

---

# **maggit Documentation**

***Release 0.1***

**Matthieu Gautier**

January 05, 2016



<b>1</b>	<b>Installing</b>	<b>3</b>
1.1	Installing with pypi . . . . .	3
1.2	Installing from source . . . . .	3
<b>2</b>	<b>Tutorial</b>	<b>5</b>
2.1	Importing Maggit . . . . .	5
2.2	Get a repository . . . . .	5
2.3	References . . . . .	5
2.4	Print the log of the 10th last commit of the master branch . . . . .	6
2.5	Print the content of a file . . . . .	6
2.6	The commit object . . . . .	6
<b>3</b>	<b>maggit package api</b>	<b>7</b>
3.1	maggit.db package . . . . .	7
3.2	maggit.io package . . . . .	10
3.3	maggit.git_objects module . . . . .	16
3.4	maggit.refs module . . . . .	18
<b>4</b>	<b>Indices and tables</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>



Contents:



---

**Installing**

---

## 1.1 Installing with pypi

Maggit is available on pypi. Just use it

```
$ pip install maggit
```

## 1.2 Installing from source

You can directly download sources and install from them

```
$ git clone https://gitlab.com/maggit/maggit.git
$ cd maggit
$ python3 setup.py install .
```

To test that everything is ok

```
$ pip install pytest
$ py.test
```

You are now ready to use Maggit. Read the [Tutorial](#) if you don't know how.





## 2.1 Importing Maggit

This is pretty simple

```
import maggit
```

You can use the star import if you want

```
from maggit import *
```

For the rest of the tutorial, we'll assume that you import maggit and do not use the star import.

## 2.2 Get a repository

A *Repo* is one the central objects when interactig with maggit:

```
# Create a repository
repo = maggit.Repo()
```

By default, Maggit will look for a git repository in your current directory. If you want to explore a repository elsewhere, specify it:

```
repo = maggit.Repo("a/git/repository")
```

You don't have to specify a the root directory of the repository. If the directory you specify is not a git repository, maggit will look up in the parents directories.

## 2.3 References

Once we've got a repository, the first thing we may want to do is list it's references (branches, tags):

```
branches = repo.branches
print(repo.branches.keys())
```

*repo.branches* is a dictionnary mapping all branches of the repository

If we want manipulate the *master* branch, lets get it:

```
master = branches['master']
```

In the same way *repo.tags* is a dictionary mapping all tags of the repository

All the references we get, whatever they are branches, lightweight tag or real tag objects share a common api. The most interesting here, is to get the commit object pointed to by the ref:

```
commit = master.commit
```

## 2.4 Print the log of the 10th last commit of the master branch

We just have to go through the parent of the commit and print the message each time

```
commit = master.commit
for i in range(10):
    print(commit.message)
    commit = commit.parents[0]
```

## 2.5 Print the content of a file

There are several way of doing it.

The low level way is the following:

```
current = commit.tree
for part in path.split('/'):
    current = current.entries[part]
blob = current
print(blob.content)
```

But you can also use a high level api

```
entry = commit.get_file(path)
# the entry is a intermediate object making the link between a gitobject
# and a particular commit
blob = entry.gitObject
print(blob.content)
```

## 2.6 The commit object

Other important object in Maggit is naturally the git objects. A git object firstly allow us to access to the content of the object itself.

The commit object is one of them. Naturally it allows us to access the commit attributes:

```
commit.message
commit.first_line #The first line of the message
commit.author
commit.committer
commit.parents # The parents of the commit
commit.tree # The tree object
```

---

## maggit package api

---

### 3.1 maggitt.db package

This module contains all the low level classes to handle a git repository.

Class defined in this module use io functions and provide a more consistent API to access content of the git repository.

Most users will not use those objects directly but better use the high level API provided by the maggitt package itself.

#### 3.1.1 maggitt.db.db module

**class** `maggitt.db.db.Gitdb` (*rootdir*)

Bases: `object`

The `Gitdb`, unify all loose/pack io function to provide a coherent access to content of a git repository.

