Rust (moderator: Kyle; notetaker: Taylor)

- Kyle: Rust code in the Git project; do we want it? Why would we want it?
- Elijah: Anybody other than Randall that objects to it?
- Peff: I don't object, but what does it mean for existing developers? Will non-Rust developers need to start learning?
- Emily: The avenue for introducing Rust is replacing low-level C libraries that are complicated and un-maintainable.
- Peff: What's low-level? Peff considers strbuf to be low-level, but Emily considers merge-ort.
- Kyle: Anything dealing with untrusted inputs would be a good spot to start.
- Patrick: Instead: take a system like reftables that is already self-contained.
- Jonathan: Another nice thing: if someone is on a platform without good Rust support, then they could still use Git without reftables.
- brian: Rust is optional, can we add small new features and performance improvements that replace existing code, but it's not required.
- Emily: are we then carrying parallel versions of the same thing? What does that mean for the C version?
- Taylor: I don't want to double our vulnerable surface area.
- Patrick: Right, either you go all in or don't.
- Taylor: I don't want to give too much stock to a small number of people/platforms and hold the project hostage.
- Emily: old versions aren't going away.
- Patrick: Those could be maintenance releases.
- Do we need to support an older maint version in pure C
- Backport security fixes? Would issues be in both implementations?
- Do we want to encourage those without Rust support to maintain a friendly fork C version
- Johannes: Transpile C to Rust?
- Elijah: Does that presume we're not using Rust libraries, or are those transpile-targets as well.
- brian: We are going to have to use 'std', the question is what versions of Rust are we going to support? Can't be the latest version, since it only lives for six weeks.
- Taylor: vague idea of why we would want to use rust, but what are the concrete benefits of moving to rust?
    - brian: increased parallelization, can't write unsafe code, incremental parsing of objects. When it fails, you get a panic instead of a segfault.
    - Patrick: clear ownership semantics, was a huge problem with the memory leak work that they have been doing, it's not clear who owns what at what time.
    - Jonathan: 3 things; (a) user-facing benefit of memory safety, (b) productivity benefit to ourselves to have better architecture makes it easier to make changes, (c) would attract new contributors to the project.
    - Kyle: that's what I was going to say on (c), writing C is daunting.

    - Patrick: I'm in the opposite camp, I'm scared of Rust!
- Why not move over to the gitoxide project? Probably not realistic within the next 10 years, though Sebastian is doing great work.
- Taylor: great, what's the transition plan?
- Patrick: can't start until libification makes progress.
- Elijah: what about a low-enough level function that we can start with?
- Patrick: Still not work-able, there are too many global variables that we need to contend with.
- Taylor: IMO better to port to rust first from an implementation that we know, then libify in a language that has better support for refactoring.
- Jonathan: likes what Patrick was saying about having modularization make this easier. The other obstacle is that if we want to use any Rust at all, we need to have a POLICY on how we use Rust.
- brian: C shared library built in Rust, or top-level build with cargo?
- Have had experience with a previous C-to-rust migration! C FFI to call into Rust from C until everything is ported over.
- Patrick: modularization should be the first step, otherwise we're going to have the same architecture that we have now. If we inherit it now, it'll never get fixed.
- Mark: need Git to continue to be relevant in 10 years, making this kind of change is part of that
- Maintainability is something we can invest in.
- Emily: governance can help corporate contributors behave better. Say "your contributions have to meet this standard"
- Patrick: meanwhile I don't want to push away drive-by contributors
- Elijah: if someone wants to contribute a specific component in Rust, such as a replacement builtin, they need to call C, what happens to infrastructure such as option parsing?
- Emily: I really like the idea of starting with reftable
- Patrick: I was going to make a Rust implementation of reftable anyway

Top-level lib/ directory (moderator: Patrick, notetaker: brian)

- Patrick: Difficult to find stuff (README, etc); top-level directory is cluttered
- J6t: Clarifies that this is the C and H files
- Emily: Originally the plan was to move libified files into lib once libified
- Jonathan:
    - Discussion has been present on the list before (in Git 2.0 discussion)
    - Bikeshedding on the list (more difficult to blame)
    - Differences of opinion about lib and include and naming
    - Blame problems, but this is an opportunity to fix that
    - Heart of it: has nonzero costs, needs to have enough benefit to offset it
- Elijah: git apply doesn't handle directory rename
- Patrick: Bikeshedding not a problem; can be handled
- brian: Not really hearing an objection

- Peff: Not a problem for him, but not a really big objection
- Kyle: Not sure the value here
- Jonathan: I really love what Patrick said about “I can handle bikeshedding”, I think it’s great if you can guide a discussion on-list toward a decision on what we want for source layout, how we structure code, etc. Exciting!
- Taylor: improves things, makes it more maintainable/discoverable, but takes time away from other things like security/performance bugs. Should be done with a healthy balance.

