gitformat-chunk

gitformat-chunk(5) ==========

NAME ---- gitformat-chunk - Chunk-based file formats

SYNOPSIS --------

Used by linkgit:gitformat-commit-graph[5] and the "MIDX" format (see the pack format documentation in linkgit:gitformat-pack[5]).

DESCRIPTION -----------

Some file formats in Git use a common concept of "chunks" to describe sections of the file. This allows structured access to a large file by scanning a small "table of contents" for the remaining data. This common format is used by the `commit-graph` and `multi-pack-index` files. See the `multi-pack-index` format in linkgit:gitformat-pack[5] and the `commit-graph` format in linkgit:gitformat-commit-graph[5] for how they use the chunks to describe structured data.

A chunk-based file format begins with some header information custom to that format. That header should include enough information to identify the file type, format version, and number of chunks in the file. From this information, that file can determine the start of the chunk-based region.

The chunk-based region starts with a table of contents describing where each chunk starts and ends. This consists of (C+1) rows of 12 bytes each, where C is the number of chunks. Consider the following table:

```
| Chunk ID (4 bytes) | Chunk Offset (8 bytes) |
|--------------------|------------------------|
| ID[0]              | OFFSET[0]              |
| ...                | ...                    |
| ID[C]              | OFFSET[C]              |
| 0x0000             | OFFSET[C+1]            |
```

Each row consists of a 4-byte chunk identifier (ID) and an 8-byte offset. Each integer is stored in network-byte order.

The chunk identifier `ID[i]` is a label for the data stored within this file from `OFFSET[i]` (inclusive) to `OFFSET[i+1]` (exclusive). Thus, the size of the `i`th chunk is equal to the difference between `OFFSET[i+1]` and `OFFSET[i]`. This requires that the chunk data appears contiguously in the same order as the table of contents.

The final entry in the table of contents must be four zero bytes. This confirms that the table of contents is ending and provides the offset for the end of the chunk-based data.

Note: The chunk-based format expects that the file contains *at least* a trailing hash after `OFFSET[C+1]`.

Functions for working with chunk-based file formats are declared in `chunk-format.h`. Using these methods provide extra checks that assist developers when creating new file formats.

Writing chunk-based file formats ----------

To write a chunk-based file format, create a `struct` `chunkfile` by calling `init_chunkfile()` and pass a `struct` `hashfile` pointer. The caller is responsible for opening the `hashfile` and writing header information so the file format is identifiable before the chunk-based format begins.

Then, call `add_chunk()` for each chunk that is intended for writing. This populates the `chunkfile` with information about the order and size of each chunk to write. Provide a `chunk_write_fn` function pointer to perform the write of the chunk data upon request.

Call `write_chunkfile()` to write the table of contents to the `hashfile` followed by each of the chunks. This will verify that each chunk wrote the expected amount of data so the table of contents is correct.

Finally, call `free_chunkfile()` to clear the `struct` `chunkfile` data. The caller is responsible for finalizing the `hashfile` by writing the trailing hash and closing the file.

Reading chunk-based file formats ----------

To read a chunk-based file format, the file must be opened as a memory-mapped region. The chunk-format API expects that the entire file is mapped as a contiguous memory region.

Initialize a `struct` `chunkfile` pointer with `init_chunkfile(NULL)`.

After reading the header information from the beginning of the file, including the chunk count, call `read_table_of_contents()` to populate the `struct` `chunkfile` with the list of chunks, their offsets, and their sizes.

Extract the data information for each chunk using `pair_chunk()` or `read_chunk()`:

- `pair_chunk()` assigns a given pointer with the location inside the memory-mapped file corresponding to that chunk's offset. If the chunk does not exist, then the pointer is not modified.

- `read_chunk()` takes a `chunk_read_fn` function pointer and calls it with the appropriate initial pointer and size information. The function is not called if the chunk does not exist. Use this method to read chunks if you need to perform immediate parsing or if you need to execute logic based on the size of the chunk.

After calling these methods, call `free_chunkfile()` to clear the `struct` `chunkfile` data. This will not close the memory-mapped region. Callers are expected to own that data for the timeframe the pointers into the region are needed.

Examples --------

These file formats use the chunk-format API, and can be used as examples for future formats:

- **commit-graph:** see `write_commit_graph_file()` and `parse_commit_graph()` in `commit-graph.c` for how the chunk-format API is used to write and parse the commit-graph file format documented in the commit-graph file format in linkgit:gitformat-commit-graph[5].

- **multi-pack-index:** see `write_midx_internal()` and `load_multi_pack_index()` in `midx.c` for how the chunk-format API is used to write and parse the multi-pack-index file format documented in the multi-pack-index file format section of linkgit:gitformat-pack[5].

GIT --- Part of the linkgit:git[1] suite

□□□ #1
□ □□□ □□□ 2026-10-31 20:13:58 UTC
□ □□□ □□□ 2026-10-31 20:13:58 UTC