A `Gitdb` handle only the reading of git objects. Not the references, remotes, ...

**Parameters** `rootdir` (*path*) – The path of the objects directory (`.git/objects`)

**blob\_content** (*sha*)

Read and parse a object assuming it is a blob.

**Parameters** `sha` – The sha of the object.

**Returns** The content of the blob (bytes).

**Raises** `ValueError` – If the object is not a blob.

**blob\_write** (*content*)

**commit\_content** (*sha*)

Read and parse a object assuming it is a commit.

**Parameters** `sha` – The sha of the object.

**Returns**

A tuple (tree, parents, message, author, committer) where:

- tree is the sha of the tree object of the commit(unhexlified bytes).
- parents is a list of sha of the parents commit.
- message is the message of the commit.
- author is the name (b'name <email> timestamp') of the author.

- author is the name (b'name <email> timestamp') of the committer.

**Return type** bytes, list[bytes], bytes, bytes, bytes

**Raises** ValueError – If the object is not a commit.

**commit\_write** (*treeSha, message, author, authorDate, committer, committerDate, parents=[]*)

**gen\_pack\_list** ()

**get\_full\_sha** (*prefix*)

Return the full Sha of the prefix

**Parameters** **prefix** (*bytes*) – The beginning of a sha.

**Returns** The corresponding (bytes).

**Exemples:**

```
>>> repo.get_full_sha(b'bf09f0a9')
<Sha b'bf09f0a9...'>
```

**Raises** :Exception – If number of object corresponding to prefix is not equal to one.

**get\_pack** (*sha*)

Get a pack containing the sha

**Parameters** **sha** – The sha of the object

**Returns** class:~maggit.io.pack.GitPack containing the sha

**Return type** The

**object\_content** (*sha*)

**object\_exists** (*sha*)

Return True if the sha exists in the db

**object\_type** (*sha*)

Return the type of the object associated to sha.

**Parameters** **sha** – The sha of the object.

**Returns** The type of the object.

**tag\_content** (*sha*)

Read and parse a object assuming it is a tag.

**Parameters** **sha** – The sha of the object.

**Returns**

A tuple (object, objecttype, tag, tagger, message) where:

- object is the sha of the tagged object (unhexlified bytes).
- objecttype is the type of the tagged object.
- tag is the name of the tag.
- tagger is the name (b'name <email> timestamp') of the tagger.
- message is the message of the tag.

**Return type** bytes, bytes, bytes, bytes, bytes

**Raises** ValueError – If the object is not a tag.

**tag\_write** (*objectsha*, *tag*, *tagger*, *tagDate*, *message*)

**tree\_content** (*sha*)

Read and parse a object assuming it is a tree.

**Parameters** *sha* – The sha of the object.

**Returns**

A list of (path, (mode, sha)) where :

- path is the name of the entry.
- mode is the git mode.
- sha is the sha of the blob/tree object.

**Return type** List[Tuple[bytes, Tuple[bytes, bytes]]]

**Raises** `ValueError` – If the object is not a tree.

**tree\_write** (*entries*)

### 3.1.2 maggit.db.repo module

**class** `maggit.db.repo.Repo` (*gitdir=None*, *disable\_directoryLooking=False*, *bare=False*)

This is the low level Repository class.

The repo make the link between all other low level subsystems and recreate a coherent database.

**Parameters**

- **gitdir** (*path*) – The directory path of the repository,  
If is None, the current working directory is assumed.
- **disable\_directoryLooking** (*bool*) – If True, assume that the gitdir is a valid path  
so do not search for a valid git repository in parents and gitdir must not be None.
- **bare** (*bool*) – Does the repository is a bare one. Only relevant if `disable_directoryLooking`  
is True. Else, the bare attribute is detected from the git repository structure.

**HEAD**

The current checkout branch

**branches**

A iterator of branches in the repo

**check\_ref\_exists** (*ref*)

Check that a ref exist in the git bdd.

