gitformat-index

- The Git index file has the following format
- Index entry
- Extensions
  - Cache tree
  - Resolve undo
  - Split index
- Untracked cache
- File System Monitor cache
- End of Index Entry
- Index Entry Offset Table
- Sparse Directory Entries

gitformat-index(5) ==========

NAME ---- gitformat-index - Git index format

SYNOPSIS --------

> $GIT_DIR/index

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

Git index format

The Git index file has the following format

```
All binary numbers are in network byte order.
In a repository using the traditional SHA-1, checksums and object IDs
(object names) mentioned below are all computed using SHA-1.  Similarly,
in SHA-256 repositories, these values are computed using SHA-256.
Version 2 is described here unless stated otherwise.
```

- A 12-byte header consisting of

```
4-byte signature:
  The signature is { 'D', 'I', 'R', 'C' } (stands for "dircache")

4-byte version number:
  The current supported versions are 2, 3 and 4.

32-bit number of index entries.
```

- A number of sorted index entries (see below).
- Extensions

```
Extensions are identified by signature. Optional extensions can
be ignored if Git does not understand them.

4-byte extension signature. If the first byte is 'A'..'Z' the
extension is optional and can be ignored.

32-bit size of the extension

Extension data
```

- Hash checksum over the content of the index file before this checksum.

Index entry

```
Index entries are sorted in ascending order on the name field,
interpreted as a string of unsigned bytes (i.e. memcmp() order, no
localization, no special casing of directory separator '/'). Entries
with the same name are sorted by their stage field.
```

```
An index entry typically represents a file. However, if sparse-checkout
is enabled in cone mode (`core.sparseCheckoutCone` is enabled) and the
`extensions.sparseIndex` extension is enabled, then the index may
contain entries for directories outside of the sparse-checkout definition.
These entries have mode `040000`, include the `SKIP_WORKTREE` bit, and
the path ends in a directory separator.

32-bit ctime seconds, the last time a file's metadata changed
  this is stat(2) data

32-bit ctime nanosecond fractions
  this is stat(2) data

32-bit mtime seconds, the last time a file's data changed
  this is stat(2) data

32-bit mtime nanosecond fractions
  this is stat(2) data

32-bit dev
  this is stat(2) data

32-bit ino
  this is stat(2) data

32-bit mode, split into (high to low bits)

16-bit unused, must be zero

4-bit object type
  valid values in binary are 1000 (regular file), 1010 (symbolic link)
  and 1110 (gitlink)

3-bit unused, must be zero

9-bit unix permission. Only 0755 and 0644 are valid for regular files.
Symbolic links and gitlinks have value 0 in this field.

32-bit uid
```

```
  this is stat(2) data

32-bit gid
  this is stat(2) data

32-bit file size
  This is the on-disk size from stat(2), truncated to 32-bit.

Object name for the represented object

A 16-bit 'flags' field split into (high to low bits)

1-bit assume-valid flag

1-bit extended flag (must be zero in version 2)

2-bit stage (during merge)

12-bit name length if the length is less than 0xFFF; otherwise 0xFFF
is stored in this field.

(Version 3 or later) A 16-bit field, only applicable if the
"extended flag" above is 1, split into (high to low bits).

1-bit reserved for future

1-bit skip-worktree flag (used by sparse checkout)

1-bit intent-to-add flag (used by "git add -N")

13-bit unused, must be zero

Entry path name (variable length) relative to top level directory
  (without leading slash). '/' is used as path separator. The special
  path components ".", ".." and ".git" (without quotes) are disallowed.
  Trailing slash is also disallowed.

The exact encoding is undefined, but the '.' and '/' characters
are encoded in 7-bit ASCII and the encoding cannot contain a NUL
byte (iow, this is a UNIX pathname).
```

```
(Version 4) In version 4, the entry path name is prefix-compressed
  relative to the path name for the previous entry (the very first
  entry is encoded as if the path name for the previous entry is an
  empty string).  At the beginning of an entry, an integer N in the
  variable width encoding (the same encoding as the offset is encoded
  for OFS_DELTA pack entries; see linkgit:gitformat-pack[5]) is stored, followed
  by a NUL-terminated string S.  Removing N bytes from the end of the
  path name for the previous entry, and replacing it with the string S
  yields the path name for this entry.

1-8 nul bytes as necessary to pad the entry to a multiple of eight bytes
while keeping the name NUL-terminated.

(Version 4) In version 4, the padding after the pathname does not
exist.

Interpretation of index entries in split index mode is completely
different. See below for details.
```

