multi-pack-index

- Design state
- File layout
- Object positions for incremental MIDXs
- Pseudo-pack order for incremental MIDXs
- Reachability bitmaps and incremental MIDXs

Multi-Pack-Index (MIDX) Design Notes ====================================

The Git object directory contains a 'pack' directory containing packfiles (with suffix ".pack") and pack-indexes (with suffix ".idx"). The pack-indexes provide a way to lookup objects and navigate to their offset within the pack, but these must come in pairs with the packfiles. This pairing depends on the file names, as the pack-index differs only in suffix with its pack- file. While the pack-indexes provide fast lookup per packfile, this performance degrades as the number of packfiles increases, because abbreviations need to inspect every packfile and we are more likely to have a miss on our most-recently-used packfile. For some large repositories, repacking into a single packfile is not feasible due to storage space or excessive repack times.

The multi-pack-index (MIDX for short) stores a list of objects and their offsets into multiple packfiles. It contains:

- A list of packfile names.
- A sorted list of object IDs.
- A list of metadata for the ith object ID including:
  - A value j referring to the jth packfile.
  - An offset within the jth packfile for the object.
- If large offsets are required, we use another list of large offsets similar to version 2 pack-indexes.
  - An optional list of objects in pseudo-pack order (used with MIDX bitmaps).

Thus, we can provide O(log N) lookup time for any number of packfiles.

Design Details ----------

- The MIDX is stored in a file named 'multi-pack-index' in the .git/objects/pack directory. This could be stored in the pack directory of an alternate. It refers only to packfiles in that same directory.

- The core.multiPackIndex config setting must be on (which is the default) to consume MIDX files. Setting it to `false` prevents Git from reading a MIDX file, even if one exists.

- The file format includes parameters for the object ID hash function, so a future change of hash algorithm does not require a change in format.

- The MIDX keeps only one record per object ID. If an object appears in multiple packfiles, then the MIDX selects the copy in the preferred packfile, otherwise selecting from the most-recently modified packfile.

- If there exist packfiles in the pack directory not registered in the MIDX, then those packfiles are loaded into the `packed_git` list and `packed_git_mru` cache.

- The pack-indexes (.idx files) remain in the pack directory so we can delete the MIDX file, set core.midx to false, or downgrade without any loss of information.

- The MIDX file format uses a chunk-based approach (similar to the commit-graph file) that allows optional data to be added.

Incremental multi-pack indexes ----------

As repositories grow in size, it becomes more expensive to write a multi-pack index (MIDX) that includes all packfiles. To accommodate this, the "incremental multi-pack indexes" feature allows for combining a "chain" of multi-pack indexes.

Each individual component of the chain need only contain a small number of packfiles. Appending to the chain does not invalidate earlier parts of the chain, so repositories can control how much time is spent updating the MIDX chain by determining the number of packs in each layer of the MIDX chain.

Design state

At present, the incremental multi-pack indexes feature is missing two important components:

- The ability to rewrite earlier portions of the MIDX chain (i.e., to "compact" some collection of adjacent MIDX layers into a single MIDX). At present the only supported way of shrinking a MIDX chain is to rewrite the entire chain from scratch without the `--split` flag.
  There are no fundamental limitations that stand in the way of being able to implement this feature. It is omitted from the initial implementation in order to reduce the complexity, but will be added later.

- Support for reachability bitmaps. The classic single MIDX implementation does support reachability bitmaps (see the section titled "multi-pack-index reverse indexes" in

linkgit:gitformat-pack[5] for more details).
As above, there are no fundamental limitations that stand in the way of extending the incremental MIDX format to support reachability bitmaps. The design below specifically takes this into account, and support for reachability bitmaps will be added in a future patch series. It is omitted from the current implementation for the same reason as above.
In brief, to support reachability bitmaps with the incremental MIDX feature, the concept of the pseudo-pack order is extended across each layer of the incremental MIDX chain to form a concatenated pseudo-pack order. This concatenation takes place in the same order as the chain itself (in other words, the concatenated pseudo-pack order for a chain `{$H1, $H2, $H3}` would be the pseudo-pack order for `$H1`, followed by the pseudo-pack order for `$H2`, followed by the pseudo-pack order for `$H3`).
The layout will then be extended so that each layer of the incremental MIDX chain can write a `*.bitmap`. The objects in each layer's bitmap are offset by the number of objects in the previous layers of the chain.

File layout

Instead of storing a single `multi-pack-index` file (with an optional `.rev` and `.bitmap` extension) in `$GIT_DIR/objects/pack`, incremental MIDXs are stored in the following layout:

```
$GIT_DIR/objects/pack/multi-pack-index.d/
$GIT_DIR/objects/pack/multi-pack-index.d/multi-pack-index-chain
$GIT_DIR/objects/pack/multi-pack-index.d/multi-pack-index-$H1.midx
$GIT_DIR/objects/pack/multi-pack-index.d/multi-pack-index-$H2.midx
$GIT_DIR/objects/pack/multi-pack-index.d/multi-pack-index-$H3.midx
```

The `multi-pack-index-chain` file contains a list of the incremental MIDX files in the chain, in order. The above example shows a chain whose `multi-pack-index-chain` file would contain the following lines:

```
$H1
$H2
$H3
```

The `multi-pack-index-$H1.midx` file contains the first layer of the multi-pack-index chain. The `multi-pack-index-$H2.midx` file contains the second layer of the chain, and so on.

