
git-revise

Release 0.7.0

Jan 05, 2022

Contents:

1	Installing	3
2	git-revise (1) – Efficiently update, split, and rearrange git commits	5
2.1	SYNOPSIS	5
2.2	DESCRIPTION	5
2.3	OPTIONS	5
2.4	CONFIGURATION	6
2.5	CONFLICT RESOLUTION	7
2.6	NOTES	7
2.7	INTERACTIVE MODE	7
2.8	REPORTING BUGS	8
2.9	SEE ALSO	8
3	Performance	9
3.1	How is it faster?	10
4	The gitrevise module	11
4.1	merge – Quick in-memory merges	11
4.2	odb – Git object database interaction	11
4.3	todo – History edit sequences	16
4.4	tui – git-revise entry point	16
4.5	utils – Misc. helper methods	16
5	Contributing	17
5.1	Running Tests	17
5.2	Code Formatting	17
5.3	Building Documentation	17
5.4	Publishing	18
	Python Module Index	19
	Index	21

`git-revise` is a *git (1)* subcommand, and *python (1)* library for efficiently updating, splitting, and rearranging commits.

git revise is open-source, and can be found on [GitHub](#)

CHAPTER 1

Installing

git-revise can be installed from [PyPi](#). Python 3.8 or higher is required.

```
$ pip install --user git-revise
```

git-revise(1) – Efficiently update, split, and rearrange git commits

2.1 SYNOPSIS

git revise [<options>] [<target>]

2.2 DESCRIPTION

git revise is a *git (1)* subcommand to efficiently update, split, and rearrange commits. It is heavily inspired by *git-rebase (1)*, however tries to be more efficient and ergonomic for patch-stack oriented workflows.

By default, **git revise** will apply staged changes to <target>, updating HEAD to point at the revised history. It also supports splitting commits, rewording commit messages.

Unlike *git-rebase (1)*, **git revise** avoids modifying working directory and index state, performing all merges in-memory, and only writing them when necessary. This allows it to be significantly faster on large codebases, and avoid invalidating builds.

If *--autosquash* or *--interactive* is specified, the <target> argument may be omitted or given as the special value *:option:-root*. If it is omitted, **git revise** will consider a range of unpublished commits on the current branch. If given as *:option:-root*, all commits including the root commit will be considered.

2.3 OPTIONS

2.3.1 General options

-a, --all

Stage changes to tracked files before revising.

-p, --patch

Interactively stage hunks from the worktree before revising.

--no-index

Ignore staged changes in the index.

--reauthor

Reset target commit's author to the current user.

--ref <gitref>

Working branch to update; defaults to HEAD.

-S, --gpg-sign, --no-gpg-sign

GPG-sign commits. Overrides both the `commit.gpgSign` and `revise.gpgSign` git configurations.

2.3.2 Main modes of operation

-i, --interactive

Rather than applying staged changes to <target>, edit a todo list of actions to perform on commits after <target>. See [INTERACTIVE MODE](#).

--autosquash, --no-autosquash

Rather than directly applying staged changes to <target>, automatically perform fixup or squash actions marked with `fixup!` or `squash!` between <target> and the current HEAD. For more information on what these actions do, see [INTERACTIVE MODE](#).

These commits are usually created with `git commit --fixup=<commit>` or `git commit --squash=<commit>`, and identify the target with the first line of its commit message.

This option can be combined with `--interactive` to modify the generated todos before they're executed.

If the `--autosquash` option is enabled by default using a configuration variable, the option `--no-autosquash` can be used to override and disable this setting. See [CONFIGURATION](#).

-c, --cut

Interactively select hunks from <target>. The chosen hunks are split into a second commit immediately after the target.

After splitting is complete, both commits' messages are edited.

See the "Interactive Mode" section of `git-add(1)` to learn how to operate this mode.

-e, --edit

After applying staged changes, edit <target>'s commit message.

This option can be combined with `--interactive` to allow editing of commit messages within the todo list. For more information on, see [INTERACTIVE MODE](#).

-m <msg>, --message <msg>

Use the given <msg> as the new commit message for <target>. If multiple `-m` options are given, their values are concatenated as separate paragraphs.

--version

Print version information and exit.

