| 1 Example | 3 |
| 1.1 More Examples | 3 |
| 2 Modules | 13 |
| 2.1 API | 13 |
| 2.2 Authorization | 19 |
| 2.3 Events | 21 |
| 2.4 Gists | 22 |
| 2.5 Git | 25 |
| 2.6 GitHub | 30 |
| 2.7 Issue | 43 |
| 2.8 Legacy | 50 |
| 2.9 Models | 54 |
| 2.10 Organization | 57 |
| 2.11 Pull Request | 63 |
| 2.12 Repository | 69 |
| 2.13 Structures | 89 |
| 2.14 User | 90 |
| 2.15 Internals | 95 |
| 3 Installation | 101 |
| 3.1 Dependencies | 101 |
| 4 Contributing | 103 |
| 4.1 Contributor Friendly Work | 103 |
| 4.2 Running the Unit tests | 103 |
| 5 Contact | 105 |
| 6 History/Changelog | 107 |
| 6.1 0.6.0: 2013-xx-xx | 107 |
| 6.2 0.5.3: 2013-03-19 | 107 |
| 6.3 0.5.2: 2013-03-02 | 107 |
| 6.4 0.5.1: 2013-02-21 | 107 |
| 6.5 0.5: 2013-02-16 | 108 |
| 6.6 0.4: 2013-01-16 | 109 |
| 6.7 0.3: 2013-01-01 | 109 |
| 6.8 0.2: 2012-11-21 | 110 |
| 6.9 0.1: 2012-11-13 | 111 |
| 6.10 0.1b2: 2012-11-10 | 111 |
Python Module Index
GitHub3.py is a wrapper for the GitHub API written in Python. The design of GitHub3.py is centered around having a logical organization of the methods needed to interact with the API. Let me demonstrate this with a code example.
Let’s get information about a user:

```python
from github3 import login

gh = login('sigmavirus24', password='<password>')</gh>
sigmavirus24 = gh.user()
# <User [sigmavirus24:Ian Cordasco]>

print(sigmavirus24.name)
# Ian Cordasco
print(sigmavirus24.login)
# sigmavirus24
print(sigmavirus24.followers)
# 4

for f in gh.iter_followers():
    print(str(f))

kennethreitz = gh.user('kennethreitz')
# <User [kennethreitz:Kenneth Reitz]>

print(kennethreitz.name)
print(kennethreitz.login)
print(kennethreitz.login)
print(kennethreitz.followers)

followers = [str(f) for f in gh.iter_followers('kennethreitz')]```

### 1.1 More Examples

#### 1.1.1 Using Tokens for Your Projects

Let’s say you’re designing an application that uses github3.py. If your intention is to have users authenticate, you have a few options.

1. Ask the user to enter their credentials each time they start the application. (Or save the username somewhere, and just ask for the password.)
2. Ask the user to supply their credentials once and store them somewhere for later use. (VERY VERY BAD)
3. Ask the user to supply their credentials once, get an authorization token and store that for later use.
The first isn’t a bad method at all, it just unfortunately may lead to unhappy users, this should always be an option though. The second (as I already noted) is a bad idea. Even if you obfuscate the username and password, they can still be discovered and no level of obfuscation is clever enough. (May I also take this moment to remind people that base64 is \textit{not} encryption.) The last is probably the least objectionable of the evils. The token has scopes so there is only so much someone can do with it and it works well with github3.py.

\section*{Requesting a token}

If you’re not doing a web application, you are more than welcome to use github3.py (otherwise work with \texttt{redirects}). Let’s say your application needs access to public and private repositories, and the users but not to gists. Your \texttt{scopes} should be \texttt{['user', 'repo']}. I’m also assuming your application will not be deleting any repositories. The only things left to do are collect the username and password and give a good description for your application.

\begin{verbatim}
from github3 import authorize
from getpass import getuser, getpass

user = getuser()
password = ''
while not password:
    password = getpass('Password for {0}: '.format(user))

note = 'github3.py example app'
note_url = 'http://example.com'
scopes = ['user', 'repo']

auth = authorize(user, password, scopes, note, note_url)

with open(CREDENTIALS_FILE, 'w') as fd:
    fd.write(auth.token + '
    fd.write(auth.id)

In the future, you can then read that token in without having to bother your user. If at some later point in the lifetime of your application you need more privileges, you simply do the following:

from github3 import login

id = ''
with open(CREDENTIALS_FILE, 'r') as fd:
    token = fd.readline().strip()  # Can't hurt to be paranoid
    id = fd.readline().strip()

gh = login(token=token)
auth = gh.authorization(id)
auth.update(add_scopes=['repo:status', 'gist'], rm_scopes=['user'])

# if you want to be really paranoid, you can then test:
# token == auth.token
# in case the update changes the token

Hopefully this helps someone.
\end{verbatim}

\subsection*{1.1.2 Gist Code Examples}

Examples with \texttt{Gists}
Listing gists after authenticating

```python
from github3 import login
gh = login(username, password)
gists = [g for g in gh.iter_gists()]
```

Creating a gist after authenticating

```python
from github3 import login
gh = login(username, password)
files = {
    'spam.txt' : {
        'content': 'What... is the air-speed velocity of an unladen swallow?'
    }
}
gist = gh.create_gist('Answer this to cross the bridge', files, public=False)
# gist == <Gist [gist-id]>
print(gist.html_url)
```

Creating an anonymous gist

```python
from github3 import create_gist
files = {
    'spam.txt' : {
        'content': 'What... is the air-speed velocity of an unladen swallow?'
    }
}
gist = create_gist('Answer this to cross the bridge', files, public=False)
comments = [c for c in gist.iter_comments()]
# []
comment = gist.create_comment('Bogus. This will not work."
# Which of course it didn’t, because you’re not logged in
# comment == None
print(gist.html_url)
```

1.1.3 Git Code Examples

The GitHub API does not just provide an API to interact with GitHub’s features. A whole section of the API provides a RESTful API to git operations that one might normally perform at the command-line or via your git client.

Creating a Blob Object

One of the really cool (and under used, it seems) parts of the GitHub API involves the ability to create commit and blob objects.

```python
from github3 import login
g = login(username, password)
repo = g.repository('sigmavirus24', 'Todo.txt-python')
sha = repo.create_blob('Testing blob creation', 'utf-8')
```
sha
# u'57fad9a39b27e5eb4700f66673ce860b65b93ab8'
blob = repo.blob(sha)
blob.content
# u'VEGzdGlzMyBibG9iIGNyZWF0aW9u

blob.decoded
# u'Testing blob creation'
blob.encoding
# u'base64'

Creating a Tag Object

GitHub provides tar files for download via tag objects. You can create one via `git tag` or you can use the API.

