, especially for non-native English speakers, people who aren't great writers, etc.
        - *(nasamuffin): i mean, how many of us got into programming because we hated writing essays in school? ;)*
    - (Emily): there is Review Club
- (Jonathan Tan) The other kind of churn in looking back at history and seeing what has happened in the file. One thing I worry about is that there may be another feature in the future that forces us to partially or entirely revert the refactoring. That reduces the probability of the refactoring being "good" in the first place.
- (Terry) Emily's point about inclusivity: that work (writing a persuasive essay, emulating examples) is tedious and difficult, it may not be natural to everybody. As a project, we should be creating those examples. Reviewers should help newer contributors along the way.

Project management practices

Presenter: Emily Shaffer, Notetaker: Jonathan Nieder

- high hopes and low expectations! Let's play nice
- Project management related tools and practices we may be used to from $DAYJOB
    - Bug trackers
    - Long-term and short-term project planning
    - Roadmaps
    - Documented project direction
    - Tools, like wiki
- Some things other open source projects do

    - Some do things like two-week sprints with contributors, more structured development of some features
- Some things that happen in regular workspaces
    - Ad hoc whiteboarding through a problem
    - Live chat with history
        - We have IRC but it's dwindling outside of IRC standup: irc://irc.libera.chat/#git
            - *(Jakub): Is IRC channel logged? I think it is, but https://git-scm.com/community does not include link to IRC logs*
            - *(Taylor): @jakub it is, see: https://j.mp/gitdevlog*
                - *Listed on https://gitirc.eu/ - linked in #git channel description*
        - There's informal Discord: https://discord.gg/bMmx8bt8Cs (and https://discord.gg/GRFVkzgxRd)
- Lots of tools! Are there ones we want to benefit from?
- Example: bug tracker
    - Lots of interest, but don't have a shared understanding of what we want from a bug tracker
- If you could pick something from your day job and apply it to the Git project, what would you look for and what would you want from it?
- (Taylor) "Let's have a quick VC"
    - All being at the same organization within GitHub, people are very willing to just jump into a Zoom meeting and talk through a thing, whiteboard
    - There's benefit to having things documented on-list, but I think we could walk that back a little
    - (Emily) One thing I've liked with the contributor summit and similar events is sending something to the list so people who weren't there can still follow and respond
    - Would "Taylor and I just talked about cruft packs" emails be something we want more of?
    - (Taylor) Yes and no. Sometimes a conversation is for getting ideas, sometimes for making a decision. They deserve different approaches.
    - (Emily) By the way, I've been surprised at how open people are to VCing when I try it. Conversations about cruft packs, conversation about config based hooks series, tried "let's have a VC" and Ævar was very open to it in that example and it worked well
    - So it might be something we should just try more often
- (Emily) At a Git contributor summit, some attendees mentioned wishing they could see a published Git project direction
    - Various companies are putting dedicated time into the Git project, and those aren't published anyway
        - Page with "GitHub cares a lot about cruft packs, here's how you can help"
        - Is that something we could write down somewhere more static than the mailing list?

    - (Junio) How quickly would that become stale?
    - (Emily) It depends on how you build it into your own processes. Every quarter we write quarterly objectives and key results for my team, publish that to everyone at Google. I could also publish that to the Git mailing list, part of that normal process could be posting it on a Wiki page.
    - (Jonathan Tan) One thing I'd find difficult in publishing such a thing is understanding the priority of line items. Currently I learn about people's priorities from patches and from what they say in venues like a contributor summit
        - Contributor summit is once/year, if you're saying something, it's probably important to you
    - Things that change once/quarter are harder to judge. How much are you dedicating to them?
    - (Taylor) Suppose this existed. What would people use it to answer? Mutex?
    - (Emily) There have been some good cross-company collaborations in the past, such as partial clones. Noticing the opportunity to work together on such things is the kind of thing I'm thinking of.
    - (Taylor) Back in 2014-15, GitHub had an awesome tradition of there being a "radar issue", people could comment on this big long thread on what they want to hack on.
        - I think that's a little different than publishing a committed roadmap, with pressure and accountability. "What is Jonathan Tan interested in"
        - Could be as simple as every quarter we send an email to the list and all reply to it.
    - (Jakub) We can attach a roadmap to the same place Git Rev News is, you can get news and a roadmap in the same place.
    - (jrnieder) Sharing your plans and priorities helps people know what they can expect you to care about. E.g. if your work is all reliability, all the time, maybe a new UX change is not as exciting right now, versus reliability focused work is a good place for collaboration.
    - (Calvin) Having timestamps, e.g. a pinned message on Discord, helps you know if something is stale.
        - (Emily): Rolling thread of what we work on, can be done with wiki + timestamp, cleanup aftera year
    - (Jeff Hostetler) Make a repository, publish there "I'm working on this". Send a pull request, get feedback. Nice and compact, has timestamps, stays in our ecosystem.
        - Well known spot
- (Emily) I don't like the trend of projects being only managed on discord. *But:* I'm wondering, what changes would make the git community discord more of an official channel in the same way as the git mailing list is?
    - https://discord.gg/aUCkDVUqqu
    - (Elijah) There's a Git Discord?
    - (Taylor) We just needed to make Elijah aware of it. ;-)