Structured Error Handling (moderator: brian, notetaker: jrnieder)

- brian: idea for structured error handling!
    - Very little needed - pointer to error string, uint64_t error code, and ?? (third thing I didn’t hear). Return it on the stack. Rust does this kind of thing all the tihme.
    - Having that structured return value lets a caller decide what to do with it - print a message, decide whether to exit or recover, etc.
- Patrick: a few requirements
    - want to attach arbitrary metadata to an error (e.g. “I had a conflict between that file and that file for this revision”). Especially useful on the server side.
    - avoid having to parse error messages. Gitaly runs into this. Can imagine setting an envvar to get json or some other parsable format instead of human-consumable error messages.
- brian: sounds doable. GitHub also has the same hassle with parsing error messages.
- Peff: in your proposal, low-level code produces the error struct which is consumed at a higher level. Sometimes, though, you have many errors that are related.
    - “I couldn’t merge these files because I couldn’t open this file, because of this errno”.
- One thing we’ve considered is having a callback to decide what to do
    - - print the error, collect into a tree of errors, etc.
    - The point is to keep what’s happening in the generators of errors as simple as possible - I have this type of error and maybe some kind of context strings. That context could be owned by the caller, the callee can be responsible for copying it, etc. Inversion of control.
- Patrick: I like the way Go does things, can wrap errors or append messages, return up the stack until someone handles the erro7r. Why are we afraid of allocations?
    - Peff: What do you do when the allocation fails?
    - Patrick: can handle allocation failure by having a special error that has a statically allocated string.
    - Peff: sounds good, getting rid of die() on alloc failure is okay
    - brian: Rust panics on out-of-memory anyway
    - Peff: there are two different cases - small allocations are “you’re in trouble anyway”, big allocation of user-supplied length is something else
    - Carlos: Rust has a “try to allocate memory”, relatively new
- Calvin: how do you propagate up the call stack?

    - Peff: in my proposal, every function would take an error context struct, call the callback when there's an error, and keep the integer function returns. In brian's proposal, we instead return the struct.
    - Emily: Are we comfortable with the amount of churn that generates in the codebase?
    - Patrick: my inspiration is Subversion in that respect. It has nice error handling in C, they're a role model for me in how to do a library. It has nice error types that aren't a hassle to use.
    - J6t: if you compile in C++ and use exceptions, the problem has been solved 25 years ago.
    - brian: allocating strings for errors and then freeing them is a hassle in C. Versus Rust where that's handled automatically.
    - Emily: so it sounds like this is temporary pain
- Jonathan: I like Patrick's description of requirements. One thing that would make me more comfortable with the churn is when we get benefit even from partial conversion
    - E.g. could say we get structured error messages for the call sites that have been converted and not from others. And when I go on a debugging quest and wish I had a machine-readable structured error message, I can send a patch to convert more call sites
    - Peff: Refs code uses its own strbuf based error handling which is worse in every way than the options we've been discussing. :) That can be a good playground to try things out.
    - Patrick: +1, seems like a good place to start.

Platform Support Policy (moderator: Emily; notetaker: Calvin)

- Emily: if you want Git to support your platform, you have to provide your tests (e.g. provide your own CI test runner)
    - Should we be more explicit about, for example, the version of C that one must support.
- Brian: less-common architectures (e.g. MIPS) sometimes can catch problems (e.g. unaligned access) that are not a good practice anyway
    - qemu based CI is so slow
    - netbsd, openbsd, freebsd, probably solaris are up to date with modern standards, probably supportable
    - I'm more comfortable with "you have to have threading and POSIX 2008" than with "you have to provide a CI runner"
- Peff: I'm not sure what people *have* been counting on
    - Do we have a way to find out what people are using?
    - "Take a little risk, see who screams" has worked okay in the past but takes a while
    - Rust is probably a big change
- Patrick: keep in mind that we're at the core of the whole operating system, part of the bootstrapping path