2.4 CONFIGURATION

Configuration is managed by `git-config(1)`.

revise.autoSquash

If set to true, imply `--autosquash` whenever `--interactive` is specified. Overridden by `--no-autosquash`. Defaults to false. If not set, the value of `rebase.autoSquash` is used instead.

revise.gpgSign

If set to true, GPG-sign new commits; defaults to false. This setting overrides the original git configuration `commit.gpgSign` and may be overridden by the command line options `--gpg-sign` and `--no-gpg-sign`.

2.5 CONFLICT RESOLUTION

When a conflict is encountered, **git revise** will attempt to resolve it automatically using standard git mechanisms. If automatic resolution fails, the user will be prompted to resolve them manually.

There is currently no support for using *git-mergetool* (1) to resolve conflicts.

No attempt is made to detect renames of files or directories. **git revise** may produce suboptimal results across renames. Use the interactive mode of *git-rebase* (1) when rename tracking is important.

2.6 NOTES

A successful **git revise** will add a single entry to the reflog, allowing it to be undone with `git reset @{1}`. Unsuccessful **git revise** commands will leave your repository largely unmodified.

No merge commits may occur between the target commit and HEAD, as rewriting them is not supported.

See *git-rebase* (1) for more information on the implications of modifying history on a repository that you share.

2.7 INTERACTIVE MODE

git revise supports an interactive mode inspired by the interactive mode of *git-rebase* (1).

This mode is started with the last commit you want to retain “as-is”:

```
git revise -i <after-this-commit>
```

The special target `-root` is available to revise everything up to the root commit:

```
git revise -i --root
```

An editor will be fired up with the commits in your current branch after the given commit. If the index has any staged but uncommitted changes, a `<git index>` entry will also be present.

```
pick 8338dfa88912 Online summary of first commit
pick 735609912343 Summary of second commit
index 672841329981 <git index>
```

These commits may be re-ordered to change the order they appear in history. In addition, the `pick` and `index` commands may be replaced to modify their behaviour. If present, `index` commands must be at the bottom of the list, i.e. they can not be followed by non-index commands.

If `-e` was specified, the full commit message will be included, and each command line will begin with a `++`. Any changes made to the commit messages in this file will be applied to the commit in question, allowing for simultaneous editing of commit messages during the todo editing phase.

```
++ pick 8338dfa88912
Online summary of first commit

Body of first commit

++ pick 735609912343
Summary of second commit

Body of second commit

++ index 672841329981
<git index>
```

The following commands are supported in all interactive modes:

index

Do not commit these changes, instead leaving them staged in the index. Index lines must come last in the file.

pick

Use the given commit as-is in history. When applied to the generated `index` entry, the commit will have the message `<git index>`.

squash

Add the commit's changes into the previous commit and open an editor to merge the commits' messages.

fixup

Like squash, but discard this commit's message rather than editing.

reword

Open an editor to modify the commit message.

cut

Interactively select hunks from the commit. The chosen hunks are split into a second commit immediately after it.

After splitting is complete, both commits' messages are edited.

See the “Interactive Mode” section of *git-add(1)* to learn how to operate this mode.

2.8 REPORTING BUGS

Please report issues and feature requests to the issue tracker at <https://github.com/mystor/git-revise/issues>.

Code, documentation and other contributions are also welcomed.

2.9 SEE ALSO

git(1) *git-rebase(1)* *git-add(1)*

CHAPTER 3

Performance

Note: These numbers are from an earlier version, and may not reflect the current state of *git-revise*.

With large repositories such as `mozilla-central`, **git revise** is often significantly faster than *git-rebase(1)* for incremental, due to not needing to update the index or working directory during rebases.

I did a simple test, applying a single-line change to a commit 11 patches up the stack. The following are my extremely non-scientific time measurements:

Command	Real Time
<code>git rebase -i --autosquash</code>	16.931s
<code>git revise</code>	0.541s

The following are the commands I ran:

```
# Apply changes with git rebase -i --autosquash
$ git reset 6fceb7da316d && git add .
$ time bash -c 'TARGET=14f1c85bf60d; git commit --fixup=$TARGET; EDITOR=true git_
↪rebase -i --autosquash $TARGET~'
<snip>

real    0m16.931s
user    0m15.289s
sys     0m3.579s

# Apply changes with git revise
$ git reset 6fceb7da316d && git add .
$ time git revise 14f1c85bf60d
<snip>

real    0m0.541s
user    0m0.354s
sys     0m0.150s
```

3.1 How is it faster?

In-Memory Cache

To avoid spawning unnecessary subprocesses and hitting disk too frequently, **git revise** uses an in-memory cache of objects in the ODB which it has already seen.