```
from github3 import login

g = login(username, password)
repo = g.repository('sigmavirus24', 'github3.py')
tag = repo.tag('cdba84b4fede2c69cb1ee246b33f49f19475abfa')
tag
# <Tag [cdba84b4fede2c69cb1ee246b33f49f19475abfa]>
tag.object.sha
# u'24ea44d302c6394a0372dcde8fd8aed899c0034b'
tag.object.type
# u'commit'
```

1.1.4 GitHub Examples

Examples using the GitHub object.

Assumptions

I’ll just make some basic assumptions for the examples on this page. First, let’s assume that all you ever import from github3.py is `login` and GitHub and that you have already received your GitHub object `g`. That might look like this:

```
from github3 import login, GitHub
from getpass import getpass, getuser
import sys

try:
    import readline
except ImportError:
    pass

try:
    user = raw_input('GitHub username: ')
except KeyboardInterrupt:
    user = getuser()

password = getpass('GitHub password for {0}: '.format(user))

# Obviously you could also prompt for an OAuth token
if not (user and password):
    print("Cowardly refusing to login without a username and password.")
    sys.exit(1)
```
g = login(user, password)

So anywhere you see g used, you can safely assume that it is an instance where a user has authenticated already.

For the cases where we do not need an authenticated user, or where we are trying to demonstrate the differences between the two, I will use anon. anon could be instantiated like so:

anon = GitHub()

Also let’s define the following constants:

\[
\begin{align*}
\sigma &= 'sigmavirus24' \\
github3 &= 'github3.py'
\end{align*}
\]

todopy &= 'Todo.txt-python'
kr &= 'kennethreitz'
requests &= 'requests'

We may not need all of them, but they’ll be useful

**Adding a new key to your account**

```python
try:
    path = raw_input('Path to key: ')
except KeyboardInterrupt:
    path = ''

try:
    name = raw_input('Key name: ')
except KeyboardInterrupt:
    name = ''

if not (path and name):  # Equivalent to not path or not name
    print("Cannot create a new key without a path or name")
    sys.exit(1)

with open(path, 'r') as key_file:
    key = g.create_key(name, key_file)
    if key:
        print('Key {0} created.'.format(key.title))
    else:
        print('Key addition failed.')
```

**Deleting the key we just created**

Assuming we still have key from the previous example:

```python
if g.delete_key(key.id):
    print("Successfully deleted key {0}".format(key.id))
```

There would actually be an easier way of doing this, however, if we do have the key object that we created:

```python
if key.delete():
    print("Successfully deleted key {0}".format(key.id))
```
Creating a new repository

repo = {}
keys = ['name', 'description', 'homepage', 'private', 'has_issues', 'has_wiki', 'has_downloads']

for key in keys:
    try:
        repo[key] = raw_input(key + ': ')
    except KeyboardInterrupt:
        pass

r = None
if repo.get('name'):
    r = g.create_repo(repo.pop('name'), **repo)

if r:
    print("Created {0} successfully.").format(r.name)

Follow another user on GitHub

I’m cheating here and using most of the follow functions in one example

if not g.is_following(sigma):
    g.follow(sigma)

if not g.is_subscribed(sigma, github3py):
    g.subscribe(sigma, github3py)

if g.is_subscribed(sigma, todopy):
    g.unsubscribe(sigma, todopy)

for follower in g.iter_followers():
    print("{0} is following me.".format(follower.login))

for followee in g.iter_following():
    print("I am following {0}".format(followee.login))

if g.is_following(sigma):
    g.unfollow(sigma)

Changing your user information

Note that you can not change your login name via the API.

new_name = ‘J. Smith’
blog = ‘http://www.example.com/’
company = ‘Vandelay Industries’
bio = """J. Smith
A simple man working at a latex factory"
""

if g.update_user(new_name, blog, company, bio=bio):
    print(‘Profile updated.’)
This is the same as:

```python
me = g.user()
if me.update(new_name, blog, company, bio=bio):
    print('Profile updated.')
```

### 1.1.5 Issue Code Examples

Examples using Issues

#### Administrating Issues

Let’s assume you have your username and password stored in `user` and `pw` respectively, you have your repository name stored in `repo`, and the number of the issue you’re concerned with in `num`.

```python
from github3 import login
gh = login(user, pw)
issue = gh.issue(user, repo, num)
if issue.is_closed():
    issue.reopen()

issue.edit('New issue title', issue.body + '

**Update:** Text to append')

# Assuming issue is the same as above ...
issue.create_comment('This should be fixed in 6d4oe5. Closing as fixed.')
issue.close()
```

The following shows how you could use github3.py to fetch and display your issues in your own style and in your webbrowser.

Or how to do the same by wrapping the lines in your terminal.

### 1.1.6 Taking Advantage of GitHubIterator

Let’s say that for some reason you’re stalking all of GitHub’s users and you just so happen to be using github3.py to do this. You might write code that looks like this:

```python
import github3

g = github3.login(USERNAME, PASSWORD)

for u in g.iter_all_users():
    add_user_to_database(u)
```

The problem is that you will then have to reiterate over all of the users each time you want to get the new users. You have two approaches you can take to avoid this with `GitHubIterator`.

You can not call the method directly in the for-loop and keep the iterator as a separate reference like so:

```python
i = g.iter_all_users():
for u in i:
    add_user_to_database(u)
```
The First Approach

Then after your first pass through your GitHubIterator object will have an attribute named etag. After you’ve added all the currently existing users you could do the following to retrieve the new users in a timely fashion:

```python
import time

while True:
    i.refresh(True)
    for u in i:
        add_user_to_database(u)
    time.sleep(120)  # Sleep for 2 minutes
```

The Second Approach

```python
etag = i.etag
# Store this somewhere

# Later when you start a new process or go to check for new users you can
# then do
i = g.iter_all_users(etag=etag)

for u in i:
    add_user_to_database(u)
```

If there are no new users, these approaches won’t impact your ratelimit at all. This mimics the ability to conditionally refresh data on almost all other objects in github3.py.

1.1.7 Using Logging with github3.py

New in version 0.6.0. The following example shows how to set up logging for github3.py. It is off by default in the library and will not pollute your logs.

One thing to note is that if you want more detailed information about what is happening while the requests are sent, you can do the following:

```python
import logging
urllib3 = logging.getLogger('requests.packages.urllib3')

And configure the logger for urllib3. Unfortunately, requests itself doesn’t provide any logging, so the best you can actually get is by configuring urllib3.

You will see messages about the following in the logs:

- Construction of URLs used in requests, usually in the form: ('https://api.github.com', 'repos', 'sigmavirus24', 'github3.py')
- What request is being sent, e.g., POST https://api.github.com/user kwargs={}
- If JSON is trying to be extracted from the response, what the response’s status code was, what the expected status code was and whether any JSON was actually returned.
### A Conversation With Octocat

What you should see

Hey Octocat!

```
MM..  ,MM  
MMMMMMMMMMMMMMMM  
MMMMMMMMMMMMMMMM  |   
MMMMMMMMMMMMMMMM  | Hey Ian! | 
MMMMMMMMMMMMMMMM  |   _____| 
MMMM::: -::::: - -::MM   |
MM~:~ ~:::~ ~:~MM   / 
.. MMMM:::. :::+:::. ::MMMMM ..
.MM::: :_, :::::MM, 
MM;:::;MM 
-MM MMMMMM
^ M+ MMMMMMMMM
```

What do you think about github3.py?

```
MM..  ,MM  
MMMMMMMMMMMMMMMM  
MMMMMMMMMMMMMMMM  | [github3.py rocks!] | 
MMMMMMMMMMMMMMMM  |   _____| 
MMMM::: -::::: - -::MM   |
MM~:~ ~:::~ ~:~MM   / 
.. MMMM:::. :::+:::. ::MMMMM ..
.MM::: :_, :::::MM, 
MM;:::;MM 
-MM MMMMMM
^ M+ MMMMMMMMM
```

Thanks Octocat, that means a lot coming from you.

FIN.

Epilog:

The preceding conversation was entirely fictional. If you didn’t realize that, you need to get out more. And yes, I did just have a conversation with an API. Cool, no? (Sad too, I guess.)

---

1.1. More Examples
CHAPTER TWO

MODULES

2.1 API

This part of the documentation covers the API. This is intended to be a beautifully written module which allows the user (developer) to interact with github3.py elegantly and easily.

2.1.1 Module Contents

To interact with the GitHub API you can either authenticate to access protected functionality or you can interact with it anonymously. Authenticating provides more functionality to the user (developer).