    - (Jeff Hostetler) I think Discord is a bit childish, git repos are something professional we all use every day.
    - (jrnieder) There's a bit of a shift IRC -> Slack -> Discord in a lot of projects
    - (Emily) A big benefit of IRC is that it's a decentralized protocol. Having a part of our infrastructure be a centralized, non transferable thing is scary to me, but maybe there are technical ways to address that. Export logs, matrix bridge to IRC, ?
    - (Taylor) I think a barrier to use of chat can be fear of *decentralization* of information, it's convenient that the git mailing list is a one-stop shop. Discord can dilute importance of git mailing list.
        - (Jeff Hostetler) +1, having too many things to check
        - (Emily) I think this is also why we're hesitant about other things like bug trackers etc
- (Jonathan) Bug tracking
    - (Emily) This year we moved crbug.com/git (Monorail) to git.issues.gerritcodereview.com. There's 80ish issues there. Our team within Google uses it. But of course in reality no one else is making use of that issue tracker. If there were somewhere else to put bugs instead, we'd use it - I don't think it's too important where that is, as long as we can do it somewhere.
    - (Junio) Someone needs to curate it.
    - (Emily) It would be possible for us to curate, triage git.issues.gerritcodereview.com if people start using it.
    - (Junio) Not limited to bugs, but we from time to time talk about other aspects of tracking. Things like patchwork. We talk about mechanisms, but not so much about enforcing *use* of those mechanisms.
    - One work practice I like at work is that anyone can write a CL, and then people are *forced* to review or look at the patch in a reasonable amount of time.
    - It can be frustrating as a maintainer, because I don't want to be reviewing and looking at all the patches on the list myself. And I don't like having to queue patches not looked at by anybody.
    - (Emily) This makes me wonder if we should be having conversations about things like "whose turn is it to take action on this patch".
-

Improving new contrib onboarding

Presenter: Jonathan Nieder, Notetaker: Xing Huang, Ronald Bhuleskar

- (Jonathan Nieder) Not as structured of a conversation, but I see a lot of interest, let's see how the conversation goes. Any open sourced project can be scary for newcomers; the git project in particular has its unique aspects of its workflow, such as the mailing list that rejects http formatted mails, etc. I think overall we are welcoming. Ideally, we would like to attract all types of contributors, in part because they help different kinds of users have more of a voice.