When both an incremental- and non-incremental MIDX are present, the non-incremental MIDX is always read first.

Object positions for incremental MIDXs

In the original multi-pack-index design, we refer to objects via their lexicographic position (by object IDs) within the repository's singular multi-pack-index. In the incremental multi-pack-index design, we refer to objects via their index into a concatenated lexicographic ordering among each component in the MIDX chain.

If `objects_nr()` is a function that returns the number of objects in a given MIDX layer, then the index of an object at lexicographic position `i` within, say, $H3 is defined as:

```
objects_nr($H2) + objects_nr($H1) + i
```

(in the C implementation, this is often computed as `i` `m→num_objects_in_base`).

Pseudo-pack order for incremental MIDXs

The original implementation of multi-pack reachability bitmaps defined the pseudo-pack order in linkgit:gitformat-pack[5] (see the section titled "multi-pack-index reverse indexes") roughly as follows:

> In short, a MIDX's pseudo-pack is the de-duplicated concatenation of objects in packs stored by the MIDX, laid out in pack order, and the packs arranged in MIDX order (with the preferred pack coming first).

In the incremental MIDX design, we extend this definition to include objects from multiple layers of the MIDX chain. The pseudo-pack order for incremental MIDXs is determined by concatenating the pseudo-pack ordering for each layer of the MIDX chain in order. Formally two objects `o1` and `o2` are compared as follows:

1. If `o1` appears in an earlier layer of the MIDX chain than `o2`, then `o1` sorts ahead of `o2`.

2. Otherwise, if `o1` and `o2` appear in the same MIDX layer, and that MIDX layer has no base, then if one of `pack(o1)` and `pack(o2)` is preferred and the other is not, then the preferred one sorts ahead of the non-preferred one. If there is a base layer (i.e. the MIDX layer is not the first layer in the chain), then if `pack(o1)` appears earlier in that MIDX layer's pack order, then `o1` sorts ahead of `o2`. Likewise if `pack(o2)` appears earlier, then the opposite is true.

3. Otherwise, `o1` and `o2` appear in the same pack, and thus in the same MIDX layer. Sort `o1` and `o2` by their offset within their containing packfile.

Note that the preferred pack is a property of the MIDX chain, not the individual layers themselves. Fundamentally we could introduce a per-layer preferred pack, but this is less relevant now that we can perform multi-pack reuse across the set of packs in a MIDX.

Reachability bitmaps and incremental MIDXs

Each layer of an incremental MIDX chain may have its objects (and the objects from any previous layer in the same MIDX chain) represented in its own `*.bitmap` file.

The structure of a `*.bitmap` file belonging to an incremental MIDX chain is identical to that of a non-incremental MIDX bitmap, or a classic single-pack bitmap. Since objects are added to the end of the incremental MIDX's pseudo-pack order (see above), it is possible to extend a bitmap when appending to the end of a MIDX chain.

(Note: it is possible likewise to compress a contiguous sequence of MIDX incremental layers, and their `*.bitmap` files into a single layer and `*.bitmap`, but this is not yet implemented.)

The object positions used are global within the pseudo-pack order, so subsequent layers will have, for example, `m→num_objects_in_base` number of `0` bits in each of their four type bitmaps. This follows from the fact that we only write type bitmap entries for objects present in the layer immediately corresponding to the bitmap).

Note also that only the bitmap pertaining to the most recent layer in an incremental MIDX chain is used to store reachability information about the interesting and uninteresting objects in a reachability query. Earlier bitmap layers are only used to look up commit and pseudo-merge bitmaps from that layer, as well as the type-level bitmaps for objects in that layer.

To simplify the implementation, type-level bitmaps are iterated simultaneously, and their results are OR'd together to avoid recursively calling internal bitmap functions.

Future Work ----------

- If the multi-pack-index is extended to store a "stable object order" (a function Order(hash) = integer that is constant for a given hash, even as the multi-pack-index is updated) then MIDX bitmaps could be updated independently of the MIDX.

- Packfiles can be marked as "special" using empty files that share the initial name but replace ".pack" with ".keep" or ".promisor". We can add an optional chunk of data to the multi-pack-index that records flags of information about the packfiles. This allows new states, such as ’repacked’ or ’redeltified’, that can help with pack maintenance in a multi-pack environment. It may also be helpful to organize packfiles by object type (commit, tree, blob, etc.) and use this metadata to help that maintenance.

Related Links ---------- [0] https://bugs.chromium.org/p/git/issues/detail?id=6 Chromium work item for: Multi-Pack Index (MIDX)

[1] https://lore.kernel.org/git/20180107181459.222909-1-dstolee@microsoft.com/ An earlier RFC for the multi-pack-index feature

[2] https://lore.kernel.org/git/alpine.DEB.2.20.1803091557510.23109@alexmv-linux/ Git Merge 2018 Contributor’s summit notes (includes discussion of MIDX)

□□□ #2
□ □□□ □□□ 2026-10-31 20:16:37 UTC
□ □□□ □□□ 2026-10-31 20:16:39 UTC