Intermediate git trees, blobs, and commits created during processing are held exclusively in-memory, and only persisted when necessary.

Custom Merge Algorithm

A custom implementation of the merge algorithm is used which directly merges trees rather than using the index. This ends up being faster on large repositories, as only the subset of modified files and directories need to be examined when merging.

Note: Currently this algorithm is incapable of handling copy and rename operations correctly, instead treating them as file creation and deletion actions. This may be resolveable in the future.

Avoiding Index & Working Directory

The working directory and index are never examined or updated during the rebasing process, avoiding disk I/O and invalidating existing builds.

The gitrevise module

Python modules for interacting with git objects used by *git-revise(1)* – *Efficiently update, split, and rearrange git commits.*

4.1 merge – Quick in-memory merges

This module contains a basic implementation of an efficient, in-memory 3-way git tree merge. This is used rather than traditional git mechanisms to avoid needing to use the index file format, which can be slow to initialize for large repositories.

The INDEX file for my local mozilla-central checkout, for reference, is 35MB. While this isn't huge, it takes a perceptible amount of time to read the tree files and generate. This algorithm, on the other hand, avoids looking at unmodified trees and blobs when possible.

exception gitrevise.merge.**ConflictParseFailed**

exception gitrevise.merge.**MergeConflict**

4.2 odb – Git object database interaction

Helper classes for reading cached objects from Git's Object Database.

class gitrevise.odb.**Blob**

In memory representation of a git blob object

class gitrevise.odb.**Commit**

In memory representation of a git commit object

author

Signature of this commit's author

committer

Signature of this commit's committer

gpgsig
GPG signature of this commit

is_root
Whether this commit has no parents

message
Body of this commit's message

parent () → gitrevise.odb.Commit
Helper method to get the single parent of a commit. Raises `ValueError` if the incorrect number of parents are present.

parent_oids
List of *Oid* for this commit's parents

parent_tree () → gitrevise.odb.Tree
`tree` object corresponding to the first parent of this commit, or the null tree if this is a root commit

parents () → Sequence[gitrevise.odb.Commit]
List of parent commits

rebase (parent: Optional[gitrevise.odb.Commit]) → gitrevise.odb.Commit
Create a new commit with the same changes, except with `parent` as its parent. If `parent` is `None`, this becomes a root commit.

summary () → str
The summary line of the commit message. Returns the summary as a single line, even if it spans multiple lines.

tree () → gitrevise.odb.Tree
`tree` object corresponding to this commit

tree_oid
Oid of this commit's `tree` object

update (tree: Optional[gitrevise.odb.Tree] = None, parents: Optional[Sequence[gitrevise.odb.Commit]] = None, message: Optional[bytes] = None, author: Optional[gitrevise.odb.Signature] = None, recommit: bool = False) → gitrevise.odb.Commit
Create a new commit with specific properties updated or replaced

class gitrevise.odb.**Entry** (repo: gitrevise.odb.Repository, mode: gitrevise.odb.Mode, oid: gitrevise.odb.Oid)
In memory representation of a single `tree` entry

