Improve disk space recovery for partial clones

Mentors: Christian Couder, Karthik Nayak, Justin Tobler, Siddharth Asthana, Ayush Chandekar.
Project Size: 175 hours
Difficulty: Medium to Hard.

Personal Information

Name: Amisha Chhajed
Email: amishhhaaaa@gmail.com
github: https://github.com/amishhaa
Time Zone: UTC +5:30 (IST)
Education: SVKM's Dwarkadas J. Sanghvi College of Engineering
Year: 3rd year, 6th semester
Degree: Bachelor of Technology in Artificial Intelligence and Data Science

About me and past experience

Hello, I am Amisha, currently in my penultimate year of engineering.I am deeply passionate about contributing to open source and find great fulfillment when I see my code helping people. Given this motivation, I have contributed to various open source projects and the experience has been extremely rewarding and ethereal.
Apart from open source, I make art and I like building games.

I am currently doing my LFX at OpenTelemetry, my project is about building a GO CLI tool that runs tests of the dependents of a GO library to record any regressions caused by new changes in the library (more about my project: https://mentorship.lfx.linuxfoundation.org/project/5f537fc2-548b-487a-99ed-c61f7e8bcd47) inspired by Rust's Crater(https://github.com/rust-lang/crater), this is a project/tool that I really love as it is the first time i have made something E2E with a huge impact.

Read my blog about my OpenSource journey and LFX here https://amisha.pika.page/posts/how-i-started-with-opensource-and-lfx-month-1
Some open source contributions that I am most proud of are in bitcoin core(refer credits: https://bitcoincore.org/en/releases/29.2/ my PR: https://github.com/bitcoin/bitcoin/pull/33482) and git :)

My contributions in git

cat-file: exit code of 'git cat-file' is suppressed by piping it directly into grep.

Status: Awaiting review

Mailing List: https://lore.kernel.org/git/20260113180409.36683-1-amishhhaaaa@gmail.com/

Log: This was the first patch I ever created for git, made me familiar with the mailing list workflow.

sparse-checkout: optimize string_list construction and add tests to verify deduplication.

Status: merged in 'master'

Mailing List: https://lore.kernel.org/git/20260121130005.72375-1-amishhhaaaa@gmail.com/

Log: This is a really important patch for me, improves O(n^2) complexity to O(n log n) of sparse-checkout by building a sorted 'string_list' by constructing it unsorted then sorting it followed by removing duplicates, this triggered a series of patches and uncovered various bugs when i worked on replacing calls of string_list_sort() and string_list_remove_duplicates() with string_list_sort_u().

u-string-list: add unit tests for string-list methods.

Status: merged in 'master'

Mailing List: https://lore.kernel.org/git/20260129121220.69267-1-amishhhaaaa@gmail.com/

Log: Adding unit tests for string-list methods which i saw were not present when i was creating a new API string_list_sort_u.

string-list: add string_list_sort_u() that mimics "sort -u"

Status: merged in 'master'

Mailing List: https://lore.kernel.org/git/20260129121220.69267-2-amishhhaaaa@gmail.com/

Log: Adding a new API string_list_sort_u and cleaning up the callsites that can directly adopt this new replacement, string_list_sort_u mimics sort -u.

sparse-checkout: use string_list_sort_u

Status: merged in 'master'

Mailing List: https://lore.kernel.org/git/20260212041017.91370-2-amishhhaaaa@gmail.com/

Log: Small fix to replace a callsite of string_list_sort and string_list_remove_duplicates with string_list_sort_u.

help: cleanup the construction of keys_uniq

Status: merged in 'master'

Mailing List: https://lore.kernel.org/git/20260311192453.62213-1-amishhhaaaa@gmail.com/

Log: Cleaning up complex callsites of string_list_sort and string_list_remove_duplicates, this one involved finding a test case that demonstarted a breakage, https://lore.kernel.org/git/CAPvEtrenMBMFaMxcCR4VwoyMFU-_Z+bqq5nJaWv5eyn3HRutEA@mail.gmail.com/, then replacing them with string_list_sort_u, another bug found as an effort is https://lore.kernel.org/git/CAPvEtrfEZXHxcDf=z60ODfUA8cS81rhF1y7KEZApEBby7aCa1A@mail.gmail.com/, this is still a pending bug which is good to work around in future, I have provided a test to demonstrate an existing breakage.

This patch also had collaboration with patch https://lore.kernel.org/git/20260310160029.44605-1-r.siddharth.shrimali@gmail.com/ to make a combined patch of good parts https://lore.kernel.org/git/CAPvEtrd9Yri5LQu9DiMAO4EDquyd-JxwNBGn+h=+=E+oKJ2ERw@mail.gmail.com/,

History/Background, Overview and Project Theory

Background of Partial Clone

The "Partial Clone" feature is a performance optimization for Git that allows Git to function without having a complete copy of the repository, building partial clone was a community effort, with contributions referenced here, https://git-scm.com/docs/partial-clone#_related_links. Most of my knowledge of this project came from researching and reading blogs like https://github.blog/open-source/git/get-up-to-speed-with-partial-clone-and-shallow-clone/. The talk https://www.youtube.com/watch?v=YdstUWcg5j4 by Derrick Stolee are good sources to learn about how git currently handles objects.