    - Emily: yes and no - Git is not just the client, but Git is a standard. You can use older versions of Git and clone things from GitHub. If we still support the same protocols, I don't think needing native git.git CLI support to run on your platform is as compelling as new Git being able to support these older standards.
    - brian: OpenBSD doesn't like the GPL, has a project for getting trees called "got", it's in C and supportable. It can be a valid bootstrapping tool.
    - Patrick: The user experience there is a little closer to CVS. But it's still an option.
- Jrnieder: looking from user perspective, Git is the tool people are used to for day to day development
    - Emily: There's a difference between using "a git" vs using the latest version.
    - Jnieder: telling users to use an older version might result in users asking questions on the mailing list about those older versions, it's also not free.
    - Peff: to be fair, HP Nonstop support hasn't been a matter of "please support me for free" - the maintainer there has been active in helping test and debug things. The question here is not about whether to continue that but rather about whether we're willing to increase the platform dependencies when it breaks such a use case.
    - Peff: Are we okay with dropping NO_PTHREADS support?
    - Brian: POSIX 2008 shouldn't be that controversial. Neither C11 should be. We shouldn't take it too far like POSIX 2024, but we have to set "some" standards.
    - Emily: So next week when we come home we update the "minimum requirements" on the mailing list, and everybody upvotes?
    - Jonathan: we live in the real world - the *spirit* of "let's require POSIX 2008" sounds right, but real-world considerations should matter more than the exact text of the standard
    - Peff: example: Android is missing pthread_setcanceltype, which leads to Git on Android using NO_PTHREADS
    - brian: it can be enough to pretend to support (compatibility shim)
- Calvin: is there a threshold % of users for "unimportant enough to break"?
- Jrnieder: it depends on the platform. Do the requirements that a given platform imposes push us in a good direction in general as a project?
    - For example, Windows is a very non-POSIXy platform, but it has nudged us toward thinking about subprocesses in a different way, and I think that's been really healthy
    - brian: As another example, Plan 9 is really difficult to support, it won't pass the test suite
    - z/OS patches originally came in and were gross. I saw a patch come in recently that was more acceptable.

SHA-256 / Git 3.0 (moderator: Patrick, notetaker: Taylor)

- Patrick: some of you may have seen some patches from a few months ago to start thinking about this, and we now have a "deprecations" doc of things that we plan to get rid of.

    - One thing going away is grafts
    - Another thing is git pack-redundant
- Patrick: SHA-256 will become the default object hash, but is contingent on major platforms supporting SHA-256.
- Emily: this helps us too, because it provides some motivation that this is going to happen.
- Patrick: having it in the 3.0 document makes it easier to push at large organizations.
- Jonathan: with partial clones if I understand correctly users were already trying it and complained to GitHub about it not being turned on.
    - Taylor: not really…
    - Peff: I was embarrassed, so I wrote the bitmap support for filtering objects
    - Taylor: yeah, and turning it on was easier, but didn't require updating database columns, etc. so the transition period here would be much more expensive.
    - brian: "can you put it in the customer feedback repo" is something customers can ask for
- Jonathan: could use interop as a way for Git to default to sha256 with the only harm to end users being that clones and fetches get slower until servers support it
- brian: not planning on working on it unless their employer pays for them to do it.
- Mark: customers wanting to start with SHA256 so they don't have to migrate later
- Patrick: GitLab never really wanted SHA-256 until it was done on the Gitaly side.
- brian: has definitely seen it on StackOverflow as something that customers want. Not a huge selling point right now, but will become a humongous selling point at some point (beyond >2030).
- Peff: if the proposal is to make SHA-256 the default, then we need to be developing with it now, and we're not because there is no interop.
- Taylor: we should do it via interop, (a) because GitHub could not host it, but (b) because we would introduce the same lack of testing just with the interop code, not the SHA-256 code. So it must be done via interop.
- Patrick: OK, but why do we care about SHA-256 locally if they're using SHA-1 on the remote? The remote could be compromised
- Jonathan: You can verify things locally. This is the core idea of distributed version control.
- Patrick: Oh, I see.
- brian: Signatures work with both hash functions, you can sign with both.
- Peff: It does not feel right to me to set the user default until we the project are using it, and that the interop code is a part of that story.

(back to Git 3.0, we combined the SHA-256 and Git 3.0 discussions into one)

- Peff: I have a proposal for Git 3.0, maybe this has been discussed? Can we get rid of some of the older protocols (dumb HTTP)?
- Patrick: Lots of esoteric things, like show-branch, which apparently nobody uses.
- Elijah: not just removals, but changing defaults, etc.