To authenticate, you simply use `github3.login()`.

```
github3.login(username=None, password=None, token=None)
```

Constructs and returns a GitHub session with the username and password, or token

**Parameters**

- `username` (*str*) – login name
- `password` (*str*) – password for the login
- `token` (*str*) – OAuth token

**Returns** GitHub

With the GitHub object that is returned you have access to more functionality. See that object’s documentation for more information.

To use the API anonymously, you can create a new GitHub object, e.g.,

```
from github3 import GitHub

gh = GitHub()
```

Or you can simply use the following functions

```
github3.authorize(login, password, scopes, note='', note_url='', client_id='', client_secret='')
```

Obtain an authorization token from the GitHub API for the GitHub API.

**Parameters**

- `login` (*str*) – (required)
- `password` (*str*) – (required)
github3.py Documentation, Release 0.6.0.b

- **scopes** (`list`) – (required), areas you want this token to apply to, i.e., ‘gist’, ‘user’
- **note** (`str`) – (optional), note about the authorization
- **note_url** (`str`) – (optional), url for the application
- **client_id** (`str`) – (optional), 20 character OAuth client key for which to create a token
- **client_secret** (`str`) – (optional), 40 character OAuth client secret for which to create the token

**Returns** Authorization

github3.create_gist (description, files)
Creates an anonymous public gist.

**Parameters**
- **description** (`str`) – (required), short description of the gist
- **files** (`dict`) – (required), file names with associated dictionaries for content, e.g. {'spam.txt': {'content': 'File contents ...'}}

**Returns** Gist

github3.gist (id_num)
Get the gist identified by id_num.

**Parameters** id_num (`int`) – (required), unique id of the gist

**Returns** Gist

github3.gitignore_template (language)
Returns the template for language.

**Returns** str

github3.gitignore_templates ()
Returns the list of available templates.

**Returns** list of template names

github3.issue (owner, repository, number)
Anonymously gets issue :number on :owner/:repository.

**Parameters**
- **owner** (`str`) – (required), repository owner
- **repository** (`str`) – (required), repository name
- **number** (`int`) – (required), issue number

**Returns** Issue

github3.iter_all_repos (number=-1, etag=None)
Iterate over every repository in the order they were created.
Parameters

• number \((\text{int})\) – (optional), number of repositories to return. Default: -1, returns all of them

• etag \((\text{str})\) – (optional), ETag from a previous request to the same endpoint

Returns  generator of Repository

github3.iter_all_users\( (\text{number}=-1, \text{etag}=\text{None})\)
Iterate over every user in the order they signed up for GitHub.

Parameters

• number \((\text{int})\) – (optional), number of users to return. Default: -1, returns all of them

• etag \((\text{str})\) – (optional), ETag from a previous request to the same endpoint

Returns  generator of User

github3.iter_events\( (\text{number}=-1, \text{etag}=\text{None})\)
Iterate over public events.

Parameters

• number \((\text{int})\) – (optional), number of events to return. Default: -1 returns all available events

• etag \((\text{str})\) – (optional), ETag from a previous request to the same endpoint

Returns  generator of Events

github3.iter_followers\( (\text{username}, \text{number}=-1, \text{etag}=\text{None})\)
List the followers of username.

Parameters

• username \((\text{str})\) – (required), login of the person to list the followers of

• number \((\text{int})\) – (optional), number of followers to return, Default: -1, return all of them

• etag \((\text{str})\) – (optional), ETag from a previous request to the same endpoint

Returns  generator of User

github3.iter_following\( (\text{username}, \text{number}=-1, \text{etag}=\text{None})\)
List the people username follows.

Parameters

• username \((\text{str})\) – (required), login of the user

• number \((\text{int})\) – (optional), number of users being followed by username to return. Default: -1, return all of them

• etag \((\text{str})\) – (optional), ETag from a previous request to the same endpoint

Returns  generator of User
github3.iter_gists(username=None, number=-1, etag=None)

Get public gists or gists for the provided username.

**Parameters**

- **username** *(str)* – (optional), if provided, get the gists for this user instead of the authenticated user.
- **number** *(int)* – (optional), number of gists to return. Default: -1, return all of them
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Gists**

github3.iter_orgs(username, number=-1, etag=None)

List the organizations associated with **username**.

**Parameters**

- **username** *(str)* – (required), login of the user
- **number** *(int)* – (optional), number of orgs to return. Default: -1, return all of the issues
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Organizations**

github3.iter_repos(login='', type='', sort='', direction='', number=-1, etag=None)

List public repositories for the specified **login** or all repositories for the authenticated user if **login** is not provided.

**Parameters**

- **login** *(str)* – (required)
- **type** *(str)* – (optional), accepted values: (‘all’, ‘owner’, ‘public’, ‘private’, ‘member’) API default: ‘all’
- **sort** *(str)* – (optional), accepted values: (‘created’, ‘updated’, ‘pushed’, ‘full_name’) API default: ‘created’
- **direction** *(str)* – (optional), accepted values: (‘asc’, ‘desc’), API default: ‘asc’ when using ‘full_name’, ‘desc’ otherwise
- **number** *(int)* – (optional), number of repositories to return. Default: -1 returns all repositories
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Repository** objects

github3.iter_repo_issues(owner, repository, milestone=None, state=None, assignee=None, mentioned=None, labels=None, sort=None, direction=None, since=None, number=-1, etag=None)

List issues on owner/repository. Only owner and repository are required.

**Parameters**

- **owner** *(str)* – login of the owner of the repository
- **repository** *(str)* – name of the repository
github3.py Documentation, Release 0.6.0.b

- **milestone** *(int)* – None, ‘*’, or ID of milestone
- **state** *(str)* – accepted values: (‘open’, ‘closed’) api-default: ‘open’
- **assignee** *(str)* – ‘*’ or login of the user
- **mentioned** *(str)* – login of the user
- **labels** *(str)* – comma-separated list of label names, e.g., ‘bug.ui,@high’
- **sort** *(str)* – accepted values: (‘created’, ‘updated’, ‘comments’) api-default: created
- **direction** *(str)* – accepted values: (‘asc’, ‘desc’) api-default: desc
- **since** *(str)* – ISO 8601 formatted timestamp, e.g., 2012-05-20T23:10:27Z
- **number** *(int)* – (optional), number of issues to return. Default: -1 returns all issues
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of `Issues`

---

github3.iter_starred *(username, number=-1, etag=None)*

Iterate over repositories starred by `username`.

**Parameters**
- **username** *(str)* – (optional), name of user whose stars you want to see
- **number** *(int)* – (optional), number of repositories to return. Default: -1 returns all repositories
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of `Repository`

---

github3.iter_subscriptions *(username, number=-1, etag=None)*

Iterate over repositories subscribed to by `username`.

**Parameters**
- **username** *(str)* – (optional), name of user whose subscriptions you want to see
- **number** *(int)* – (optional), number of repositories to return. Default: -1 returns all repositories
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of `Repository`

---

github3.markdown *(text, mode='', context='', raw=False)*

Render an arbitrary markdown document.

**Parameters**
- **text** *(str)* – (required), the text of the document to render
- **mode** *(str)* – (optional), ‘markdown’ or ‘gfm’
- **context** *(str)* – (optional), only important when using mode ‘gfm’, this is the repository to use as the context for the rendering
- **raw** *(bool)* – (optional), renders a document like a README.md, no gfm, no context

---

2.1. API
**Returns**  str – HTML formatted text

### github3.octocat (say=None)

Returns an easter egg from the API.

**Params**
- `str say` (optional), pass in what you’d like Octocat to say

**Returns**  ascii art of Octocat

### github3.organization (login)

See `organization`.

### github3.pull_request (owner, repository, number)

Anonymously retrieve pull request :number on :owner/:repository

**Parameters**
- `owner` *(str)* – (required), repository owner
- `repository` *(str)* – (required), repository name
- `number` *(int)* – (required), pull request number

**Returns**  PullRequest

### github3.ratelimit_remaining ()

Get the remaining number of requests allowed.