blob () → gitrevise.odb.Blob
Get the data for this entry as a *Blob*

mode
Mode of the entry

oid
Oid of this entry's object

persist () → None
`GitObj.persist()` the git object referenced by this entry

repo
Repository this entry originates from

symlink () → bytes
Get the data for this entry as a symlink


```

tree () → gitrevise.odb.Tree
    Get the data for this entry as a Tree

exception gitrevise.odb.GPGSError(stderr: str)
    Exception raised when we fail to sign a commit

class gitrevise.odb.GitObj
    In-memory representation of a git object. Instances of this object should be one of Commit, Tree or Blob

    body
        Raw body of object in bytes

    oid
        Oid of this git object

    persist () → gitrevise.odb.Oid
        If this object has not been persisted to disk yet, persist it

    persisted
        If True, the object has been persisted to disk

    repo
        Repository object is associated with

class gitrevise.odb.Index(repo: gitrevise.odb.Repository, index_file: Optional[pathlib.Path] = None)
    Handle on an index file

    commit (message: bytes = b'<git index>', parent: Optional[gitrevise.odb.Commit] = None) → gitrevise.odb.Commit
        Get a Commit for this index's state. If parent is None, use the current HEAD

    git (*cmd, cwd: Optional[pathlib.Path] = None, stdin: Optional[bytes] = None, trim_newline: bool = True, env: Optional[Mapping[str, str]] = None, nocapture: bool = False) → bytes
        Invoke git with the given index as active

    index_file = None
        Index file being referenced

    repo = None

    tree () → gitrevise.odb.Tree
        Get a Tree object for this index's state

exception gitrevise.odb.MissingObject(ref: str)
    Exception raised when a commit cannot be found in the ODB

class gitrevise.odb.Mode
    Mode for an entry in a tree

    DIR = b'40000'
        directory entry

    EXEC = b'100755'
        executable entry

    GITLINK = b'160000'
        submodule entry

    REGULAR = b'100644'
        regular entry

    SYMLINK = b'120000'
        symlink entry

```

```
class gitrevise.odb.Oid
    Git object identifier

    classmethod for_object (tag: str, body: bytes) → gitrevise.odb.Oid
        Hash an object with the given type tag and body to determine its Oid

    classmethod fromhex (instr: str) → gitrevise.odb.Oid
        Parse an Oid from a hexadecimal string

    classmethod null () → gitrevise.odb.Oid
        An Oid consisting of entirely 0s

    short () → str
        A shortened version of the Oid's hexadecimal form

class gitrevise.odb.Reference (obj_type: Type[GitObjT], repo: gitrevise.odb.Repository, name: str)
    A git reference

    name = None
        Resolved reference name, e.g. 'refs/tags/1.0.0' or 'refs/heads/master'

    refresh () → None
        Re-read the target of this reference from disk

    repo = None
        Repository reference is attached to

    shortname = None
        Short unresolved reference name, e.g. 'HEAD' or 'master'

    target = None
        Referenced git object

    update (new: GitObjT, reason: str) → None
        Update this reference to point to a new object. An entry with the reason reason will be added to the
        reflog.

class gitrevise.odb.Repository (cwd: Optional[pathlib.Path] = None)
    Main entry point for a git repository

    default_author
        author used by default for new commits

    default_committer
        committer used by default for new commits

    get_blob (ref: Union[gitrevise.odb.Oid, str]) → gitrevise.odb.Blob
        Like get_obj(), but returns a Blob

    get_blob_ref (ref: str) → gitrevise.odb.Reference[gitrevise.odb.Blob][gitrevise.odb.Blob]
        Get a Reference to a Blob

    get_commit (ref: Union[gitrevise.odb.Oid, str]) → gitrevise.odb.Commit
        Like get_obj(), but returns a Commit

    get_commit_ref (ref: str) → gitrevise.odb.Reference[gitrevise.odb.Commit][gitrevise.odb.Commit]
        Get a Reference to a Commit

    get_obj (ref: Union[gitrevise.odb.Oid, str]) → gitrevise.odb.GitObj
        Get the identified git object from this repository. If given an Oid, the cache will be checked before asking
        git.
```

```

get_obj_ref (ref: str) → gitrevise.oddb.Reference[gitrevise.oddb.GitObj][gitrevise.oddb.GitObj]
    Get a Reference to a GitObj

get_tempdir () → pathlib.Path
    Return a temporary directory to use for modifications to this repository

get_tree (ref: Union[gitrevise.oddb.Oid, str]) → gitrevise.oddb.Tree
    Like get_obj(), but returns a Tree

get_tree_ref (ref: str) → gitrevise.oddb.Reference[gitrevise.oddb.Tree][gitrevise.oddb.Tree]
    Get a Reference to a Tree

git_path (path: Union[str, pathlib.Path]) → pathlib.Path
    Get the path to a file in the .git directory, respecting the environment

gitdir
    .git directory for this repository

gpg
    path to GnuPG binary

index
    current index state

new_commit (tree: gitrevise.oddb.Tree, parents: Sequence[gitrevise.oddb.Commit], message:
    bytes, author: Optional[gitrevise.oddb.Signature] = None, committer: Optional[gitrevise.oddb.Signature] = None) → gitrevise.oddb.Commit
    Directly create an in-memory commit object, without persisting it. If a commit object with these properties
    already exists, it will be returned instead.

new_tree (entries: Mapping[bytes, gitrevise.oddb.Entry]) → gitrevise.oddb.Tree
    Directly create an in-memory tree object, without persisting it. If a tree object with these entries already
    exists, it will be returned instead.

sign_buffer (buffer: bytes) → bytes
    Return the text of the signed commit object.

sign_commits
    sign commits with gpg

workdir
    working directory for this repository

class gitrevise.oddb.Signature
    Git user signature

    email
        user email

    name
        user name

    offset
        timezone offset from UTC

    signing_key
        user name <email>

    timestamp
        unix timestamp

class gitrevise.oddb.Tree
    In memory representation of a git tree object