- I am interested in how to make the onboarding process easier for the new contributors; what do we see to make things easier? MyFirstContribution works well as a tutorial doc, what is the next step for someone after they send their first patch and get their first review in reply? How do you find a mentor? Things like how to interpret the reviewer's tone can be hard to navigate.
- (Emily) We can mark a patcher as a beginner's patch - the golang (?) project, for example, assigns a mentor to newcomers. We have a mentorship list that's inactive; could we use the same volunteers from there to give more hands-on mentoring?
    - There is Git Mentoring Google Group, but seems to have dies, spam only
    - Google Summer of Code (GSoC), Outreachy - both assign a mentor
- (Jonathan Tan) We could use a guideline on what's expected in terms of code quality, and guide on what good commit means. Information about what makes it easy to read and review.
- (Taylor) Folks who are newer contributors or haven't contributed much, do you have perspectives to share?
    - (Vincenzo) Finding a starting point, a problem to tackle, was difficult.
        - *(Christian Couder): There is https://git.github.io/General-Microproject-Information/ too. And https://git.github.io/Hacking-Git/*
    - `#leftoverbits` search term is listed in our Documentation/ReviewingGuidelines.txt, but Taylor suspects no new comers are looking into it.
    - People in the project will start looking at the next event and get to meet the person face to face to have a less daunting relationship.
    - (Phillip) There is a lot of information for new contributors to digest in CodingGuidelines, SubmittingPatches and MyFirstContribution. How do we find a balance between providing useful guidelines and overwhelming them?
    - (Jacob Stopak) As a newcomer, sent an idea that was too big to solve completely myself, but I would have liked to know where it was going, what is my part, what others will help with, and to be able to participate more in its implementation instead of it being done by others.
- (Jonathan Nieder) The mailing list is noisy and someone interested in a specific topic but the mailing list is flooded with lot of other things, unless they are specifically cc-ed on the right things. There's no easy middle ground between "my life is in the list" and "I only see what is sent to me".
- (Jakub) There's a bit of a middle ground - you can use a newsreader
    - *(nasamuffin): Jakub, do you have your newsreader setup documented somewhere? i'd like to see it in more detail*
    - *nntps://news.public-inbox.org/inbox.comp.version-control.git via Usenet newsreader (Thunderbird, Gnus for GNU Emacs, ~~KNode~~,...)*
- (Jonathan) In a project with a bug tracker, it's easier to know who is assigned to and who the collaborators are on something and what to expect moving forward. The information is in one place. In the Git project, if someone sends a patch on something I'm interested

in, I have to interpret why they're doing that - do they want to take this over? Are they giving me a suggestion?
- (Han Young) Han finds contributor guide to be lacking in details, he finds READMEs and discord to be complementary to his newcomer experience.
- (Emily) Which of these ideas should be implemented that makes the most sense?
    - Auto assign 1:1 mentors to new contributors
    - Split up the doc a bit more
    - Wiki: Where to start
    - Have more conferences
    - Have a bug tracker
    - Process documentation: What to do when a review comes in, next steps beyond what MyFirstContribution describes.
- (Taylor) The mentor assignment bit is what excites me the most
    - Most new contributors use GitGitGadget, it could notice new contributors and find a mentor for them
    - The key there would be documenting what that relationship should look like. Helps with clear guidelines on avoiding the kind of hijacking case Jacob mentioned (sorry about that!)
- (Jonathan Nieder) Great thing to do if we have a pool of mentors available. This culture is appreciated.
    - (Emily) Such culture is ingrained in Google in the form of community contribution.
    - (Junio) Hmm, where are the reviewers? :)
- (Glen) Discord or other informal channels are easier for mini-mentoring.
- (Jeff Hosteler) GitGitGadget is also doing mini-mentoring recently at a small scale that polishes before the author submits.
    - (Emily) Mostly GitHubbers? Should others pitch in?
    - (Jeff Hostetler) I think I'm auto-subscribed because I have write access to the repo.
    - (Junio) I've done some reviews there (it shouldn't be limited to GitHub folks).
- (Jacob) Thanks much for the documentation, step-by-step instructions are great
    - I used instructions on how to send patches with "`git send-email`". I didn't use GitGitGadget because it wasn't clear to me what it is.
        - *(Jacob Stopak): Oh I guess there is pretty thorough documentation for GitGitGadget* 😅
- *(nasamuffin) 1) you're welcome, 2) please feel free to send patches to myfirstcontribution.txt :)*
    - *(Taylor Blau): Very meta ;-)*