### github3.repository (owner, repository)

See `repository`.

### github3.search_issues (owner, repo, state, keyword)

Find issues by state and keyword.

**Parameters**
- `owner` *(str)* – (required)
- `repo` *(str)* – (required)
- `state` *(str)* – (required), accepted values: ('open', 'closed')
- `keyword` *(str)* – (required), what to search for
- `start_page` *(int)* – (optional), page to get (results come 100/page)

**Returns**  list of LegacyIssues

### github3.search_repos (keyword, **params)

Search all repositories by keyword.

**Parameters**
- `keyword` *(str)* – (required)
• **language** (*str*) – (optional), language to filter by
• **start_page** (*int*) – (optional), page to get (results come 100/page)

Returns list of LegacyRepos

```python
github3.search_users(keyword)
Search all users by keyword.

Parameters
• **keyword** (*str*) – (required)
• **start_page** (*int*) – (optional), page to get (results come 100/page)

Returns list of LegacyUsers
```

```python
github3.search_email(email)
Search users by email.

Parameters email (*str*) – (required)

Returns LegacyUser
```

```python
github3.user(login)
See user.
```

```python
github3.zen()
Returns a quote from the Zen of GitHub. Yet another API Easter Egg
```

### 2.1.2 Enterprise Use

If you’re using github3.py to interact with an enterprise installation of GitHub, you must use the `GitHubEnterprise` object. Upon initialization, the only parameter you must supply is the URL of your enterprise installation, e.g.

```python
from github import GitHubEnterprise

g = GitHubEnterprise('https://github.examplesintl.com')
stats = g.admin_stats('all')
assert 'issues' in stats, ('Key issues is not included in the admin' 'statistics')
```

### 2.2 Authorization

This part of the documentation covers the `Authorization` object.

```python
class github3.auths.Authorization(auth, session=None)
The Authorization object.
```
app = None
Details about the application (name, url)

created_at = None
datetime object representing when the authorization was created.

delete()
delete this authorization

classmethod from_json(json)
Return an instance of cls formed from json.

id = None
Unique id of the authorization

name = None
App name

note = None
Note about the authorization

note_url = None
URL about the note

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header ('Last-Modified', or 'ETag') on the object and send that as described in the Conditional Requests section of the docs

Returns self
The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]  
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

scopes = None
List of scopes this applies to

to_json()  
Return the json representing this object.

token = None
Returns the Authorization token

update(scopes=[], add_scopes=[], rm_scopes=[], note='', note_url='')
Update this authorization.

Parameters
• scopes (list) – (optional), replaces the authorization scopes with these
• add_scopes (list) – (optional), scopes to be added
• **rm_scopes** *(list)* – (optional), scopes to be removed
• **note** *(str)* – (optional), new note about authorization
• **note_url** *(str)* – (optional), new note URL about this authorization

Returns `bool`

```python
def updated_at = None
    datetime object representing when the authorization was created.
```

## 2.3 Events

This part of the documentation covers the `Event` object.

### 2.3.1 Event Objects

```python
class github3.events.Event(event)
    The `Event` object. It structures and handles the data returned by via the `Events` section of the GitHub API.

actor = None
    `User` object representing the actor.

created_at = None
    datetime object representing when the event was created.

classmethod from_json(json)
    Return an instance of cls formed from json.

id = None
    Unique id of the event

is_public()
    Indicates whether the Event is public or not.

    Warning: This will be deprecated in 0.6

Returns `bool` – True if event is public, False otherwise

static list_types()
    List available payload types

org = None
    List all possible types of Events

payload = None
    Dictionary with the payload. Payload structure is defined by type.

public = None
    Indicates whether the Event is public or not.

repo = None
    Return tuple(owner, repository_name)

to_json()
    Return the json representing this object.

type = None
    Event type
When accessing the payload of the event, you should notice that you receive a dictionary where the keys depend on the event type. Note:

- where they reference an array in the documentation but index it like a dictionary, you are given a regular dictionary
- where they reference a key as returning an object, you receive the equivalent object from the dictionary, e.g., for a Fork Event:

```python
>>> event
<Event [Fork]>
>>> event.payload
{u'forkee': <Repository [eweap/redactor-js]>
>>> event.payload['forkee']
<Repository [eweap/redactor-js]>
```

Using the dictionary returned as the payload makes far more sense than creating an object for the payload in this instance. For one, creating a class for each payload type would be insanity. I did it once, but it isn’t worth the effort. Having individual handlers as we have now which modify the payload to use our objects when available is more sensible.

## 2.4 Gists

This part of the documentation details the properties and methods associated with `Gist`, `GistComment`, and `GistFile` objects. These classes should never be instantiated by the user (developer) directly.

### 2.4.1 Gist Objects

```python
class github3.gists.Gist(data, session=None)
The Gist object. This object holds all the information returned by Github about a gist. With it you can comment on or fork the gist (assuming you are authenticated), edit or delete the gist (assuming you own it). You can also “star” or “unstar” the gist (again assuming you have authenticated).

comments = None
Number of comments on this gist

create_comment(body)
Create a comment on this gist.

Parameters

- **body** `(str)` – (required), body of the comment

Returns `GistComment`

created_at = None
datetime object representing when the gist was created.

delete()
Delete this gist.

Returns `bool` – whether the deletion was successful

description = None
Description of the gist

edit(description=' ', files={})
Edit this gist.

Parameters
```
- **description** *(str)* – (optional), description of the gist

- **files** *(dict)* – (optional), files that make up this gist; the key(s) should be the file name(s) and the values should be another (optional) dictionary with (optional) keys: ‘content’ and ‘filename’ where the former is the content of the file and the latter is the new name of the file.

**Returns**  bool – whether the edit was successful

**files** = None
Number of files in this gist.

**fork** ()
Fork this gist.

**Returns**  **Gist** if successful, **None** otherwise

**forks** = None
The number of forks of this gist.

**classmethod** **from_json** *(json)*
Return an instance of **cls** formed from **json**.

**git_pull_url** = None
Git URL to pull this gist, e.g., git://gist.github.com/1.git

**git_push_url** = None
Git URL to push to gist, e.g., git@gist.github.com/1.git

**history** = None
History of this gist, list of **GistHistory**

**html_url** = None
URL of this gist at Github, e.g., https://gist.github.com/1

**id** = None
Unique id for this gist.

**is_public** ()
Checks to see if this gist is public or not.

**Returns**  bool – True if public, False if private

**is_starred** ()
Checks to see if this gist is starred by the authenticated user.

**Returns**  bool – True if it is starred, False otherwise

**iter_comments** *(number=-1, etag=None)*
List comments on this gist.

**Parameters**

- **number** *(int)* – (optional), number of comments to iterate over. Default: -1 will iterate over all comments on the gist

- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns**  generator of **GistComment**s

**iter_commits** *(number=-1)*
Iter over the commits on this gist.

These commits will be requested from the API and should be the same as what is in **Gist.history**. New in version 0.6.
Parameters number (int) – (optional), number of commits to iterate over. Default: -1 will iterate over all commits associated with this gist.

Returns generator of :class:`GistHistory <github3.gists.history.GistHistory>`s

iter_files() List of :class:`GistFile` objects representing the files stored in this gist.

iter_forks() List of :class:`Gist` objects representing forks of this gist.

owner = None :class:`User` object representing the owner of this gist.

public = None Boolean describing if the gist is public or private

ratelimit_remaining Number of requests before GitHub imposes a ratelimit.

refresh (conditional=False) Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for r in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

star() Star this gist.

Returns bool – True if successful, False otherwise

to_json() Return the json representing this object.

unstar() Un-star this gist.

Returns bool – True if successful, False otherwise

updated_at = None datetime object representing the last time this gist was updated.
2.5 Git

This part of the documentation covers the module associated with the Git Data section of the GitHub API.

- Blob
- Commit
- GitData
- GitObject
- Hash
- Reference
- Tag
- Tree

2.5.1 Git Objects

class github3.git.Blob(blob)
The Blob object.

    content = None
    Raw content of the blob.

    decoded = None
    Decoded content of the blob.

    encoding = None
    Encoding of the raw content.

    @classmethod
    from_json(json)
    Return an instance of cls formed from json.

    sha = None
    SHA1 of the blob

    size = None
    Size of the blob in bytes

    to_json()
    Return the json representing this object.

class github3.git.Commit(commit, session=None)
The Commit object. This represents a commit made in a repository.

    author = None
    dict containing at least the name, email and date the commit was

    author_as_User()
    Attempt to return the author attribute as a User. No guarantees are made about the validity of this object, i.e., having a login or created_at object.

    committer = None
    dict containing similar information to the author attribute
committer_as_User()
   Attempt to return the committer attribute as a User object. No guarantees are made about the validity of this object.

classmethod from_json(json)
   Return an instance of cls formed from json.

ratelimit_remaining
   Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
   Re-retrieve the information for this object and returns the refreshed instance.

   Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

   Returns self

   The reasoning for the return value is the following example:

   ```python
gerbs = [r.refresh() for r in g.iter_repos('kennethreitz')]
```

Without the return value, that would be an array of None’s and you would otherwise have to do:

```python
repos = [r for i in g.iter_repos('kennethreitz')]
[.refresh() for r in repos]
```

Which is really an anti-pattern. Changed in version 0.5.

to_json()
   Return the json representing this object.

tree = None
   Tree the commit belongs to.

class github3.git.GitData(data, session=None)
   The GitData object. This isn’t directly returned to the user (developer) ever. This is used to prevent duplication of some common items among other Git Data objects.

classmethod from_json(json)
   Return an instance of cls formed from json.

ratelimit_remaining
   Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
   Re-retrieve the information for this object and returns the refreshed instance.

   Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

   Returns self

   The reasoning for the return value is the following example:

   ```python
gerbs = [r.refresh() for r in g.iter_repos('kennethreitz')]
```

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')] [r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

sha = None
SHA of the object
to_json()
Return the json representing this object.

class github3.git.GitObject(obj)
The GitObject object.
classmethod from_json(json)
Return an instance of cls formed from json.
ratelimt_remaining
Number of requests before GitHub imposes a ratelimit.
refresh(conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self
The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')] [r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.
to_json()
Return the json representing this object.
type = None
The type of object.

class github3.git.Hash(info)
The Hash object.
classmethod from_json(json)
Return an instance of cls formed from json.
mode = None
File mode
path = None
Path to file
sha = None
SHA of the hash
\[ size = None \]
Size of hash

\[ to\_json() \]
Return the json representing this object.

\[ type = None \]
Type of hash, e.g., blob

\[ url = None \]
URL of this object in the GitHub API

\[ class \ github3.git.\texttt{Reference}(ref, session=None) \]
The \texttt{Reference} object. This represents a reference created on a repository.

\[ delete() \]
Delete this reference.

\[ \text{Returns} \] bool

\[ \text{classmethod} \ from\_json(json) \]
Return an instance of \texttt{cls} formed from \texttt{json}.

\[ object = None \]
\texttt{GitObject} the reference points to

\[ ratelimit\_remaining \]
Number of requests before GitHub imposes a ratelimit.

\[ ref = None \]
The reference path, e.g., refs/heads/sc/featureA

\[ refresh(conditional=False) \]
Re-retrieve the information for this object and returns the refreshed instance.

\[ \text{Parameters} \] \texttt{conditional} (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

\[ \text{Returns} \] self

The reasoning for the return value is the following example:

\begin{verbatim}
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]
\end{verbatim}

Without the return value, that would be an array of None’s and you would otherwise have to do:

\begin{verbatim}
repos = [for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]
\end{verbatim}

Which is really an anti-pattern. Changed in version 0.5.

\[ to\_json() \]
Return the json representing this object.

\[ update(sha, force=False) \]
Update this reference.

\[ \text{Parameters} \]
\begin{itemize}
  \item \texttt{sha} (str) – (required), sha of the reference
  \item \texttt{force} (bool) – (optional), force the update or not
\end{itemize}
class github3.git.Tag(tag)

    classmethod from_json(json)
        Return an instance of cls formed from json.

    message = None
        Commit message for the tag

    object = None
        GitObject for the tag

    ratelimit_remaining
        Number of requests before GitHub imposes a ratelimit.

    refresh(conditional=False)
        Re-retrieve the information for this object and returns the refreshed instance.

        Parameters conditional (bool) – If True, then we will search for a stored header ('Last-
        Modified', or 'ETag') on the object and send that as described in the Conditional Requests
        section of the docs

        Returns self

        The reasoning for the return value is the following example:

        repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

        Without the return value, that would be an array of None's and you would otherwise have to do:

        repos = [r for i in g.iter_repos('kennethreitz')]  
        [r.refresh() for r in repos]

        Which is really an anti-pattern. Changed in version 0.5.

    tag = None
        String of the tag

    tagger = None
        dict containing the name and email of the person

    to_json()
        Return the json representing this object.

class github3.git.Tree(tree, session=None)

    The Tree object.

    classmethod from_json(json)
        Return an instance of cls formed from json.

    ratelimit_remaining
        Number of requests before GitHub imposes a ratelimit.

    recurse()
        Recurse into the tree.

        Returns Tree
refresh(conditional=False)

Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos(‘kennethreitz’)]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos(‘kennethreitz’)]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

to_json()

Return the json representing this object.

tree = None

list of Hash objects

2.6 GitHub

This part of the documentation covers the GitHub object. A large portion of what you will likely want to do can be found in this class. If you’re looking for anonymous functions, you’re most likely looking for the API.

2.6.1 Examples

Examples utilizing this object can be found here.

2.6.2 GitHub Object

class github3.github.GitHub (login='', password='', token='')

Stores all the session information.

There are two ways to log into the GitHub API

from github3 import login

or

from github3 import GitHub

This is simple backward compatibility since originally there was no way to call the GitHub object with authentication parameters.
authorization(id_num)
Get information about authorization id.

Parameters id_num (int) – (required), unique id of the authorization

Returns Authorization

authorize(login, password, scopes, note='', note_url='', client_id='', client_secret='')
Obtain an authorization token from the GitHub API for the GitHub API.

Parameters

• login (str) – (required)
• password (str) – (required)
• scopes (list) – (required), areas you want this token to apply to, i.e., ‘gist’, ‘user’
• note (str) – (optional), note about the authorization
• note_url (str) – (optional), url for the application
• client_id (str) – (optional), 20 character OAuth client key for which to create a token
• client_secret (str) – (optional), 40 character OAuth client secret for which to create the token

Returns Authorization

create_gist(description, files, public=True)
Create a new gist.

If no login was provided, it will be anonymous.

Parameters

• description (str) – (required), description of gist
• files (dict) – (required), file names with associated dictionaries for content, e.g. 
  {'spam.txt': {'content': 'File contents ...'}}
• public (bool) – (optional), make the gist public if True

Returns Gist

create_issue(owner, repository, title, body=None, assignee=None, milestone=None, labels=[]) 
Create an issue on the project ‘repository’ owned by ‘owner’ with title ‘title’.

body, assignee, milestone, labels are all optional.

Parameters

• owner (str) – (required), login of the owner
• repository (str) – (required), repository name
• title (str) – (required), Title of issue to be created