```

entries

mapping from entry names to entry objects in this tree

to_index (*path*: *pathlib.Path*, *skip_worktree*: *bool* = *False*) → *gitrevise.odb.Index*

Read tree into a temporary index. If *skip_workdir* is *True*, every entry in the index will have its “Skip Workdir” bit set.

4.3 todo – History edit sequences

class *gitrevise.todo.StepKind*

An enumeration.

gitrevise.todo.validate_todos (*old*: *List[gitrevise.todo.Step]*, *new*: *List[gitrevise.todo.Step]*) → *None*

Raise an exception if the new todo list is malformed compared to the original todo list

4.4 tui – git-revise entry point

4.5 utils – Misc. helper methods

exception *gitrevise.utils.EditorError*

gitrevise.utils.commit_range (*base*: *Optional[gitrevise.odb.Commit]*, *tip*: *gitrevise.odb.Commit*) → *List[gitrevise.odb.Commit]*

Oldest-first iterator over the given commit range, not including the commit base

gitrevise.utils.cut_commit (*commit*: *gitrevise.odb.Commit*) → *gitrevise.odb.Commit*

Perform a cut operation on the given commit, and return the modified commit.

gitrevise.utils.edit_commit_message (*commit*: *gitrevise.odb.Commit*) → *gitrevise.odb.Commit*

Launch an editor to edit the commit message of *commit*, returning a modified commit

gitrevise.utils.local_commits (*repo*: *gitrevise.odb.Repository*, *tip*: *gitrevise.odb.Commit*) → *Tuple[gitrevise.odb.Commit, List[gitrevise.odb.Commit]]*

Returns an oldest-first iterator over the local commits which are parents of the specified commit. May return an empty list. A commit is considered local if it is not present on any remote.

gitrevise.utils.run_editor (*repo*: *gitrevise.odb.Repository*, *filename*: *str*, *text*: *bytes*, *comments*: *Optional[str]* = *None*, *allow_empty*: *bool* = *False*) → *bytes*

Run the editor configured for git to edit the given text

gitrevise.utils.run_sequence_editor (*repo*: *gitrevise.odb.Repository*, *filename*: *str*, *text*: *bytes*, *comments*: *Optional[str]* = *None*, *allow_empty*: *bool* = *False*) → *bytes*

Run the editor configured for git to edit the given rebase/revise sequence

gitrevise.utils.run_specific_editor (*editor*: *str*, *repo*: *gitrevise.odb.Repository*, *filename*: *str*, *text*: *bytes*, *comments*: *Optional[str]* = *None*, *allow_empty*: *bool* = *False*, *allow_whitespace_before_comments*: *bool* = *False*) → *bytes*

Run the editor configured for git to edit the given text

gitrevise.utils.sh_run (*cmd*: *Sequence[Any]*, **args*, ***kwargs*) → *CompletedProcess[Any]*

Run a command within git’s shell environment. This is the same as *subprocess.run* on most platforms, but will enter the git-bash mingw environment on Windows.