**Parameters** *ref* (*str*) – the ref to check.

**Returns** True if the ref exists, else False

**Return type** bool

**get\_full\_sha** (*value*)

Return the full sha of the value

**classmethod** **get\_git\_dir** (*dirToCheck=None*)

**get\_peel\_ref\_sha** (*ref*)

**get\_ref\_sha** (*ref*)

Return the sha corresponding to the ref.

**Parameters** `ref` (*str*) – the ref to get.

**Returns** The sha corresponding to the ref

**Return type** bytes

**classmethod** `init_repo` (*gitdir*, *bare=False*)

Instantiate a new git repo at the given location.

**tags**

A iterator of tags in the repo

## 3.2 maggit.io package

This module contains all the low level functions necessary to Maggit to read, write and parse all git files.

Thoses files includes git objects, pack, packedrefs, config, index, ...

Thoses function do not provide high level API. There are mainly intended to be use by Maggit itself.

### 3.2.1 maggit.io.loose module

`maggit.io.loose.commit_parse` (*content*)

Parse a commit content.

**Parameters** `content` (*bytes*) – The content to parse (without header).

**Returns**

A tuple (tree, parents, message, author, committer) where:

- tree is the sha of the tree object of the commit(unhexlified bytes).
- parents is a list of sha of the parents commit.
- message is the message of the commit.
- author is the name (b'name <email> timestamp') of the author.
- author is the name (b'name <email> timestamp') of the committer.

**Return type** bytes, list[bytes], bytes, bytes, bytes

`maggit.io.loose.object_content` (*filepath*)

Return the content of a loose object.

**Parameters** `filepath` – The path of the loose object.

**Returns**

A tuple (type, content) where:

- type is the type of the object.
- content is the content of the object (without header).

**Return type** bytes, bytes

`maggit.io.loose.object_rawsha` (*type\_*, *content*)

Generate the raw content and the sha of a object content.

**Parameters**

- `type` (*bytes*) – The type of the object.

- **content** (*bytes*) – The content of the object (without header).

**Returns**

A tuple (raw, sha) where:

- raw is the full content of the object (with header).
- sha is the sha of the object.

**Return type** bytes, bytes

`maggit.io.loose.object_sha(type_, content)`

Generate the sha of a object content.

Is the bit more performant than `object_rawsha(...)[1]` as the raw content is not generated.

**Parameters**

- **type** (*bytes*) – The type of the object.
- **content** (*bytes*) – The content of the object (without header).

**Returns** The sha of the object.

`maggit.io.loose.object_sha_from_raw(raw)`

Generate the sha of a object from its content.

**Parameters** **raw** (*bytes*) – The content of the object (with header)

**Returns** The sha of the object.

`maggit.io.loose.object_write(filepath, content, compress_level=1)`

Correctly create the loose object file.

**Parameters**

- **filepath** – The path of the loose object to write.
- **content** (*bytes*) – The full content (with header) to write.
- **compress\_level** (*int*) – The compression level to use (default=1).

`maggit.io.loose.tag_parse(content)`

Parse a tag content.

**Parameters** **content** (*bytes*) – The content to parse (without header).

**Returns**

A tuple (object, objecttype, tag, tagger, message) where:

- object is the sha of the tagged object (unhexlified bytes).
- objecttype is the type of the tagged object.
- tag is the name of the tag.
- tagger is the name (b' name <email> timestamp') of the tagger.
- message is the message of the tag.

**Return type** bytes, bytes, bytes, bytes, bytes

`maggit.io.loose.tree_parse(content)`

Parse a tree content.

**Parameters** **content** (*bytes*) – The content to parse (without header).

**Returns**

A list of (path, (mode, sha)) where :

- path is the name of the entry.
- mode is the git mode.
- sha is the sha of the blob/tree object.

**Return type** List[Tuple[bytes, Tuple[bytes, bytes]]]

### 3.2.2 maggit.io.pack module

`class maggit.io.pack.GitPack(packfile, idxfile)`

A git pack file.

