

Git – Why should I care about the index?

Charles Bailey

Bloomberg LP

12th April 2013

Why should I care about the index?

Introduction

What is Git?

The Git object model

- ▶ blob
- ▶ tree
- ▶ commit
- ▶ tag

Synonyms for the index

- ▶ index
- ▶ dircache
- ▶ cache
- ▶ staging area

Yes, but what is it?

- ▶ binary file
- ▶ maps paths to blob ids
- ▶ caches lstat information

What is it for?

- ▶ Uses cached information to speed up tree operations (e.g. diff, status)
- ▶ Record the versions of files checked out
- ▶ Record the identity of files being checked in
- ▶ Record multiple versions of files being merged

Where do I find it?

- ▶ Only applies to non-bare repositories
- ▶ Usually found at `.git/index`
- ▶ Path can be overridden with the `GIT_INDEX_FILE` environment variable

Visualizing the index

- ▶ Like a commit, the index contains all currently tracked files
- ▶ Like a commit, we often only look at the changes implied by the index
- ▶ `git status` shows tracked changes that differ between the current state of the index and `HEAD`

Viewing the index

- ▶ `hexdump .git/index - use`
Documentation/technical/index-format.txt
- ▶ `git ls-files -s`
- ▶ `git ls-files --debug`

Delete it!

- ▶ Doesn't delete any metadata
- ▶ Easy to recreate
- ▶ May delete difficult to recreate metadata

Recreating an index

```
git reset
```

or

```
git read-tree <tree-object>
```

Config variable: `core.preloadindex`

Making a commit

```
git write-tree
```

and (now nothing to do with the index)

```
git commit-tree
```

Updating the index

- ▶ `git add`
- ▶ `git rm --cached`
- ▶ `git update-index`

Updating the index (from the database)

- ▶ `git reset <treeish> <file>`

or

- ▶ `git update-index --cacheinfo`

Updating the working tree

- ▶ `git checkout -- <file>`
- ▶ `git checkout-index`

Index “slots”

- ▶ Normally only slot 0 is populated
- ▶ In a merge, slots 1, 2 and 3 are used instead

Index “slots”

- ▶ Slot 1 - common base
- ▶ Slot 2 - “ours” (base branch in rebase)
- ▶ Slot 3 - “theirs” (feature branch in rebase)

Any of these slots may be empty, e.g. for baseless merge or “other sided delete” conflicts.

Merge resolution

The act of replacing entries in slots 1, 2 and 3 with a single resolved slot 0 entry.

- ▶ Automatically done on successful resolution with `git mergetool`
- ▶ Manually, with `git add`

Assume unchanged

```
git update-index --[no-]assume-unchanged <file>
```

Config variable: `core.ignoreStat`

Skip worktree

```
git update-index --[no-]skip-worktree <file>
```

Executable bit

```
git update-index --chmod=(+|-)x <file>
```

Config variable: `core.fileMode`

Intent to add

```
git add -N <new file>
```

- ▶ precached trees
- ▶ resolve undo

Changing an entry

Make a path refer to a different object

```
git update-index --cacheinfo <mode> <object-id> <path>
```

Use --add if the path is a new entry

Changing an entry - part ii

Update the index without adding the file contents to the repository

```
git update-index --info-only <path>
```

Use --add if the path is a new entry

DANGER!

```
git hash-object -w <path>
```

Batch update

```
git update-index --index-info
```

- ▶ Reads from stdin
- ▶ Can update index entries other than zero

Recreating a unresolved merge state

```
git update-index --index-info
```

- ▶ Write mode 0 to delete the slot 0 entry first
- ▶ Add entries for slots 1, 2 and 3 afterwards

Recreating the unresolved state the easy way

```
git ls-files --resolve-undo <path>
```

```
git update-index --unresolve <path>
```

```
git merge-index git-merge-one-file <path>
```

Q&A