5.1 Running Tests

tox is used to run tests. It will run **mypy** for type checking, **pylint** for linting, **pytest** for testing, and **black** for code formatting.

```
$ tox                # All python versions
$ tox -e py38        # Python 3.8
$ tox -e py39        # Python 3.9
$ tox -e py310       # Python 3.10

$ tox -e mypy        # Mypy Typechecking
$ tox -e pylint      # Linting
$ tox -e black       # Check Formatting
```

5.2 Code Formatting

This project uses **black** for code formatting.

```
$ black .  # format all python code
```

5.3 Building Documentation

Documentation is built using **sphinx**.

```
$ cd docs/
$ make man  # Build manpage
```

5.4 Publishing

```
$ python3 setup.py sdist bdist_wheel
$ twine check dist/*
$ twine upload dist/*
```

g

- `gitrevise.merge`, [11](#)
- `gitrevise.odb`, [11](#)
- `gitrevise.todo`, [16](#)
- `gitrevise.tui`, [16](#)
- `gitrevise.utils`, [16](#)

Symbols

-autosquash, -no-autosquash
 git-revise: command line option, 6
 -no-index
 git-revise: command line option, 5
 -reauthor
 git-revise: command line option, 6
 -ref <gitref>
 git-revise: command line option, 6
 -version
 git-revise: command line option, 6
 -S, -pgp-sign, -no-gpg-sign
 git-revise: command line option, 6
 -a, -all
 git-revise: command line option, 5
 -c, -cut
 git-revise: command line option, 6
 -e, -edit
 git-revise: command line option, 6
 -i, -interactive
 git-revise: command line option, 6
 -m <msg>, -message <msg>
 git-revise: command line option, 6
 -p, -patch
 git-revise: command line option, 5

A

author (*gitrevise.oddb.Commit attribute*), 11

B

Blob (*class in gitrevise.oddb*), 11
 blob() (*gitrevise.oddb.Entry method*), 12
 body (*gitrevise.oddb.GitObj attribute*), 13

C

Commit (*class in gitrevise.oddb*), 11
 commit() (*gitrevise.oddb.Index method*), 13
 commit_range() (*in module gitrevise.utils*), 16
 committer (*gitrevise.oddb.Commit attribute*), 11

ConflictParseFailed, 11
 cut_commit() (*in module gitrevise.utils*), 16

D

default_author (*gitrevise.oddb.Repository attribute*), 14
 default_committer (*gitrevise.oddb.Repository attribute*), 14
 DIR (*gitrevise.oddb.Mode attribute*), 13

E

edit_commit_message() (*in module gitrevise.utils*), 16
 EditorError, 16
 email (*gitrevise.oddb.Signature attribute*), 15
 entries (*gitrevise.oddb.Tree attribute*), 15
 Entry (*class in gitrevise.oddb*), 12
 EXEC (*gitrevise.oddb.Mode attribute*), 13

F

for_object() (*gitrevise.oddb.Oid class method*), 14
 fromhex() (*gitrevise.oddb.Oid class method*), 14

G

get_blob() (*gitrevise.oddb.Repository method*), 14
 get_blob_ref() (*gitrevise.oddb.Repository method*), 14
 get_commit() (*gitrevise.oddb.Repository method*), 14
 get_commit_ref() (*gitrevise.oddb.Repository method*), 14
 get_obj() (*gitrevise.oddb.Repository method*), 14
 get_obj_ref() (*gitrevise.oddb.Repository method*), 14
 get_tempdir() (*gitrevise.oddb.Repository method*), 15
 get_tree() (*gitrevise.oddb.Repository method*), 15
 get_tree_ref() (*gitrevise.oddb.Repository method*), 15
 git() (*gitrevise.oddb.Index method*), 13
 git-revise: command line option
 -autosquash, -no-autosquash, 6

- no-index, 5
- reauthor, 6
- ref <gitref>, 6
- version, 6
- S, -pgp-sign, -no-gpg-sign, 6
- a, -all, 5
- c, -cut, 6
- e, -edit, 6
- i, -interactive, 6
- m <msg>, -message <msg>, 6
- p, -patch, 5