**Parameters**

- **packfile** (*path*) – The path of the packfile.
- **idxfile** (*GitPackIndex*) – The index associated to the pack.

`read_object(offset)`

Return the content of a object at a offset.

**Parameters** **offset** – The offset to read from.

**Returns**

A tuple (type, content) where:

- type is the type of the object.
- content is the content of the object (without header).

**Return type** bytes, bytes

`class maggit.io.pack.GitPackIndex(indexfile)`

A pack index

**Parameters** **indexfile** (*path*) – The path of the index file.

`get_offset(sha)`

Return the offset in the pack associated to the sha.

### 3.2.3 maggit.io.packedref module

`maggit.io.packedref.packed_ref_parse(filename)`

Parse a packed ref file.

**Parameters** **filename** (*path*) – The path of the packed ref.

**Returns**

A list of (ref, sha, peeledsha) where:

- ref is name of the ref
- sha is the associated sha
- peeledsha is the peeled sha associated to the ref if present. Else None.

**Return type** List[Tuple[bytes, bytes, bytes]]



**class** `maggit.Sha`

Bases: `bytes`

A Sha is a git sha. It means that it is the identifier of a git object.

Sha are store in Maggit as a 20 bytes len.

**hexbytes**

The sha as a hexlified bytes

**hexstr**

The sha as a hexlified str

**class** `maggit.Repo` (*gitdir=None, disable\_directoryLooking=False, bare=False*)

Bases: `maggit.db.repo.Repo`

This is the central piece of a git repository.

**HEAD**

The current checkouted branch

**branches**

A dict of branches in the repo

**check\_ref\_exists** (*ref*)

Check that a ref exist in the git bdd.

**Parameters** **ref** (*str*) – the ref to check.

**Returns** True if the ref exists, else False

**Return type** bool

**get\_blob** (*sha*)

Get a blob object for the sha.

**Parameters** **sha** (*Sha*) – The sha of the blob.

**Returns** class:~*maggit.git\_objects.Blob* object for this sha.

**Return type** A

**Raises** :*Exception* – If sha doesn't name a blob object.

**get\_commit** (*sha*)

Get a commit object for the sha.

**Parameters** **sha** (*Sha*) – The sha of the commit.

**Returns** class:~*maggit.repo.Commit* object for this sha.

**Return type** A

**Raises** :*Exception* – If sha doesn't name a commit object.

**get\_full\_sha** (*prefix*)

Return the full Sha of the prefix

**Parameters** **prefix** (*bytes*) – The beginning of a sha.

**Returns** class:~*maggit.Sha* corresponding.

**Return type** The

**Exemples:**

```
>>> repo.get_full_sha(b'bf09f0a9')
<Sha b'bf09f0a9...'>
```

**Raises** :`Exception` – If number of object corresponding to prefix is not equal to one.

**get\_git\_dir** (*dirToCheck=None*)

**get\_object** (*sha, type\_=None*)

Get a git object for the sha.

**Parameters**

- **sha** (*Sha*) – The sha of the object.
- **type** (*ObjectType* or `None`) – The type of the object. If this is `None` (or not set), the type is detected automatically.

**Returns** class:~maggit.git\_objects.GitObject object for this sha.

**Return type** A

**Raises** :`Exception` – If sha doesn't name a object.

**get\_peel\_ref\_sha** (*ref*)

**get\_ref\_sha** (*name*)

Return the Sha corresponding to the reference name

**Parameters** **name** (*str*) – The reference name.

**Returns** class:~maggit.Sha corresponding.

**Return type** The

**Exemples:**

```
>>> repo.get_ref_sha('master')
<Sha >
```

**Raises** :`Exception` – If number of object corresponding to prefix is not equal to one.

**get\_tag** (*sha*)

Get a tag object for the sha.

**Parameters** **sha** (*Sha*) – The sha of the tag.

**Returns** class:~maggit.git\_objects.Tag object for this sha.

**Return type** A

**Raises** :`Exception` – If sha doesn't name a tag object.

**get\_tree** (*sha*)

Get a tree object for the sha.

