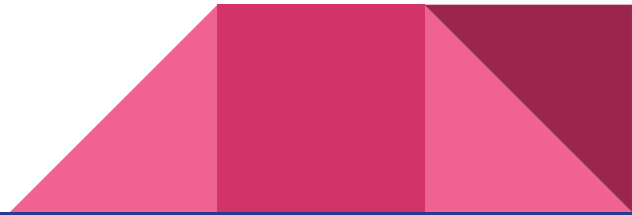
LLVM Move to GitHub

Bird view analysis of the survey (as of Tuesday)

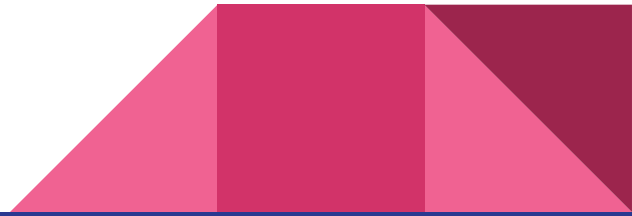
In a Nutshell

- 237 responses so far, 32 group answers
- >80% of the people work in Clang/LLVM, ~30% in RT/libc++/LLDB
- ~67% would wish SVN went away, another 22% doesn't care
- GitHub was mostly praised as a good move
- ~35% of developers worry about cross-repo breakages
- ~35% worry about cross-project bisecting
- ~35% worry about checkout disk space
- Only 12 people would start over with SVN
- The rest is evenly divided between mono / multi



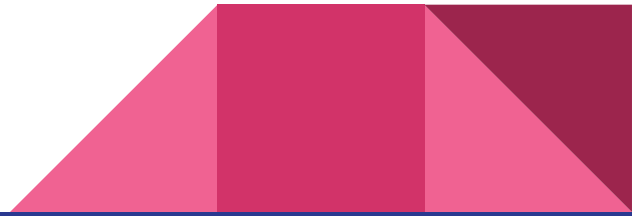
LLVM Projects Composition

- **Personal (85%)**
  - ⅓ core/active developers
  - ⅓ occasional contributors
  - ⅓ downstream users
- **Groups (15%)**
  - ⅓ core/active developers
  - ⅓ occasional contributors
  - ⅓ downstream users
- **Groups**
  - 11% full upstream, 56% mixed usage, 33% full downstream
  - ⅔ Small teams (2~10)



LLVM Projects Usage

- For 54% of the users, LLVM is a core technology in their projects.
- For the remaining, 22% is for products, 18% for pet/academia projects.
- LLVM+Clang takes >80% of the users, with 66% always checking them out.
- Compiler-RT, libc++ and LLDB take about 30%, clang-tools 25%.
- LLD, Polly, others get 15% or less.
- 53% of the developers never had the cross-repo commit problem.
- ~9% are **very** annoyed by it, 24% would like it fixed.
- 38% think it’s important to have multiple projects in the same repository.
- 26% think it’s irrelevant



LLVM Projects Usage - Comments

- **Bisection**
  - *I spent lots of time integrating compiler changes and debugging compiler issues, so it is very important to me to know the exact state and the ordering of different builds. As long as there is a schema that can be used to consistently reference the state of the project, I would be fine with the solution.*
- **Versioning**
  - *I would hate to be in a situation where we'd have to describe a Clang version as (LLVM hash1 + Clang hash2 + compiler-rt hash3 + ...)*
- **Disk Space**
  - *I use git clone's --reference option for this. It causes git to re-use data from the clone*
  - *I frequently have multiple checkouts on the same machine.*

Git & GitHub

- Massive support for Git-only usage
  - ⅔ said subversion only slows them down
  - Another 23% said they don't care
  - Most comments about subversion said whatever keeps linearity is fine
- There is an impact
  - Around CI infrastructure, mostly one off
  - On developers' scripts / workflow, one off plus mostly benefits (for at least ⅔)
- GitHub comments mostly positive
  - Some suggested GitLab
  - Others prefer "*a big company*" to sponsor and manage our own
  - But most responses were towards GitHub being the best option

Concerns about Git / GitHub

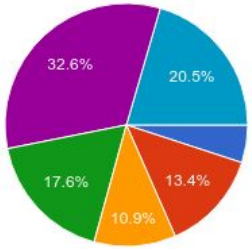
- *The existing **linear history** is extremely convenient, and I hope it stays that way.*
- *But if **version locked** repositories aren't well handled, I'll be adamantly opposed to [moving to Git].*
- *The move to git will obviously impact everyone significantly, whether or not it is worthwhile.*
- *We would need to modify our daily update, build, and tests scripts.*
- *We really appreciate the ability to pin down changes to a **specific numbered change** / check-in.*
- *Changing [commits'] **SHA-1**s will break our merge and automated QA processes.*
- *GitHub is not "free" if you actually want **responsibility** for your data.*
- *If github dies then anyone's repo copy can be pushed to GitLab or a dedicated server.*
- *would like to keep URLs pointing at llvm.org (or a subdomain) so that they're in llvm's control so we can **minimize disruption** in case we need to switch providers again.*
- *GitLab seems to open up more opportunities in the future for the project*
    - *w.r.t. [...] project management and continuous integration.*

Git Worktree - Disk Space

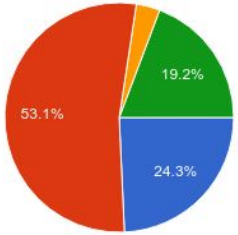
Project	Repository	In Wortree	Ratio
LLVM	570MB	219MB	38%
Clang	422MB	122MB	29%
Compiler-RT	59MB	35MB	47%
LibC++ (+ABI)	75MB	51MB	68%
LLD	39MB	7.6MB	19%