• body (str) – (optional), The text of the issue, markdown formatted
• assignee (str) – (optional), Login of person to assign the issue to
• milestone (str) – (optional), Which milestone to assign the issue to
• labels (list) – (optional), List of label names.

Returns Issue
create_key (title, key)

Create a new key for the authenticated user.

Parameters

- title (str) – (required), key title
- key – (required), actual key contents, accepts path as a string or file-like object

Returns Key

create_repo (name, description='', homepage='', private=False, has_issues=True, has_wiki=True, has_downloads=True, auto_init=False, gitignore_template='')

Create a repository for the authenticated user.

Parameters

- name (str) – (required), name of the repository
- description (str) – (optional)
- homepage (str) – (optional)
- private (str) – (optional), If True, create a private repository. API default: False
- has_issues (bool) – (optional), If True, enable issues for this repository. API default: True
- has_wiki (bool) – (optional), If True, enable the wiki for this repository. API default: True
- has_downloads (bool) – (optional), If True, enable downloads for this repository. API default: True
- auto_init (bool) – (optional), auto initialize the repository
- gitignore_template (str) – (optional), name of the git template to use; ignored if auto_init = False.

Returns Repository

delete_key (key_id)

Delete user key pointed to by key_id.

Parameters key_id (int) – (required), unique id used by Github

Returns bool

dfollow (login)

Make the authenticated user follow login.

Parameters login (str) – (required), user to follow

Returns bool

classmethod from_json (json)

Return an instance of cls formed from json.

gist (id_num)

Gets the gist using the specified id number.

Parameters id_num (int) – (required), unique id of the gist

Returns Gist

gitignore_template (language)

Returns the template for language.
Returns  str
gitignore_templates()
    Returns the list of available templates.
    Returns  list of template names
is_following(login)
    Check if the authenticated user is following login.
    Parameters  login (str) – (required), login of the user to check if the authenticated user is checking
    Returns  bool
is_starred(login, repo)
    Check if the authenticated user starred login/repo.
    Parameters
        • login (str) – (required), owner of repository
        • repo (str) – (required), name of repository
    Returns  bool
is_subscribed(login, repo)
    Check if the authenticated user is subscribed to login/repo.
    Parameters
        • login (str) – (required), owner of repository
        • repo (str) – (required), name of repository
    Returns  bool
issue(owner, repository, number)
    Fetch issue #:number: from https://github.com/:owner/:repository:
    Parameters
        • owner (str) – (required), owner of the repository
        • repository (str) – (required), name of the repository
        • number (int) – (required), issue number
    Returns  Issue
iter_all_repos(number=-1, etag=None)
    Iterate over every repository in the order they were created.
    Parameters
        • number (int) – (optional), number of repositories to return. Default: -1, returns all of them
        • etag (str) – (optional), ETag from a previous request to the same endpoint
    Returns  generator of Repository
iter_all_users(number=-1, etag=None)
    Iterate over every user in the order they signed up for GitHub.
    Parameters
        • number (int) – (optional), number of users to return. Default: -1, returns all of them
• **etag (str)**  – (optional), ETag from a previous request to the same endpoint

**Returns**  generator of *User*

**iter_authorizations** *(number=-1, etag=None)*
Iterate over authorizations for the authenticated user. This will return a 404 if you are using a token for authentication.

**Parameters**
• **number (int)**  – (optional), number of authorizations to return. Default: -1 returns all available authorizations
• **etag (str)**  – (optional), ETag from a previous request to the same endpoint

**Returns**  generator of *Authorizations*

**iter_emails** *(number=-1, etag=None)*
Iterate over email addresses for the authenticated user.

**Parameters**
• **number (int)**  – (optional), number of email addresses to return. Default: -1 returns all available email addresses
• **etag (str)**  – (optional), ETag from a previous request to the same endpoint

**Returns**  generator of dicts

**iter_events** *(number=-1, etag=None)*
Iterate over public events.

**Parameters**
• **number (int)**  – (optional), number of events to return. Default: -1 returns all available events
• **etag (str)**  – (optional), ETag from a previous request to the same endpoint

**Returns**  generator of *Events*

**iter_followers** *(login=None, number=-1, etag=None)*
If login is provided, iterate over a generator of followers of that login name; otherwise return a generator of followers of the authenticated user.

**Parameters**
• **login (str)**  – (optional), login of the user to check
• **number (int)**  – (optional), number of followers to return. Default: -1 returns all followers
• **etag (str)**  – (optional), ETag from a previous request to the same endpoint

**Returns**  generator of *Users*

**iter_following** *(login=None, number=-1, etag=None)*
If login is provided, iterate over a generator of users being followed by login; otherwise return a generator of people followed by the authenticated user.

**Parameters**
• **login (str)**  – (optional), login of the user to check
• **number (int)**  – (optional), number of people to return. Default: -1 returns all people you follow
• **etag (str)**  – (optional), ETag from a previous request to the same endpoint
Returns generator of Users

ter_gists(username=None, number=-1, etag=None)
If no username is specified, GET /gists, otherwise GET /users/:username/gists

Parameters
- **login** (str) – (optional), login of the user to check
- **number** (int) – (optional), number of gists to return. Default: -1 returns all available gists
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Gists

ter_issues(filter='', state='', labels='', sort='', direction='', since='', number=-1, etag=None)
List all of the authenticated user’s (and organization’s) issues.

Parameters
- **filter** (str) – accepted values: ('assigned', 'created', 'mentioned', 'subscribed') api-default: ‘assigned’
- **state** (str) – accepted values: ('open', 'closed') api-default: ‘open’
- **labels** (str) – comma-separated list of label names, e.g., ‘bug.ui,@high’
- **sort** (str) – accepted values: ('created', 'updated', 'comments') api-default: created
- **direction** (str) – accepted values: ('asc', 'desc') api-default: desc
- **since** (str) – ISO 8601 formatted timestamp, e.g., 2012-05-20T23:10:27Z
- **number** (int) – (optional), number of issues to return. Default: -1 returns all issues
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Issue

ter_keys(number=-1, etag=None)
Iterate over public keys for the authenticated user.

Parameters
- **number** (int) – (optional), number of keys to return. Default: -1 returns all your keys
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Keys

ter_notifications(all=False, participating=False, number=-1, etag=None)
Iterate over the user’s notification.

Parameters
- **all** (bool) – (optional), iterate over all notifications
- **participating** (bool) – (optional), only iterate over notifications in which the user is participating
- **number** (int) – (optional), how many notifications to return