**Parameters** **sha** (*Sha*) – The sha of the tree.

**Returns** class:~maggit.git\_objects.Tree object for this sha.

**Return type** A

**Raises** :`Exception` – If sha doesn't name a tree object.

**init\_repo** (*gitdir*, *bare=False*)

Instantiate a new git repo at the given location.

**tags**

A dict of tags in the repo

**class** `maggit.ObjectType`

This is a enum listing possible type for a git object.

**class** `maggit.repo.Entry` (*repo*, *commit*, *path*, *mode*, *gitObject*)

A Entry represent a entry (file or directory) at a specific time.

We can somehow see a repository as a complex 2 dimensionnal array. Commits (and so the history) are rows. Files (and Trees) are columns.

In this situations, Entry are the cells of this array.

**get\_first\_appearance** ()

Get the commit who firstly introduce the current version of the change.

**Returns** class:~*maggit.repo.Commit*

**Return type** A

**parents**

The previous versions of the files.

Previous versions can be equal to the current one if the current commit introduce no change on this file.

The length of the parents will most of the time be 1 but may be greater in case of merge.

**class** `maggit.repo.Commit` (*repo*, *sha*)

**get\_file** (*path*)

Get an entry corresponding to the path in the commit

**Parameters** **path** (*str or Path*) – The path of the file to look at.

**Returns** class:~*maggit.repo.Entry* corresponding.

**Return type** A

**Raise:** `KeyError` if the path is not existing.

**get\_first\_appearance** (*path*)

Return the commit where the present version of path was introduce.

**Parameters** **path** (*str or path*) – The path of the file.

**Returns** class:*maggit.repo.Commit*

**Return type** A

**Exemples:**

```
>>> first_appearance_commit = this_commit.get_first_appearance(path)
>>> # first_appearance_commit is the first one, so previous version differs
>>> parent = first_appearance_commit.parents[0]
>>> assert first_appearance_commit.get_file(path).gitObject != parent.get_file(path).gitObject
>>> # from this_commit to first_appearance_commit, there is no change
>>> current = this_commit
>>> while current != first_appearance_commit:
```

```
...     assert current.get_file(path).gitObject == this_commit.get_file(path).gitObject
...     current = current.parents[0]
```

**get\_first\_appearances** (*root=None, depthLimit=None*)

Return the first appearances for all entry in root.

This is mostly equivalent to *{path:commit.get\_first\_appearance(path) for path in commit.tree.entries}* (if root is None). But a way more performant as diving into history is made once.

#### Parameters

- **root** (*str or Path*) – In wich subdirectory we must look.
- **depthLimit** (*int*) – The commit limit number we go in history If depthLimit is specified, and for a entry the first appearance is older than depthLimit, the entry will be present in the dictionary with a None value.

**Returns** *class:maggit.repo.Commit*).

**Return type** A dict of (Path,

## 3.3 maggit.git\_objects module

**class** *maggit.git\_objects.GitObject* (*repo, sha*)

The base class for all git objects.

Git objects are conceptually constant. However as we try to be lazy, slots are not fill at object creation and set when user read it. So GitObject are not constant but behave as if they were.

**repo**

*Repo*

The repo associated with the object.

**sha**

*Sha*

The sha of the object.

**gitType**

*ObjectType*

The type of the object.

**class** *maggit.git\_objects.Blob* (*repo, sha*)

Bases: *maggit.git\_objects.GitObject*

A blob object.

**content**

*bytes*

This is the content of the blob.

**class** *maggit.git\_objects.Tree* (*repo, sha*)

Bases: *maggit.git\_objects.GitObject*

A blob object.

**entries**

*immutable mapping*

This is the entries of the tree.

**class** `maggit.git_objects.Commit(repo, sha)`

Bases: `maggit.git_objects.GitObject`

A commit object.

**tree**

`maggit.git_objects.Tree`

The tree object associated with the commit.

**parents**

*tuple*

The parents of the commits. Most of the time, there will only one parent. In case of branch merge, there will be more than one parent.