Additional space in main repo for work-trees: 2~5MB / tree

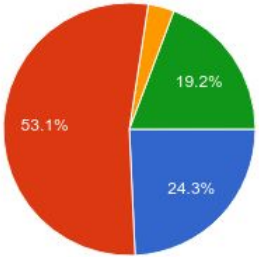
Git multi-repo (sub-modules)



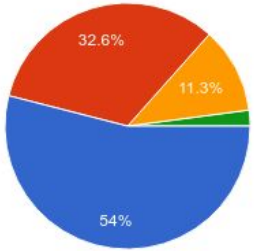
- Actively contribute tooling improvements to improve it.
- Integrate it into our downstream fork.
- Use it for upstream contributions.
- Use it as the primary interface development environment.
- Use it for cross-project operations only (bisection, blame, searches).
- I don't contribute.



- Using Git sub-modules for everything except commit
- Using the Git repos directly, sub-modules only for bisecting, etc.
- Using the SVN bridges.
- I don't contribute.



- Using Git sub-modules for everything except commit
- Using the Git repos directly, sub-modules only for bisecting, etc.
- Using the SVN bridges.
- I don't contribute.

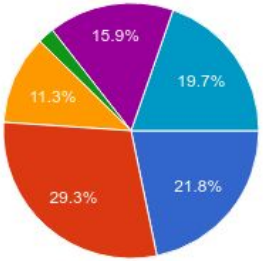


- Nothing consequential.
- A little, but it'll be fine.
- A lot, but it'll get done somehow.
- Too much. We may stop contributing to LLVM.

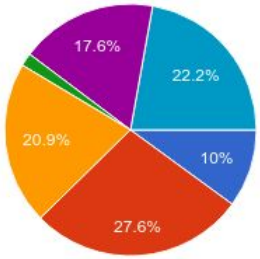
Git multi-repo (sub-modules)

- *The multi-repo is closest to the current situation so very likely to have less pain in transition for most. But I still think it's a worse position to be in.*
- *I currently use git-svn for each of llvm, clang, etc, so migrating is extremely simple. I don't bisect often so using the submodule repo for that works fine for me.*
- *the multi-repo seems too complex. It's better not to invent custom tooling for a problem that isn't unique to LLVM.*
- *Wary of the bot merging the commits into the umbrella repo. Not having guarantees on the merge + possibility of the bot going down and making a mess when catching up.*
- *committing is slightly simpler, and the tooling to create useful revision numbers exist is somewhat harder.*
- *I fear the difficulties in backing out committed changes that later turn out to have problems.*
- *Checking out and coordinating all the repos is a hassle and often developers do not test their changes with another component because they simply hadn't checked out a subproject.*
- *I'd actually prefer proper hierarchical submodules with clang in llvm/tools, but that's not an option here. If we need to contribute to git to improve this, we should do that.*

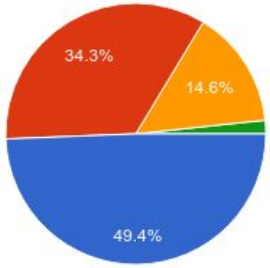
Git mono-repo



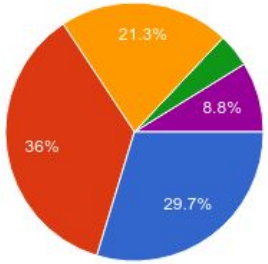
- I'll use the monorepo as soon as it's possible, even before it's canonical.
- I'll use the monorepo as soon as it's canonical.
- I'll transition to monorepo eventually.
- I'll use the SVN bridge on separated sub-projects forever.
- I'll use a Git mirror (and/or git-svn) on separated sub-projects forever.
- I don't contribute.



- We already use monorepo.
- We'll switch to pulling from monorepo during the transition period.
- We'll switch to pulling from monorepo eventually.
- We'll integrate from the SVN bridge forever.
- We'll integrate from the split sub-project Git mirror forever.
- There is no downstream.



- Nothing consequential.
- A little, but it'll be fine.
- A lot, but it'll get done somehow.
- Too much. We may stop contributing to LLVM.



- All in one. We use enough of the projects / we commit across project…
- Revision-locked plus run times. Clang, LLVM and the others that de…
- Revision-locked only. The mono-repo is more about cross-project commit,…
- None of those options would work for us, but we can deal with it.
- Any of those options would be unacceptable.

Git mono-repo

- *I'd have to dump any WIP to patches and rebuild it in the new repo. That's a pain but it's only a one-time thing.*
- *Mono-repo would discourage contributions and work on projects that use LLVM by student and hobby developers because it focuses on too many things in one main repository.*
- *Mono-repo enables us to perform cross-project code reviews and commits easily, which is useful when performing work downstream*
- *We need to review upstream commits (e.g. for malicious code) before pulling them into our internal systems. Pulling mono-repo in means more commits -> more work.*
- *For Rust we're unlikely to ever use, for example clang, so forcing working with Rust to clone all the LLVM projects would cause us to simply use the sub-repo mirrors to minimize cloning size.*
- *The only issue I have is with moving each sub-project having its own top-level directory.*
- *The mono repo proposal makes a lot of sense for clang developers and anyone else who tends to work with multiple repos simultaneously*
- *It does nicely solve some revision-lock issues with LLVM and Clang, but if another sizable project like Clang came up that was not originally part of the mono-repo it's not clear what is the path forward with integrating that project into the mono-repo.*
- *It's unclear how we might adapt... as I mention elsewhere, we already operate in a multi-repo world and have decent solutions for all the issues that the multi-repo scenario introduces, and further change isn't necessarily better.*