Overflow topics

Notetaker: Ronald Bhuleskar

- trackers - bug, review, etc

    - "whose turn" for patches
- (Minh): Multi-pack index and when you repack a large number of packs, you can rewrite pack index partially on things that have been changed but can't do with bitmap? Is this assumption correct
    - (Taylor): yes: bitmaps get rewritten from scratch to what pack they belong but it's close to an incremental approach as long as there is an existing bitmap.
- Back to project management practices that Junio mentioned, we seem to not shy away from discussing what kind of tool will help us: Bug tracker, etc but have more trouble with what practice to put in place to enforce it. Ex: with a public Bug Tracker, who responds to user issues, what does priority mean, etc. Wonder if people who are motivated to solve by making a small group to define this, bring back a proposal to the list.
    - (Josh) As Junio mentioned lot of patches getting ignored - and mostly directed to our day job, can we get cross company commitment to review/volunteer/run a bug tracker to explicitly help community
    - (Emily) Can view this as a donation, “donating project management headcount”.
    - (Jonathan) In the linux kernel there's a “Contribution maturity model” document. Common definition on what it means to be doing your part, allows companies to assess themselves.
    - (Taylor) Open Source donation was something that happened recently
    - (Emily) The Linux Foundation sponsors work to measure contribution: which company contributes how many patches/reviews
    - (Pono) Can help here to define qualitative metrics. Tools: CHAOSS that plugs into repository. Can work with someone to outlay this.
    - (Phillip) It’s easier to find reviewers in some areas than others. Different companies have different areas of interest.
    - (Emily) Yeah, we’ve noticed this at Google. Example: Submodule
        - Hard to review submodules; historically bad
        - Reviewers need experience
    - (Josh) Specific people to volunteer and how we recognize that volunteering effort in a smaller group. How about a Git reviewer conspiracy to honor people.
    - (Jonathan Nider) Make a small group and Jonathan can volunteer in it and Taylor is happy to help too.. (4 people volunteered - jrn, nasamuffin, keanen, ttaylorr)
- (Terry) semver was brought up in the compatibility discussion
    - I’d recommend looking at the Eclipse project’s practices. It’s Java based, has very clear language-based definition of what an API or ABI break is. But they also have a notion of “public” versus “public-internal” -- public-internal interfaces can be things that are known to be unstable, and when you use them you know you’ll be doing work to keep working on top of it. They also built a bunch of tools for checking when you break API/ABI. This was very successful.
    - Teams at Google building a web service don’t have to deal with nearly as much of that - you can roll forward and back quickly - but that’s not the case for things running on people’s desktops, where you need to take a more principled approach to API evolution.
- (Emily) library API stability (or lack thereof)

    - Integrity tests for API, will warn when something breakable
- (Minh) Reg SHA256 migration - Git forges
    - First-mover topic - once one forge moves, others will have to scramble. Should we coordinate?
    - (Patrick) Gitaly supports SHA256, unofficially already works in importing code to GitLab. But we need to adapt a lot of the frontend to support it.
        - *(Christian Couder):* *https://about.gitlab.com/blog/2023/08/28/sha256-support-in-gitaly/*
    - (Taylor) GitHub is in an earlier state but is also interested in picking this stuff up.
- (Emily) Backward compatibility discussion - Library API Stability
    - Put off version over version API guarantees
    - From talking with the LLVM team at Google, I learned the LLVM project adopts a similar attitude towards API backward compatibility. Should be an active contributor to not break your API.
    - (Jonathan Nieder) Maintaining C++ compatibility is hard, fully expressive API in C isn't easy. So it's a nice dividing line there. In Git it's all in C, annotations/signals where people can distinguish between "#include this header for a stable API, #include this other header for use-at-your-own-risk shifting foundation".
    - (Terry) LLVM is for static analysis, but the Git project should probably provide a higher level of API guarantee as these 2 projects are at different levels.
    - (Jeff Hostetler) Is there a roadmap with milestones around things like "at this point, you can work with multiple independent object database objects"?
        - (Emily) Yes, that's part of the holy grail of what we're trying to accomplish, and it's needed for submodules.
    - (Pono) licensing? Okay with current license, not a concern for Google but its a concern for other people using it. *(Pono): I did have a quick question about a new library. libgit2 is gplv2 with the linking exception. Are there thoughts about changing the license for a new library?I realize this could be very preemptive thinking :)*
        - (Jonathan) License as part of the interface, as soon as we have potential callers for whom GPL is not suitable this conversation will be easier. "Shall we relicense this subsystem to support this caller?"