**author**

*bytes*

The author of the commit. (This is the raw content in the commit. This means that it is a bytes with the name, email and commit timestamp)

**committer**

*bytes*

The committer of the commit. (This is the raw content in the commit. This means that it is a bytes with the name, email and commit timestamp)

**first\_line**

*str*

The first line of the commit message.

**message**

*str*

The full commit message (including the first line).

**class** `maggit.git_objects.Tag(repo, sha)`

Bases: `maggit.git_objects.GitObject`

A tag object.

**object**

`GitObject`

The git object tagged by this tag.

**objectType**

`ObjectType`

The type of the git object tagged by this tag.

**tag**

*str*

The name of the tag.

**tager**

*bytes*

The person who create the tag. (This is the raw content in the tag. This means that it is a bytes with the name, email and tag timestamp)

**first\_line**

*str*

The first line of the tag message.

**message**

*str*

The full tag message (including the first line).

## 3.4 maggit.refs module

```
class maggit.refs.Ref(repo, sha)
    Bases: maggit.refs.BaseRef
```

**commit**

**repo**

**sha**

```
class maggit.refs.Branche(repo, branchName)
    Bases: maggit.refs.BaseRef
```

**commit**

**name**

**repo**

**sha**

```
class maggit.refs.Tag(repo, tagName)
    Bases: maggit.refs.BaseRef
```

**commit**

**name**

**object**

**repo**

**sha**

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`





## m

- `maggit.db`, [7](#)
- `maggit.db.db`, [7](#)
- `maggit.db.repo`, [9](#)
- `maggit.git_objects`, [16](#)
- `maggit.io`, [10](#)
- `maggit.io.loose`, [10](#)
- `maggit.io.pack`, [12](#)
- `maggit.io.packedref`, [12](#)
- `maggit.refs`, [18](#)
- `maggit.repo`, [15](#)