Extensions

Cache tree

```
Since the index does not record entries for directories, the cache
entries cannot describe tree objects that already exist in the object
database for regions of the index that are unchanged from an existing
commit. The cache tree extension stores a recursive tree structure that
describes the trees that already exist and completely match sections of
the cache entries. This speeds up tree object generation from the index
for a new commit by only computing the trees that are "new" to that
commit. It also assists when comparing the index to another tree, such
as `HEAD^{tree}`, since sections of the index can be skipped when a tree
comparison demonstrates equality.

The recursive tree structure uses nodes that store a number of cache
```

```
entries, a list of subnodes, and an object ID (OID). The OID references
the existing tree for that node, if it is known to exist. The subnodes
correspond to subdirectories that themselves have cache tree nodes. The
number of cache entries corresponds to the number of cache entries in
the index that describe paths within that tree's directory.

The extension tracks the full directory structure in the cache tree
extension, but this is generally smaller than the full cache entry list.

When a path is updated in index, Git invalidates all nodes of the
recursive cache tree corresponding to the parent directories of that
path. We store these tree nodes as being "invalid" by using "-1" as the
number of cache entries. Invalid nodes still store a span of index
entries, allowing Git to focus its efforts when reconstructing a full
cache tree.

The signature for this extension is { 'T', 'R', 'E', 'E' }.

A series of entries fill the entire extension; each of which
consists of:
```

- NUL-terminated path component (relative to its parent directory);
- ASCII decimal number of entries in the index that is covered by the tree this entry represents (entry_count);
- A space (ASCII 32);
- ASCII decimal number that represents the number of subtrees this tree has;
- A newline (ASCII 10); and
- Object name for the object that would result from writing this span of index as a tree.

```
An entry can be in an invalidated state and is represented by having
a negative number in the entry_count field. In this case, there is no
object name and the next entry starts immediately after the newline.
When writing an invalid entry, -1 should always be used as entry_count.

The entries are written out in the top-down, depth-first order.  The
first entry represents the root level of the repository, followed by the
```

```
first subtree--let's call this A--of the root level (with its name
relative to the root level), followed by the first subtree of A (with
its name relative to A), and so on. The specified number of subtrees
indicates when the current level of the recursive stack is complete.
```

Resolve undo

```
A conflict is represented in the index as a set of higher stage entries.
When a conflict is resolved (e.g. with "git add path"), these higher
stage entries will be removed and a stage-0 entry with proper resolution
is added.

When these higher stage entries are removed, they are saved in the
resolve undo extension, so that conflicts can be recreated (e.g. with
"git checkout -m"), in case users want to redo a conflict resolution
from scratch.

The signature for this extension is { 'R', 'E', 'U', 'C' }.

A series of entries fill the entire extension; each of which
consists of:
```

- NUL-terminated pathname the entry describes (relative to the root of the repository, i.e. full pathname);
- Three NUL-terminated ASCII octal numbers, entry mode of entries in stage 1 to 3 (a missing stage is represented by "0" in this field); and
- At most three object names of the entry in stages from 1 to 3 (nothing is written for a missing stage).

Split index

```
In split index mode, the majority of index entries could be stored
in a separate file. This extension records the changes to be made on
top of that to produce the final index.
```

```
The signature for this extension is { 'l', 'i', 'n', 'k' }.

The extension consists of:
```

- Hash of the shared index file. The shared index file path is $GIT_DIR/sharedindex.<hash>. If all bits are zero, the index does not require a shared index file.
- An ewah-encoded delete bitmap, each bit represents an entry in the shared index. If a bit is set, its corresponding entry in the shared index will be removed from the final index. Note, because a delete operation changes index entry positions, but we do need original positions in replace phase, it’s best to just mark entries for removal, then do a mass deletion after replacement.
- An ewah-encoded replace bitmap, each bit represents an entry in the shared index. If a bit is set, its corresponding entry in the shared index will be replaced with an entry in this index file. All replaced entries are stored in sorted order in this index. The first "1" bit in the replace bitmap corresponds to the first index entry, the second "1" bit to the second entry and so on. Replaced entries may have empty path names to save space.

```
The remaining index entries after replaced ones will be added to the
final index. These added entries are also sorted by entry name then
stage.
```

Untracked cache

```
Untracked cache saves the untracked file list and necessary data to
verify the cache. The signature for this extension is { 'U', 'N',
'T', 'R' }.

The extension starts with
```

- A sequence of NUL-terminated strings, preceded by the size of the sequence in variable width encoding. Each string describes the environment where the cache can be used.
- Stat data of $GIT_COMMON_DIR/info/exclude. See "Index entry" section from ctime field until "file size".
- Stat data of core.excludesFile
- 32-bit dir_flags (see struct dir_struct)
- Hash of $GIT_COMMON_DIR/info/exclude. A null hash means the file does not exist.