- Emily: are we interested in non-backwards compatible changes, like adding multi-Author fields to commits?
- Peff: I think that's a bad example, it can be done without breaking compatibility, but it was decided to not to do it. You're welcome to resurrect the discussion.
- Patrick: … but it's a different question of whether or not that would end up in the document.
- brian: multi-signatures
- Jonathan: Two questions I'm hearing:
    - Should we include things that haven't been implemented yet? Probably not.
    - What do we think about this major version as a way to break interoperability with older versions of Git?
- Patrick: I would not be in favor of breaking something, at least in cases where we can add a protocol extension and/or new capabilities. Intentionally breaking interoperability with an older version does not seem like the right way to go.
- brian: I agree, but for the dumb HTTP protocol, C git uses it, Eric Wong is really into it (lore.kernel.org supports it), etc.
- Emily: Bundle URIs and resumable clones, could in theoretically work for resumable clones, but we don't have client-side support.
- Peff: Pretty sure that this isn't the case.
- Mark: can I ask a "dumb" question? What would it take to get a schedule for 3.0?
    - Patrick: Junio says not too often, maybe only breaking releases every 5-10 years, which means 3.0 would come in 1-2 years.
    - Peff: 1-2 years is what is in my mind.
    - Taylor: I think that whatever answer we come to agreement here will not be satisfying for you.
    - Taylor: the items on that document aren't a checkbox list of things to do before Git 3.0, but isn't a "let's get all of these things done and then we'll release Git 3.0".
        - More that we'll all wake up one day, realize that we've done all or enough of what would go into Git 3.0, then remove a bunch of code, and ship it.
- Jonathan: collecting breaking changes and aggregating them so users can prepare for them together is helpful, but it's not the only way that breaking changes will happen, especially if there is something that needs to go away.
- Patrick: I want three things from this document:
    - Reminder / documented intent.
    - User feedback to hear things like "this is important to me, what is (if any) the replacement?"
    - ???
- brian: changing the default branch name?
    - Taylor: I would be in favor of doing it sooner
    - (some discussion)
    - Taylor: We should consider doing it for git.git as well.
- Peff: we might write a bunch of those patches, move them into 'next'. Could we have a Git 3.0 pseudo-maintainer.

- Jonathan: I have a naive question: why wouldn't it look like turning on a single Makefile variable?
- Emily: and then we go and delete the code inside of the #ifdefs
- Taylor: whoever is the maintainer at that point in time could consider a double-wide release cycle, where we delete that code, implement new things, and then at the end of that cycle the artifact is called Git 3.0.
- Peff: very few people run "next"
    - Jonathan: True. Could imagine some % of GitHub Actions runners automatically running "next", that's the kind of thing that gets a more representative workload.
- Toon: do we want to have maintenance releases?
- Emily: if we're dropping support for earlier versions, we should just do it.
- Taylor: we should probably have a few supported release tracks that we designate as LTS releases.
- Patrick: For security issues only, probably for a period of 1-2 years.
- Peff: Should we write this up as a plan for the person who actually does release engineering?
    - Taylor: I can write up that plan, and be the point person for the LTS releases.
    - Jonathan: the Linux kernel has dedicated maintainers for LTS releases, which seems to work well.
- brian: we can certainly tell that to Junio, but it's also ultimately their decision.

Git / Conservancy Stuff (moderator: Taylor; notetaker: Patrick)

- Taylor: Sent out new mail to the list with SFC activities.
    - https://lore.kernel.org/git/Zusxcweod1O88h7j@nand.local/T/#t
        - LOTS_OF_MONEY spam alert
    - Generally have more money in our account than we know what to do with.
    - SFC consists of 4 persons: Junio, Taylor, Ævar, Christian
- Taylor: Trademark discussion
    - Enforcing the trademark is untenable because there are so many projects out there using "Git". We cannot go after those to defend our trademark, and would also not be in the spirit of the Git project.
- Taylor: Money
    - We've got a large amount of money for an OSS project with few expenses, $90k USD.
    - Heroku is our biggest expense at $60 USD/mo.
    - What can we do with this money?
- Chris: Mentor Outreachy this year, we'll likely have to sponsor them.
    - $8,000 USD/student. GitHub can probably cover one or two such students.
    - GitLab sponsored last year, but getting funding was taking a very long time on their side. John probably doesn't want to repeat the process again due to the hassle.
- Jonathan: Git gets money due to GSoC, too. These kinds of mentoring projects are things where it's easy for donors to justify giving.