There are 4 types of clones that we support, full clone, blobless clone, treeless clone and shallow clone, each of them have their own set of configurations that tells rest of git

what objects are OK to be missing from repository as they are promised, git stores objects in .git/ folder under objects/, these objects can be of 4 types, tag, tree, commits and blobs. When we partial clone, git writes the partial clone type in config file inside .git/, for example,

```
(base) amishachhajed@Amishas-MacBook-Pro .git % cat config
[core]
	repositoryformatversion = 1
	filemode = true
	bare = false
	logallrefupdates = true
	ignorecase = true
	precomposeunicode = true
[remote "origin"]
	url = https://github.com/python/cpython.git
	fetch = +refs/heads/*:refs/remotes/origin/*
	promisor = true
	partialclonefilter = tree:0
[branch "main"]
	remote = origin
	merge = refs/heads/main
```

Each promised object has to be on some of the remote otherwise our partial clone is broken and we do not account for broken remotes. However overtime a lot of objects might be fetched and currently there is no functionality that, first checks if an object is promised and then we can remove it safely to free up our disk space, fundamentally users who make a partial clone want to save up space so being able to remove the acquired objects on demand would be a great addition.

Proposed Plan

Project abstract

This project aims to implement a command 'git evict' that can carry out safe removal of promised objects from our disk space such that it can be fetched later on from any of the remotes if needed, dynamically.

Whenever we are evicting something we are unsure of its usefulness to users, unless obvious, which git gc already handles. To overcome this, we can make a new command 'git evict', essentially this command gives users options and freedom to evict objects that they no longer require, either it is out of cone or not on a checkout and a lot of other situations, instead of predicting what users may or may not need we can give

them the choice of removal. In a case where users have a cone and have set up a maintenance task of evicting objects outside of cone, in that case we can make it automatic.

Why is it okay to evict without cross-checking with the promisor?

As git currently treats remotes as source of truth, we also trust those remotes when evicting objects because if all the remotes break the principal of not strictly presenting the promised objects then the partial clone is inherently broken, we can display a warning that remote might not be able to fetch your file again if it gets unavailable on the remote while evicting, however by making a partial clone we have already placed our trust on the remotes.

Even if we do query the remote to check if the object we are attempting to evict is present, the behaviour is still not deterministic as if the remote breaks the exact second after the status 'OK' is delivered to us we would evict the object and the user would loose it without any warning showed to them when the main goal of checking with the remotes was to make sure the object presented is there deterministically, and if it is not then the partial clone is broken and git already treats one as such when an object we want to fetch is not present on any of the remotes.

Implementation details and proof of concept

In the implementation of pack-objects, the method `want_object_in_pack()` is present from where we can skip any object we want and not include it in the pack. We can use `is_promisor_object()` to verify if the object we are attempting to evict is promised and can be fetched later.

Locally i attempted implementing '`git evict OID_TO_EVICT`' which is available here https://github.com/git/git/compare/master...amishhaa:git:par-c. We will improve how OIDs are currently being passed for skip in `want_object_in_pack()`. Steps to test it out:
Partial clone a repository blobless,

```
(base) amishachhajed@Amishas-MacBook-Pro % git clone --filter=blob:none
https://github.com/python/cpython.git
```

Get OID for a blob, here i am getting it for README,

```
(base) amishachhajed@Amishas-MacBook-Pro cpython % git rev-parse
HEAD:README.rst
1d2874e9ca4fdcbfc5ef14b0202dd73d707bc9ec
```

cat-file the blob to make sure it is present on disk

```
(base) amishachhajed@Amishas-MacBook-Pro cpython % git cat-file -t
1d2874e9ca4fdcbfc5ef14b0202dd73d707bc9ec
remote: Enumerating objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 1 (from 1)
Receiving objects: 100% (1/1), 3.50 KiB | 3.50 MiB/s, done.
blob
```

Run the git evict command with the object oid we recieved from the previous step.

```
(base) amishachhajed@Amishas-MacBook-Pro cpython % git evict
1d2874e9ca4fdcbfc5ef14b0202dd73d707bc9ec
DEBUG: Evicted 1d2874e9ca4fdcbfc5ef14b0202dd73d707bc9ec from pack entry search
Enumerating objects: 810655, done.
Counting objects: 100% (810654/810654), done.
Delta compression using up to 12 threads
Compressing objects: 100% (199618/199618), done.
Writing objects: 100% (810654/810654), done.
Total 810654 (delta 608469), reused 810654 (delta 608469), pack-reused 0 (from 0)
```