- Hash of core.excludesFile. A null hash means the file does not exist.
- NUL-terminated string of per-dir exclude file name. This usually is ".gitignore".
- The number of following directory blocks, variable width encoding. If this number is zero, the extension ends here with a following NUL.
- A number of directory blocks in depth-first-search order, each consists of
- The number of untracked entries, variable width encoding.
- The number of sub-directory blocks, variable width encoding.
- The directory name terminated by NUL.
- A number of untracked file/dir names terminated by NUL.

The remaining data of each directory block is grouped by type:

- An ewah bitmap, the n-th bit marks whether the n-th directory has valid untracked cache entries.
- An ewah bitmap, the n-th bit records "check-only" bit of read_directory_recursive() for the n-th directory.
- An ewah bitmap, the n-th bit indicates whether hash and stat data is valid for the n-th directory and exists in the next data.
- An array of stat data. The n-th data corresponds with the n-th "one" bit in the previous ewah bitmap.
- An array of hashes. The n-th hash corresponds with the n-th "one" bit in the previous ewah bitmap.
- One NUL.

File System Monitor cache

```
The file system monitor cache tracks files for which the core.fsmonitor
hook has told us about changes.  The signature for this extension is
{ 'F', 'S', 'M', 'N' }.

The extension starts with
```

- 32-bit version number: the current supported versions are 1 and 2.
- (Version 1) 64-bit time: the extension data reflects all changes through the given time which is stored as the nanoseconds elapsed since midnight, January 1, 1970.
- (Version 2) A null terminated string: an opaque token defined by the file system monitor application. The extension data reflects all changes relative to that token.
- 32-bit bitmap size: the size of the CE_FSMONITOR_VALID bitmap.
- An ewah bitmap, the n-th bit indicates whether the n-th index entry is not CE_FSMONITOR_VALID.

End of Index Entry

```
The End of Index Entry (EOIE) is used to locate the end of the variable
length index entries and the beginning of the extensions. Code can take
advantage of this to quickly locate the index extensions without having
to parse through all of the index entries.

Because it must be able to be loaded before the variable length cache
entries and other index extensions, this extension must be written last.
The signature for this extension is { 'E', 'O', 'I', 'E' }.

The extension consists of:
```

- 32-bit offset to the end of the index entries
- Hash over the extension types and their sizes (but not their contents). E.g. if we have "TREE" extension that is N-bytes long, "REUC" extension that is M-bytes long, followed by "EOIE", then the hash would be:

```
Hash("TREE" + <binary-representation-of-N> +
    "REUC" + <binary-representation-of-M>)
```

Index Entry Offset Table

```
The Index Entry Offset Table (IEOT) is used to help address the CPU
cost of loading the index by enabling multi-threading the process of
converting cache entries from the on-disk format to the in-memory format.
The signature for this extension is { 'I', 'E', 'O', 'T' }.

The extension consists of:
```

- 32-bit version (currently 1)
- A number of index offset entries each consisting of:
- 32-bit offset from the beginning of the file to the first cache entry in this block of entries.
- 32-bit count of cache entries in this block

Sparse Directory Entries

```
When using sparse-checkout in cone mode, some entire directories within
the index can be summarized by pointing to a tree object instead of the
entire expanded list of paths within that tree. An index containing such
entries is a "sparse index". Index format versions 4 and less were not
implemented with such entries in mind. Thus, for these versions, an
index containing sparse directory entries will include this extension
with signature { 's', 'd', 'i', 'r' }. Like the split-index extension,
tools should avoid interacting with a sparse index unless they understand
this extension.
```

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

□□□ #1
□ □□□ □□□ 2026-10-31 20:14:01 UTC
□ □□□ □□□ 2026-10-31 20:14:01 UTC