- Chris: We do get money indeed due to the GSoC mentoring.
- Chris: We have tried to sponsor travels for GSoC students in the past to come to Git Merge. Was quite easy to do that with our funds. Covid restricted that somewhat.
    - Students used to have problems acquiring a visa, creating a catch 22 because we had to book the flight up front, but it wasn't yet clear whether they'd get the visa. BUt without the flight, they wouldn't get the visa, either.
- Peff: THe wasted money on that is probably small enough compared to how much funds we have, so the risk is comparatively small.
- Jonathan: Do we have a rough estimate around how big yearly influx minus expenses is?
- Taylor: The exact year is $93k USD. It's hard to tell exactly due to changes in reporting by SFC. Last year it was roughly $89k USD, so only up ~$4k. Typically we have a yearly income of around $10k-$20k USD/yr.
- Peff: if you want something, would be relatively easy to run a fundraising campaign for Git
- brian: I'm sure that we can easily raise additional money via individual contributors, too.
- Jonathan: If we have the money for it, I think it might be relatively easy to justify hiring a full-time community manager for the Git project, for example.
- Peff: 90k is a high amount of money to do small things, but it's not really enough to actually sponsor full positions for an extended period of time.
- brian: True. Nobody is going to work for you without insurance, so it indeed isn't all that much money indeed.
- Jonathan: Edward Thomson also mentioned that libgit2 passed on some money to other projects that can use it better
- Emily: also wanting to spend money on contracting on particular projects
- Brian: there are people who do contractor work like this. I just don't know whether we can find somebody who is willing to accept that little money.
- Emily: Developers are expensive, tech writers are a lot cheaper. Would that be a viable option? They could for example rewrite lots of our documentation.
- Johannes: We could delete obsolete documentation!
- Patrick: I tried to do this on the GitLab side, but it never really worked out. They didn't want to join the Git mailing list.
- Emily: Same.
- Mark: There would be some concern about having tech writers interact with the mailing list flow.
- Peff: Ævar is not active on the PLC anymore for a long time. Should he be removed?
- Taylor: We've been talking about that in the PLC. There are two options:
    - EIther remove him without replacement, such that we have three people on it. That would also make it less awkward when it comes to voting ties.
    - If we replace him, it would almost happen to have to come from someone who isn't affiliated with a large forge / company. We already have representation from GitHub, Git Lab, and Google, so would want an OSS contributor not affiliated with a major company.

    - Ideally I would like him to come back, but haven't been able to hear from him whether or not he is interested in doing so.
- Michael: The trademark was originally owned by GitHub?
- Peff: The git-scm.com web site was originally owned by Scott, was then moved over to SFC. The SFC applied for the Git trademark, came to an agreement with GitHub about details of how it would relate to the GitHub trademark.
- Michael: Is there any obligation of Conservancy to GitHub that the trademark is enforced?
- Peff: No.
- Michael: Background is that the proposal was to stop enforcing it. SO the question is whether we *have* to due to an agreement.
- Johannes: You lose the trademark if you don't enforce it in many different countries, so we either have to or don't and may thus lose it.
- Chris: We mostly enforce it by privately emailing e.g. website owners that use Git in a way we don't agree with. So we do enforce it as necessary, even if it only happens very infrequently. Some companies do not register a trademark that's conflicting, but they still try to use it.
- Jonathan: I think it would not be a good choice to not be assertive about the trademark at all. On the other hand, filing lawsuits against low-harm cases doesn't seem like a great use of time and money. So I feel like the current balance is sensible.
- Peff: SFC's lawyers might disagree with that assessment.
- Taylor: They want to see some new version of the trademark where we e.g. only enforce our trademark on the logo.
- Chris: We raised the question a couple years ago of what to do about the trademark, and folks basically agreed with the current way of how we handle it.
- Brian: Debian has a trademark policy that might be sensible to have a look at for inspiration. It e.g. says that you have to communicate truthfully.
- Peff: We have similar stuff like that in our policy. The problem is that back when writing it we had another company that was trying to use the trademark for a "shitty" reimplementation of GIt, and we didn't want that. We don't really care for "git-foo". Dashed commands are explicitly allowed. GitOxide is e.g. technically against the trademark policy due to being CamelCased, but we are fine with it.
- Taylor: We cannot do that, we have to consistently enforce the trademark. It needs to be very strictly defined what is and is not okay.
- brian: it needs to be definitive whether you're doing the "good" or "bad" thing.
- Jonathan: It might be interesting to tie this to compatibility with the Git test suite. SO if you pass it you are allowed to use it, otherwise not.
- Jonathan: So IIUC, we have questions for the SFC lawyers because we canot really answer a lot of the questions?
- Chris/Taylor: We can do that.
    - Taylor: We will talk offline to figure out who does what.