## A

author (maggit.git\_objects.Commit attribute), 17

## B

Blob (class in maggit.git\_objects), 16

blob\_content() (maggit.db.db.Gitdb method), 7

blob\_write() (maggit.db.db.Gitdb method), 7

Branche (class in maggit.refs), 18

branches (maggit.db.repo.Repo attribute), 9

branches (maggit.Repo attribute), 13

## C

check\_ref\_exists() (maggit.db.repo.Repo method), 9

check\_ref\_exists() (maggit.Repo method), 13

Commit (class in maggit.git\_objects), 17

Commit (class in maggit.repo), 15

commit (maggit.refs.Branche attribute), 18

commit (maggit.refs.Ref attribute), 18

commit (maggit.refs.Tag attribute), 18

commit\_content() (maggit.db.db.Gitdb method), 7

commit\_parse() (in module maggit.io.loose), 10

commit\_write() (maggit.db.db.Gitdb method), 8

committer (maggit.git\_objects.Commit attribute), 17

content (maggit.git\_objects.Blob attribute), 16

## E

entries (maggit.git\_objects.Tree attribute), 16

Entry (class in maggit.repo), 15

## F

first\_line (maggit.git\_objects.Commit attribute), 17

first\_line (maggit.git\_objects.Tag attribute), 17

## G

gen\_pack\_list() (maggit.db.db.Gitdb method), 8

get\_blob() (maggit.Repo method), 13

get\_commit() (maggit.Repo method), 13

get\_file() (maggit.repo.Commit method), 15

get\_first\_appearance() (maggit.repo.Commit method), 15

get\_first\_appearance() (maggit.repo.Entry method), 15

get\_first\_appearances() (maggit.repo.Commit method), 16

get\_full\_sha() (maggit.db.db.Gitdb method), 8

get\_full\_sha() (maggit.db.repo.Repo method), 9

get\_full\_sha() (maggit.Repo method), 13

get\_git\_dir() (maggit.db.repo.Repo class method), 9

get\_git\_dir() (maggit.Repo method), 14

get\_object() (maggit.Repo method), 14

get\_offset() (maggit.io.pack.GitPackIndex method), 12

get\_pack() (maggit.db.db.Gitdb method), 8

get\_peel\_ref\_sha() (maggit.db.repo.Repo method), 9

get\_peel\_ref\_sha() (maggit.Repo method), 14

get\_ref\_sha() (maggit.db.repo.Repo method), 9

get\_ref\_sha() (maggit.Repo method), 14

get\_tag() (maggit.Repo method), 14

get\_tree() (maggit.Repo method), 14

Gitdb (class in maggit.db.db), 7

GitObject (class in maggit.git\_objects), 16

GitPack (class in maggit.io.pack), 12

GitPackIndex (class in maggit.io.pack), 12

gitType (maggit.git\_objects.GitObject attribute), 16

## H

HEAD (maggit.db.repo.Repo attribute), 9

HEAD (maggit.Repo attribute), 13

hexbytes (maggit.Sha attribute), 13

hexstr (maggit.Sha attribute), 13

## I

init\_repo() (maggit.db.repo.Repo class method), 10

init\_repo() (maggit.Repo method), 14

## M

maggit.db (module), 7

maggit.db.db (module), 7

maggit.db.repo (module), 9

maggit.git\_objects (module), 16

maggit.io (module), 10

maggit.io.loose (module), 10

maggit.io.pack (module), 12

maggit.io.packedref (module), 12  
maggit.refs (module), 18  
maggit.repo (module), 15  
message (maggit.git\_objects.Commit attribute), 17  
message (maggit.git\_objects.Tag attribute), 18

## N

name (maggit.refs.Branche attribute), 18  
name (maggit.refs.Tag attribute), 18

## O

object (maggit.git\_objects.Tag attribute), 17  
object (maggit.refs.Tag attribute), 18  
object\_content() (in module maggit.io.loose), 10  
object\_content() (maggit.db.db.Gitdb method), 8  
object\_exists() (maggit.db.db.Gitdb method), 8  
object\_rawsha() (in module maggit.io.loose), 10  
object\_sha() (in module maggit.io.loose), 11  
object\_sha\_from\_raw() (in module maggit.io.loose), 11  
object\_type() (maggit.db.db.Gitdb method), 8  
object\_write() (in module maggit.io.loose), 11  
ObjectType (class in maggit), 15  
objectType (maggit.git\_objects.Tag attribute), 17

## P

packed\_ref\_parse() (in module maggit.io.packedref), 12  
parents (maggit.git\_objects.Commit attribute), 17  
parents (maggit.repo.Entry attribute), 15

## R

read\_object() (maggit.io.pack.GitPack method), 12  
Ref (class in maggit.refs), 18  
Repo (class in maggit), 13  
Repo (class in maggit.db.repo), 9  
repo (maggit.git\_objects.GitObject attribute), 16  
repo (maggit.refs.Branche attribute), 18  
repo (maggit.refs.Ref attribute), 18  
repo (maggit.refs.Tag attribute), 18

## S

Sha (class in maggit), 12  
sha (maggit.git\_objects.GitObject attribute), 16  
sha (maggit.refs.Branche attribute), 18  
sha (maggit.refs.Ref attribute), 18  
sha (maggit.refs.Tag attribute), 18

## T

Tag (class in maggit.git\_objects), 17  
Tag (class in maggit.refs), 18  
tag (maggit.git\_objects.Tag attribute), 17  
tag\_content() (maggit.db.db.Gitdb method), 8  
tag\_parse() (in module maggit.io.loose), 11  
tag\_write() (maggit.db.db.Gitdb method), 9

tagger (maggit.git\_objects.Tag attribute), 17  
tags (maggit.db.repo.Repo attribute), 10  
tags (maggit.Repo attribute), 15  
Tree (class in maggit.git\_objects), 16  
tree (maggit.git\_objects.Commit attribute), 17  
tree\_content() (maggit.db.db.Gitdb method), 9  
tree\_parse() (in module maggit.io.loose), 11  
tree\_write() (maggit.db.db.Gitdb method), 9