- git_path() (*gitrevise.odb.Repository method*), 15
- gitdir (*gitrevise.odb.Repository attribute*), 15
- GITLINK (*gitrevise.odb.Mode attribute*), 13
- GitObj (*class in gitrevise.odb*), 13
- gitrevise.merge (*module*), 11
- gitrevise.odb (*module*), 11
- gitrevise.todo (*module*), 16
- gitrevise.tui (*module*), 16
- gitrevise.utils (*module*), 16
- gpg (*gitrevise.odb.Repository attribute*), 15
- gpgsig (*gitrevise.odb.Commit attribute*), 11
- GPSSignError, 13

I

- Index (*class in gitrevise.odb*), 13
- index (*gitrevise.odb.Repository attribute*), 15
- index_file (*gitrevise.odb.Index attribute*), 13
- is_root (*gitrevise.odb.Commit attribute*), 12

L

- local_commits() (*in module gitrevise.utils*), 16

M

- MergeConflict, 11
- message (*gitrevise.odb.Commit attribute*), 12
- MissingObject, 13
- Mode (*class in gitrevise.odb*), 13
- mode (*gitrevise.odb.Entry attribute*), 12

N

- name (*gitrevise.odb.Reference attribute*), 14
- name (*gitrevise.odb.Signature attribute*), 15
- new_commit() (*gitrevise.odb.Repository method*), 15
- new_tree() (*gitrevise.odb.Repository method*), 15
- null() (*gitrevise.odb.Oid class method*), 14

O

- offset (*gitrevise.odb.Signature attribute*), 15
- Oid (*class in gitrevise.odb*), 13
- oid (*gitrevise.odb.Entry attribute*), 12
- oid (*gitrevise.odb.GitObj attribute*), 13

P

- parent() (*gitrevise.odb.Commit method*), 12
- parent_oids (*gitrevise.odb.Commit attribute*), 12
- parent_tree() (*gitrevise.odb.Commit method*), 12
- parents() (*gitrevise.odb.Commit method*), 12
- persist() (*gitrevise.odb.Entry method*), 12
- persist() (*gitrevise.odb.GitObj method*), 13
- persisted (*gitrevise.odb.GitObj attribute*), 13

R

- rebase() (*gitrevise.odb.Commit method*), 12
- Reference (*class in gitrevise.odb*), 14
- refresh() (*gitrevise.odb.Reference method*), 14
- REGULAR (*gitrevise.odb.Mode attribute*), 13
- repo (*gitrevise.odb.Entry attribute*), 12
- repo (*gitrevise.odb.GitObj attribute*), 13
- repo (*gitrevise.odb.Index attribute*), 13
- repo (*gitrevise.odb.Reference attribute*), 14
- Repository (*class in gitrevise.odb*), 14
- run_editor() (*in module gitrevise.utils*), 16
- run_sequence_editor() (*in module gitrevise.utils*), 16
- run_specific_editor() (*in module gitrevise.utils*), 16

S

- sh_run() (*in module gitrevise.utils*), 16
- short() (*gitrevise.odb.Oid method*), 14
- shortname (*gitrevise.odb.Reference attribute*), 14
- sign_buffer() (*gitrevise.odb.Repository method*), 15
- sign_commits (*gitrevise.odb.Repository attribute*), 15
- Signature (*class in gitrevise.odb*), 15
- signing_key (*gitrevise.odb.Signature attribute*), 15
- StepKind (*class in gitrevise.todo*), 16
- summary() (*gitrevise.odb.Commit method*), 12
- SYMLINK (*gitrevise.odb.Mode attribute*), 13
- symlink() (*gitrevise.odb.Entry method*), 12

T

- target (*gitrevise.odb.Reference attribute*), 14
- timestamp (*gitrevise.odb.Signature attribute*), 15
- to_index() (*gitrevise.odb.Tree method*), 16
- Tree (*class in gitrevise.odb*), 15
- tree() (*gitrevise.odb.Commit method*), 12
- tree() (*gitrevise.odb.Entry method*), 12
- tree() (*gitrevise.odb.Index method*), 13
- tree_oid (*gitrevise.odb.Commit attribute*), 12

U

- update() (*gitrevise.odb.Commit method*), 12
- update() (*gitrevise.odb.Reference method*), 14

V

- validate_todos() (*in module gitrevise.todo*), 16

W

`workdir` (*gitrevise.odbc.Repository attribute*), [15](#)