How to attract new contributors? + Community Discord (moderator: Jonathan N + Calvin Wan, notetaker: Emily)

- jonathan: we talk about this a lot 🙂 let's avoid the common pitfall of catering to the tiktokkers and the youts (hypothesizing about "current generation"):
- but, our contributor base isn't representative of our user base
- and our current contributor base doesn't reflect exactly the skills needed to improve git - like interface design is not our strong suit. how to attract people who are better at our weak spots?
    - taylor: this weakness is an existential problem for Git. jj, gitbutler, gitkraken, etc
    - mark: +1
    - peff: one size doesn't fit all, us deciding not to include a GUI is understandable, but workflow improvements like jj's are pretty interesting
    - jonathan: ex. in hg there's someone very involved in UX review. we don't have someone like that
- missing other disciplines - tech writing, product management, UX research, etc.
    - common problem in open source but would be cool if we could get good at attracting/retaining these people - and cool for the not-eng-discipline people
    - patrick: we could adopt a style guide or guideline but we still wouldn't be good at enforcement
    - john: people need to know what they can contribute to - cf. project tracking discussion later on
- jonathan: instead of trying to guess - can we think generally, how do we make work easier to approach? how can we lower the barrier to entry?
- patrick: someone is writing third-party rewrite of gitglossary. huge improvement over what we have, well made, but the person didn't want to come back to contribute. was afraid of the community giving pushback
    - patrick was willing to handhold this potential contributor, but it didn't seem like enough to make this person comfortable
- jonathan: related to community discord server - what does it mean to function better as a community?
- calvin: the entry point doesn't **need** to be discord, but we should pick some entry point that lets users contribute other than mailing list participation
    - and need to be able to navigate new contributions comfortably
- brian: how to write text that's accessible to non-native english speakers, for example? the mailing list isn't great for these kinds of changes.
- discord is proprietary, that is sometimes an issue
- moderation on discord is an issue - having an unmoderated discord will actually drive away contributors. that means actual dedicated moderation
    - balancing between sufficient moderation (list) and ease of use (discord)
- patrick: new contributors sending changes but the changes being ignored
- brian: git-send-email is a barrier, but so are PRs/MRs in some cases

- jonathan: the localization example is a good one - the translation layer is in github, uses a very typical dev workflow, and that's working well. there's a strong community there. are there other places we can do something similar?
- peff: can we do that with documentation?
    - jonathan: can we have a documentation maintainer? hypothetically: we hire a tech writer, and that tech writer acts as the documentation maintainer only. curating existing docs, making sure docs changes get good reviews, how to attract new tech writer contributors, etc
    - peff: can we manage documentation as a subproject that doesn't use the mailing list, and make tech writers' lives easier?
        - how to negotiate that with code changes that require doc changes is trickier, we'd have to figure out how to do it, but doable
        - jonathan: readthedocs
- jonathan: we don't advertise well that we can accept contributions in a different way if people are committed to the improvement
    - peff: sometimes a mentor can “translate” a contribution. Individual contributors are already interested in mentoring, do we need more/different mentoring?
    - mentoring list isn't working well yet - maybe it's too faceless? should we get a list of individuals who want to mentor?
    - taylor: should we literally put photos of the people on the mentoring list up somewhere? "here are real humans, they will reply to you on git-mentoring@"?
    - jonathan: in-person meetups help with this. emailing is transactional, but e.g. python meetups are interactive
- patrick: we had the git berlin meetup a few months ago, lot of people came, we did lightning talks and user conversations. it worked well - let's use that model more
    - taylor: hey, we can help spend money on that
    - brian: those are cool but for example, houston linux users group is quite small. meetups like this can be helpful, but it's not the only source.
    - peff: it doesn't really scale up. python users group are user-to-user, doesn't necessarily draw python project contributors.
    - nasamuffin: Gerrit has a community meeting once/month, should we use discord for f2f video meetups?
    - peff: if people want to do big group meetups great. we could also use it for 1:1 meetups that way, and advertise that it's an option

Modern Build System (moderator: Patrick; notetaker: brian)

- Patrick: three different build system; should get rid of at least one of them
    - Should delete autoconf because it’s not really maintained
    - Think about a proper build system
    - Obvious choice: cmake
- Taylor: What’s the problem with Makefiles
- Patrick: Non-standard
    - Meson is nicer than cmake as an alternative