Attempt to cat-file again.
(Without network it should fail at fetching)

```
(base) amishachhajed@Amishas-MacBook-Pro cpython % git cat-file -t
1d2874e9ca4fdcbfc5ef14b0202dd73d707bc9ec
fatal: unable to access 'https://github.com/python/cpython.git/': Could not
resolve host: github.com
fatal: could not fetch 1d2874e9ca4fdcbfc5ef14b0202dd73d707bc9ec from promisor
remote
```

(With network it would re-fetch)

```
(base) amishachhajed@Amishas-MacBook-Pro cpython % git cat-file -t
1d2874e9ca4fdcbfc5ef14b0202dd73d707bc9ec
remote: Enumerating objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 1 (from 1)
Receiving objects: 100% (1/1), 3.50 KiB | 3.50 MiB/s, done.
blob
```

To enhance the current implementation I have locally we need validation on an OID, showing a detailed message like 'cannot evict commit object', and check if a hash exists before attempting to evict it while showing necessary error messages.

```
(base) amishachhajed@Amishas-MacBook-Pro cpython % git rev-parse HEAD^{commit}
306c556fdbe7034c9a6d87516534eecdb911ad11
(base) amishachhajed@Amishas-MacBook-Pro cpython % git evict
306c556fdbe7034c9a6d87516534eecdb911ad11
```

```
(base) amishachhajed@Amishas-MacBook-Pro cpython % git evict
306c556fdbe7034c9a6d87516534eecdb911ad11
Enumerating objects: 810831, done.
```

```
Counting objects: 100% (810831/810831), done.
Delta compression using up to 12 threads
Compressing objects: 100% (200113/200113), done.
Writing objects: 100% (810831/810831), done.
Total 810831 (delta 608149), reused 810831 (delta 608149), pack-reused 0 (from
0)
fatal: bad object refs/heads/main
```

So ideally when a user calls git evict, it should internally finds an oidset with objects to evict based on the flag the user has set, and then skip those objects in `want_object_in_pack()` method, so we would be able to evict everything in a single repack operation.

How will we pass the OIDs to skip?

Currently on the user interface I am planning to implement these flags, inspired heavily from reversing how we partial clone, as an example when partial cloning I set blob:limit=<size> then I might want to evict blobs greater than that size in future.

- --outside-cone: Evict objects that are not part of the current sparse-checkout
- --large-only=<size>: Evict objects that are bigger than a certain size.
- --outside-checkout: Evict objects that are not part of the current checkout.

OIDs for outside-cone and outside-checkout can be found by checking for the SKIP_WORKTREE bit and objects greater than a certain size by using rev-list.

Project Deliverables

- Given an object, we can use the method `is_promisor_object()` to check if it is promised, if yes we can repack our packfiles without that object, we would skip those OIDs in `want_object_in_pack()`.
  These OIDs that we have to skip will be passed from evict to `want_object_in_pack()` method.

- Now comes gathering the list of objects for oidset, for each command written above we would need a different method to find and append the object to evict, in oidset, and then pass the oidset for eviction.

- Also add git evict methods to git maintenance so users can set this as background tasks optionally.

Project Timeline

May 1st - May 31st (Community Bonding)

- Discuss project ideas and design implementation details with mentors, possible subcommands to keep and the overall architecture of command, any **optimizations we can make when repacking objects**.
- Study different types of partial clones and how refs work, and which OIDs I can evict in different types of flags we are planning to implement for git evict.

June 1st - June 28th

- Implement the basic git evict command.
- Implement evict_objects method, this evict objects method would need to check if the object is promised and pass it for skipping.

June 29th - July 26th

- Implement different ways to gather OIDs for the flag that we are referencing.
- Pass on those OIDs to evict_objects method which adds the object to evict to oidset.

July 27 – August 16

- Add 'git evict' methods to git maintenance.
- This is a buffer period for any unforeseen delays and prepare a final report of everything we have accomplished over the summer.

August 17 – August 24 (Finalisation of the project)

- Final submission, documentation and buffer period for unexpected delays.

Availability

I would be dedicating 45 hours per week of my time to this project weekly. I don't have any other commitments apart from LFX in month of May which would be easy to manage given my summer break, so during the month of May my time commitment would be 25 hours to this project, in case of extensions I have my internship break next semester so this would be considered as that hence I do not have any university work which would be interfering with the timeline even after my summer break.

Post-GSOC and Appreciation

I want my journey with git to be a long one. It is very fulfilling for me to see my code running on many devices. It is like a dream come true for me, so even post GSOC I intend to keep contributing to git. I would also write blogs as a token of appreciation, because most of my knowledge for this project came from blogs and I want to do something similar. My blog page: https://amisha.pika.page/

Contributing to git uptill now has been insanely rewarding. I learned about commands that I did not even know existed. It feels magical to know the internals of a tool I have been using for years and be a part of the making as well.

I would like to thank the reviewers and maintainers, without them my journey wouldn't have been so easy. I learned some of my best software development lessons on this very list which I am insanely grateful for. Thank you.