- Jonathan: xz compromise shows autoconf is a risk
    - The Makefile without autoconf should be fine for distros. The way it is configured is a distro's dream
- Taylor: distro builds that can't handle "make" without configure are a distro problem
- Jonathan: Modern build system can reflect the structure of how your code is set up
    - Declared dependencies
- Brian: Rust will make the decision for us: cargo
    - BSDs use Make (granted, not GNU make) for building
- Patrick: Is anyone else in favour of a *proper* build system
    - Ninja is way faster than make to build the projects
- Taylor: Feels odd to build with a fancy tool that might have a dependency on Git
- Dscho: --help is a autoconf feature and removed features are detected
- Patrick: Isn't that an argument for cmake over autoconf? Dscho: yes
- Kyle: Editor integration is useful
- brian: standard structure is helpful for LSPs
- Emily: libification has shown that makefile is cumbersome
- Jonathan: Should we do a comparison of build systems in terms of what we need from them on the list? Similar to Documentation/technical/unit-tests.txt
    - Patrick: I can write such a thing.
- Patrick: Are there any features we need to consider?
- Johannes Sixt: Consider supported platforms
- Patrick: Want to verify that cmake is up to the task by testing in CI?
    - Will volunteer to post something to the list

Bundle-URI on fetch / resumable clone (moderator: Toon; notetaker: Taylor)

- Toon: We have bundle-uris, which allows us to download bundles before cloning. What would it take to use them on fetch, too?
- Toon: Complex; you might end up downloading bundles with a lot of objects you don't need.
- Toon: There was a bundle token that could be used by the client to determine whether or not it needs the bundle. Unfortunately, since you don't know what amount of history you're missing, the bundle in and of itself may not be sufficient.
- Toon: How do we make bundles more efficient on fetch and aware of what the clients do and don't have.
- Johannes: In VFS for Git, the set of bundles is "opinionated" and regenerated often (e.g., weekly, daily, hourly, etc.). We don't really care about oversharing, because it's fast enough to download those static files.
- Toon: That's true for the server, not for a singular client.
- Johannes: Right, but it's to protect the server. If you don't care about having too-big of a bundle, it's OK because you can just fetch the last hour and then see what you still need on top of that.

- Johannes: There is logic to determine what you need to download based on the time that you last fetched, etc.
- Elijah: Are they thin packs? Yes.
- Patrick: The heuristics that we have to advertise what kind of bundles are on the server is not sufficient enough to determine which bundles the client may or may not want to download. The client must guess.
- Jonathan: It seems to me that the property list for the bundles as they are today should probably be considered a starting point (as in “part of a feature”). The heuristics need to be extended.
- Jonathan: At Google we use packfile-URIs, and one of the advantages of how that works is that since the packfiles are advertised after the normal fetch negotiation, you get a curated list. For bundle-URIs, we need something analogous to the fetch negotiation for the client to be able to make a similar decision of which bundle to download.
- brian: Extending the feature to store timestamps, then you could store the last fetch on the system would inform some better selection of which bundles to clone down during a fetch.
- brian: Would make a big difference for folks in environments which do not benefit from reliable Internet connections.
- Toon: Sure… but you have to put a lot of pressure to keep those bundles up-to-date on the server side.
- Jonathan: One of the advantages of bundles over packs is that they have information about the references. One potential property could be “here’s the length of this header” and then having the client download it to examine whether or not it wants such a bundle.
- Patrick: Probably would want to cache those on the client so that we’re able to avoid re-downloading these every single time.
- Toon: About resumable clones
- Jonathan: Protocol supports it, just hasn’t been implemented. Someone needs to just get it done. :)
- Beyond that, the main complication is how you store the state and what the UX is for resuming. But a person implementing it can figure those things out.
- Brian: Broken-proxy situations require some configurability of where your bundles can be downloaded from.

Project Tracking (moderator: Emily, Patrick; notetaker: Taylor)

- Emily: Patrick and I were talking about roadmap-like things sometimes this week. It would be nice if we had a list of projects that the community has agreed are good ideas. For e.g., “generally we are working on getting rid of ‘the_repository’”.
- Emily: We have agreed that libification is a good thing, so anytime that someone sends a patch in that area it isn’t up for discussion whether or not the patch is fundamentally going in the right direction.
- Emily: It would be cool to have a list of agreed-upon things. I also think that we should get better at figuring out what we agree on. I.e., the status-quo is argument on the list

until quorum, and would like some kind of more structured way to determine when we are in agreement.

- Jonathan: When you say “predictable process”, what does that actually mean? Python has PEPs, they have a very clear state machine that stringently dictates the process.
- Emily: probably not something that strict, but would like to generally be able to tell “is it going in the right direction?” And approximately, “Approximately when will we be able to tell when it’s done?”
- Taylor: how do we quantify that? A lot of what developing consensus looks like is developing patches and then sometimes throwing them away if/when we find a fundamental design bug, etc.
- Patrick: “Are we going to do it?” is when something like a brief RFC is merged.
- Taylor: Is the Git 3.0 deprecation doc a good example?
- Patrick: Yeah.
- Patrick: We would also want to have a document with a little check mark indicating when we are going to do small things like clean up memory leaks.
- Jrn: The pain point is being able to know when making a proposal how much support they have.
- Taylor: Sometimes implementation clarifies one’s thinking.
- John: We have some tribal knowledge on what is accepted in which areas. Can we codify that?
    - Patrick: The BreakingChanges statement says that we can change things on the document if there is new information. I’d like to see that this statement is echoed elsewhere.
- Taylor: It feels like it would be more useful to have a discussion based on a WIP or PoC (e.g. UNLEAK() and FREE_AND_NULL()).
- Peff: I don’t want people to produce patches, say, if we blessed unit tests, and then consider that discussion immutable. That doesn’t give us the opportunity to make changes to the fundamental design. I don’t want to use it as a cudgel to shut further discussion down.
- Jonathan: “I guess I am going to argue in favor of cudgel.” Taking unit tests as an example, if someone later says “I don’t see any value in unit tests”, that’s not very productive. If they say “I see the value of unit tests, but not at this cost”, it’s a more worthwhile conversation.
- brian: Maybe we can only use that as an argument for 4-5 weeks. Having some disagreement is fine, just don’t want to make it indefinite.
- Peff: On PEPs. Often someone will say “I’m going to go in this direction”, and I’ll feel mildly negative about it. That’s very different from voting against something.
- Emily: Sure, there can be more voting options than just yes/no/abstain.
    - Jonathan: Apache does a nice job with +/- 0, +/- 1 (https://www.apache.org/foundation/voting.html#expressing-votes-1-0-1-and-fractions)
- Emily: we should be better about saying generally how we feel about things at the end of the series).

- Taylor: Taking a step back for a moment, what are some examples of things that are going wrong that need to be changed? What is wrong with the project here?
    - Emily: things are taking too long.
    - Emily: Goal-posting moving, etc.
- Johannes: Yes, that can be very frustrating and cause a lot of friction.
- brian: When you do creative work, you get feedback and sometimes you take it and sometimes you don't.
- Taylor: at the risk of repeating what brian said, it's hard/impossible to please 100% of reviewers on 100% of series. At some point you have to step back and let the maintainer evaluate.
- Jonathan: I think the point of Patrick's proposal is "how do you go about navigating this with a 10-series long project?" Then you're in trouble. The proposal would allow you to check in a bullet point, or a design doc, and go through that process and navigate to the point where you get the maintainer's blessing and the conclusion is checked in.
- brian: Maybe a bullet point is too small for some things.
- Taylor: so it's a continuum, sometimes you are going to save time by just writing the patches, sometimes you are going to save time by writing out what you are doing (over a multi-series effort) first.
- Patrick: discoverability!
- Mark: could we encourage newcomers to email one of a few people for a particular effort (like libification) and have them comment on it?
- Taylor: you might not want to centralize that much power.
- Toon: How about structured errors? We agree it's a good idea, but not on the mechanism of implementation.
    - Patrick: Sure, sometimes that just takes time. We don't have to document how to fix them (maybe how to find them, but I digress).
    - Patrick: For structured error handling, it should be more involved. It makes sense to have an RFC or design document that documents how it's supposed to look like.
    - brian: or an initial patch series that comes up with a design.
- Peff: In our project, the formalism of voting is "is it merged to 'master' in Junio's tree".
- Emily: I want to have the process of getting from discussion to merge be less fuzzy.
- Peff: So in brian's example, let's take SHA-256. The process by which the maintainer decides that is inherently fuzzy.
- Emily: Sure, but I would like to be obvious to someone besides the maintainer.
- Jonathan: (to Peff) you mentioned sometimes you have a mild negative feeling about something and you're good about expressing it on-list, but for a lot of contributors that will cause some discomfort and it will cause them to stay away from that thread. If we're a little more clear about what's expected, then conversations can get stalled less often - e.g. when a thread needs a comment from a refs expert, getting that comment that supports forward progress.
- brian: I just gave Taylor feedback on the SHA-1 series that he wrote, saying that I didn't love it. But others felt OK about it, so we moved forward.

- Emily: strawman doc: https://github.com/nasamuffin/git/commit/54079ab00002c6dfa7ac1a33d9810792978d2cce - maybe it's bad enough we can poke holes and salvage something we do like
- Peff: it’s important to leave at the end of your review the way you feel about something instead of just having a few comments.

lgit-scm.com state of the site

- Johannes: worked on a static site version of git-scm.com.
- (Johannes: demonstration)
- Peff: let’s move it over.
- Taylor: AGREED

Library LICENSE

Anyone with opinions about this, reach out to jrnieder and he’ll aggregate opinions and report back.