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Thread

ter_org_issues(name, filter='', state='', labels='', sort='', direction='', since='', number=-1, etag=None)
Iterate over the organization’s issues if the authenticated user belongs to it.
Parameters

- **name** (*str*) – (required), name of the organization
- **state** (*str*) – accepted values: (‘open’, ‘closed’) api-default: ‘open’
- **labels** (*str*) – comma-separated list of label names, e.g., ‘bug,ui,@high’
- **sort** (*str*) – accepted values: (‘created’, ‘updated’, ‘comments’) api-default: created
- **direction** (*str*) – accepted values: (‘asc’, ‘desc’) api-default: desc
- **since** (*str*) – ISO 8601 formatted timestamp, e.g., 2012-05-20T23:10:27Z
- **number** (*int*) – (optional), number of issues to return. Default: -1, returns all available issues
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of **Issue**

**iter_orgs** *(login=None, number=-1, etag=None)*

Iterate over public organizations for login if provided; otherwise iterate over public and private organizations for the authenticated user.

Parameters

- **login** (*str*) – (optional), user whose orgs you wish to list
- **number** (*int*) – (optional), number of organizations to return. Default: -1 returns all available organizations
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of **Organizations**

**iter_repo_issues** *(owner, repository, milestone=None, state=None, assignee=None, mentioned=None, labels=None, sort=None, direction=None, since=None, number=-1, etag=None)*

List issues on owner/repository. Only owner and repository are required.

Parameters

- **owner** (*str*) – login of the owner of the repository
- **repository** (*str*) – name of the repository
- **milestone** (*int*) – None, ‘*’, or ID of milestone
- **state** (*str*) – accepted values: (‘open’, ‘closed’) api-default: ‘open’
- **assignee** (*str*) – ‘*’ or login of the user
- **mentioned** (*str*) – login of the user
- **labels** (*str*) – comma-separated list of label names, e.g., ‘bug,ui,@high’
- **sort** (*str*) – accepted values: (‘created’, ‘updated’, ‘comments’) api-default: created
- **direction** (*str*) – accepted values: (‘asc’, ‘desc’) api-default: desc
- **since** (*str*) – ISO 8601 formatted timestamp, e.g., 2012-05-20T23:10:27Z
- **number** (*int*) – (optional), number of issues to return. Default: -1 returns all issues
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint
Returns generator of `Issue`

`iter_repos(login=None, type='', sort='', direction='', number=-1, etag=None)`
List public repositories for the specified `login` or all repositories for the authenticated user if `login` is not provided.

Parameters
- `login (str)` – (optional)
- `type (str)` – (optional), accepted values: (‘all’, ‘owner’, ‘public’, ‘private’, ‘member’) API default: ‘all’
- `sort (str)` – (optional), accepted values: (‘created’, ‘updated’, ‘pushed’, ‘full_name’) API default: ‘created’
- `direction (str)` – (optional), accepted values: (‘asc’, ‘desc’), API default: ‘asc’ when using ‘full_name’, ‘desc’ otherwise
- `number (int)` – (optional), number of repositories to return. Default: -1 returns all repositories
- `etag (str)` – (optional), ETag from a previous request to the same endpoint

Returns generator of `Repository` objects

`iter_starred(login=None, sort=None, direction=None, number=-1, etag=None)`
Iterate over repositories starred by `login` or the authenticated user. Changed in version 0.5: Added sort and direction parameters (optional) as per the change in GitHub’s API.

Parameters
- `login (str)` – (optional), name of user whose stars you want to see
- `sort (str)` – (optional), either ‘created’ (when the star was created) or ‘updated’ (when the repository was last pushed to)
- `direction (str)` – (optional), either ‘asc’ or ‘desc’. Default: ‘desc’
- `number (int)` – (optional), number of repositories to return. Default: -1 returns all repositories
- `etag (str)` – (optional), ETag from a previous request to the same endpoint

Returns generator of `Repository`

`iter_subscriptions(login=None, number=-1, etag=None)`
Iterate over repositories subscribed to by `login` or the authenticated user.

Parameters
- `login (str)` – (optional), name of user whose subscriptions you want to see
- `number (int)` – (optional), number of repositories to return. Default: -1 returns all repositories
- `etag (str)` – (optional), ETag from a previous request to the same endpoint

Returns generator of `Repository`

`iter_user_issues(filter='', state='', labels='', sort='', direction='', since='', number=-1, etag=None)`
List only the authenticated user’s issues. Will not list organization’s issues.

Parameters
• **state** *(str)* – accepted values: (‘open’, ‘closed’) api-default: ‘open’
• **labels** *(str)* – comma-separated list of label names, e.g., ‘bug,ui,@high’
• **sort** *(str)* – accepted values: (‘created’, ‘updated’, ‘comments’) api-default: created
• **direction** *(str)* – accepted values: (‘asc’, ‘desc’) api-default: desc
• **since** *(str)* – ISO 8601 formatted timestamp, e.g., 2012-05-20T23:10:27Z
• **number** *(int)* – (optional), number of issues to return. Default: -1 returns all issues
• **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Issue**

**key** *(id_num)*

Gets the authenticated user’s key specified by id_num.

**Parameters**

- **id_num** *(int)* – (required), unique id of the key

**Returns** **Key**

**login** *(username=None, password=None, token=None)*

Logs the user into GitHub for protected API calls.

**Parameters**

- **username** *(str)* – (optional)
- **password** *(str)* – (optional)
- **token** *(str)* – (optional)

**markdown** *(text, mode='', context='', raw=False)*

Render an arbitrary markdown document.

**Parameters**

- **text** *(str)* – (required), the text of the document to render
- **mode** *(str)* – (optional), ‘markdown’ or ‘gfm’
- **context** *(str)* – (optional), only important when using mode ‘gfm’, this is the repository to use as the context for the rendering
- **raw** *(bool)* – (optional), renders a document like a README.md, no gfm, no context

**Returns** **str** – HTML formatted text

**meta** *

Returns an array of addresses in CIDR format specifying the addresses that the incoming service hooks will originate from. New in version 0.5.

**octocat** *(say=None)*

Returns an easter egg of the API.

**Params** **str** **say** (optional), pass in what you’d like Octocat to say

**Returns** ascii art of Octocat

**organization** *(login)*

Returns a Organization object for the login name

**Parameters**

- **login** *(str)* – (required), login name of the org
Returns Organization

pubsubhubbub (mode, topic, callback, secret='')
Create/update a pubsubhubbub hook.

Parameters

- **mode** (str) – (required), accepted values: (‘subscribe’, ‘unsubscribe’)
- **topic** (str) – (required), form: https://github.com/:user/:repo/events/:event
- **callback** (str) – (required), the URI that receives the updates
- **secret** (str) – (optional), shared secret key that generates a SHA1 HMAC of the payload content.

Returns bool

pull_request(owner, repository, number)
Fetch pull_request #:number: from :owner/:repository

Parameters

- **owner** (str) – (required), owner of the repository
- **repository** (str) – (required), name of the repository
- **number** (int) – (required), issue number

Returns Issue

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters **conditional** (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self

The reasoning for the return value is the following example:

```
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]
```

Without the return value, that would be an array of None’s and you would otherwise have to do:

```
repos = [r for i in g.iter_repos('kennethreitz')]  
[r.refresh() for r in repos]
```

Which is really an anti-pattern. Changed in version 0.5.

repository(owner, repository)
Returns a Repository object for the specified combination of owner and repository

Parameters

- **owner** (str) – (required)
- **repository** (str) – (required)

Returns Repository

search_email(email)
Search users by email.
**Parameters**

- **email** (*str*) – (required)

**Returns**  
*LegacyUser*

**search_issues( owner, repo, state, keyword, start_page=0)**

Find issues by state and keyword.

**Parameters**

- **owner** (*str*) – (required)
- **repo** (*str*) – (required)
- **state** (*str*) – (required), accepted values: ('open', 'closed')
- **keyword** (*str*) – (required), what to search for
- **start_page** (*int*) – (optional), page to get (results come 100/page)

**Returns**  
list of *LegacyIssues*

**search_repos( keyword, language=None, start_page=None, sort=None, order=None)**

Search all repositories by keyword.

**Parameters**

- **keyword** (*str*) – (required)
- **language** (*str*) – (optional), language to filter by
- **start_page** (*int*) – (optional), page to get (results come 100/page)
- **sort** (*str*) – (optional), how to sort the results; accepted values: ('stars', 'forks', 'updated')
- **order** (*str*) – (optional), sort order if sort isn't provided, accepted values: ('asc', 'desc')

**Returns**  
list of *LegacyRepos*

**search_users( keyword, start_page=0, sort=None, order=None)**

Search all users by keyword.

**Parameters**

- **keyword** (*str*) – (required)
- **start_page** (*int*) – (optional), page to get (results come 100/page)
- **sort** (*str*) – (optional), how to sort the results; accepted values: ('followers', 'joined', 'repositories')
- **order** (*str*) – (optional), sort order if sort isn't provided, accepted values: ('asc', 'desc')

**Returns**  
list of *LegacyUsers*

**set_client_id(id, secret)**

Allows the developer to set their client_id and client_secret for their OAuth application.

**Parameters**

- **id** (*str*) – 20-character hexidecimal client_id provided by GitHub
- **secret** (*str*) – 40-character hexidecimal client_secret provided by GitHub

**set_user_agent(user_agent)**

Allows the user to set their own user agent string to identify with the API.

**Parameters**

- **user_agent** (*str*) – String used to identify your application.  Library default: “github3.py/{version}”, e.g., “github3.py/0.5”
star (login, repo)
  Star to login/repo

  Parameters
  • login (str) – (required), owner of the repo
  • repo (str) – (required), name of the repo

  Returns bool

subscribe (login, repo)
  Subscribe to login/repo

  Parameters
  • login (str) – (required), owner of the repo
  • repo (str) – (required), name of the repo

  Returns bool

to_json ()
  Return the json representing this object.

unfollow (login)
  Make the authenticated user stop following login

  Parameters login (str) – (required)

  Returns bool

unstar (login, repo)
  Unstar to login/repo

  Parameters
  • login (str) – (required), owner of the repo
  • repo (str) – (required), name of the repo

  Returns bool

unsubscribe (login, repo)
  Unsubscribe to login/repo

  Parameters
  • login (str) – (required), owner of the repo
  • repo (str) – (required), name of the repo

  Returns bool

update_user (name=None, email=None, blog=None, company=None, location=None, hireable=False, bio=None)
  If authenticated as this user, update the information with the information provided in the parameters. All parameters are optional.

  Parameters
  • name (str) – e.g., ‘John Smith’, not login name
  • email (str) – e.g., ‘john.smith@example.com’
  • blog (str) – e.g., ‘http://www.example.com/jsmith/blog’
  • company (str) – company name
• **location** *(str)* – where you are located
• **hireable** *(bool)* – defaults to False
• **bio** *(str)* – GitHub flavored markdown

Returns  **bool**

**user** *(login=None)*

Returns a User object for the specified login name if provided. If no login name is provided, this will return a User object for the authenticated user.

**Parameters**

- **login** *(str)* – (optional)

Returns  **User**

**zen()**

Returns a quote from the Zen of GitHub. Yet another API Easter Egg

Returns  **str**

### 2.6.3 GitHubEnterprise Object

This has all of the same attributes as the GitHub object so for brevity’s sake, I’m not listing all of it’s inherited members.

**class**  **github3.github.GitHubEnterprise**( *url, login='', password='', token=''* )

For GitHub Enterprise users, this object will act as the public API to your instance. You must provide the URL to your instance upon initialization and can provide the rest of the login details just like in the GitHub object.

There is no need to provide the end of the url (e.g., /api/v3/), that will be taken care of by us.

**admin_stats**( *option* )

This is a simple way to get statistics about your system.

**Parameters**


Returns  **dict**

### 2.6.4 GitHubStatus Object

**class**  **github3.github.GitHubStatus**

A sleek interface to the GitHub System Status API. This will only ever return the JSON objects returned by the API.

**api()**

GET /api.json

**last_message()**

GET /api/last-message.json

**messages()**

GET /api/messages.json

**status()**

GET /api/status.json
2.7 Issue

This part of the documentation covers the module which handles Issues and their related objects:

- IssueComment
- IssueEvent
- Milestone
- Label.

2.7.1 Issue Objects

class github3.issues.Issue(issue, session=None)
The Issue object. It structures and handles the data returned via the Issues section of the GitHub API.

add_labels(*args)
Add labels to this issue.

Parameters args (str) – (required), names of the labels you wish to add

Returns list of Labels

assign(login)
Assigns user login to this issue. This is a short cut for issue.edit.

Parameters login (str) – username of the person to assign this issue to

Returns bool

assignee = None
User representing the user the issue

body = None
Body (description) of the issue.

body_html = None
HTML formatted body of the issue.

body_text = None
Plain text formatted body of the issue.

close()
Close this issue.

Returns bool

closed_at = None
datetime object representing when the issue was closed.

comment(id_num)
Get a single comment by its id.

The catch here is that id is NOT a simple number to obtain. If you were to look at the comments on issue #15 in sigmavirus24/Todo.txt-python, the first comment’s id is 4150787.

Parameters id_num (int) – (required), comment id, see example above

Returns IssueComment

comments = None
Number of comments on this issue.
create_comment (body)
Create a comment on this issue.

Parameters  body (str) – (required), comment body

Returns  IssueComment

created_at = None
datetime object representing when the issue was created.

edit (title=None, body=None, assignee=None, state=None, milestone=None, labels=None)
Edit this issue.

Parameters
  •  title (str) – Title of the issue
  •  body (str) – markdown formatted body (description) of the issue
  •  assignee (str) – login name of user the issue should be assigned to
  •  state (str) – accepted values: (‘open’, ‘closed’)
  •  milestone (int) – the NUMBER (not title) of the milestone to assign this to
  •  labels (list) – list of labels to apply this to

Returns  bool

classmethod from_json (json)
Return an instance of cls formed from json.

html_url = None
URL to view the issue at GitHub.

id = None
Unique ID for the issue.

is_closed ()
Checks if the issue is closed.

Returns  bool

iter_comments (number=-1)
Iterate over the comments on this issue.

Parameters  number (int) – (optional), number of comments to iterate over

Returns  iterator of IssueComment

iter_events (number=-1)
Iterate over events associated with this issue only.

Parameters  number (int) – (optional), number of events to return. Default: -1 returns all events available.

Returns  generator of IssueEvents

labels = None
Returns the list of Labels on this issue.

milestone = None
Milestone this issue was assigned to.

1 Milestone numbering starts at 1, i.e. the first milestone you create is 1, the second is 2, etc.
number = None
    Issue number (e.g. #15)

pull_request = None
    Dictionary URLs for the pull request (if they exist)

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh (conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header ('Last-Modified', or 'ETag') on the object and send that as described in the Conditional Requests section of the docs

    Returns self

    The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

remove_all_labels ()
    Remove all labels from this issue.

    Returns an empty list if successful

remove_label (name)
    Removes label name from this issue.

    Parameters name (str) – (required), name of the label to remove

    Returns bool

reopen ()
    Re-open a closed issue.

    Returns bool

replace_labels (labels)
    Replace all labels on this issue with labels.

    Parameters labels (list) – label names

    Returns bool

repository = None
    Returns ('owner', 'repository') this issue was filed on.

state = None
    State of the issue, e.g., open, closed

title = None
    Title of the issue.

to_json ()
    Return the json representing this object.
updated_at = None
datetime object representing the last time the issue was updated.

user = None
User who opened the issue.

class github3.issues.IssueComment(comment, session=None)
The IssueComment object. This structures and handles the comments on issues specifically.

def delete()
    Delete this comment.

    Returns bool

def edit(body)
    Edit this comment.

    Parameters
        - body (str) -- (required), new body of the comment, Markdown formatted

    Returns bool

classmethod from_json(json)
    Return an instance of cls formed from json.

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

    Parameters
        - conditional (bool) -- If True, then we will search for a stored header ('Last-Modified', or 'ETag') on the object and send that as described in the Conditional Requests section of the docs

    Returns self

    The reasoning for the return value is the following example:
    repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

    Without the return value, that would be an array of None's and you would otherwise have to do:
    repos = [r for i in g.iter_repos('kennethreitz')]
    [r.refresh() for r in repos]

    Which is really an anti-pattern. Changed in version 0.5.

to_json()
    Return the json representing this object.

user = None
User who made the comment

class github3.issues.IssueEvent(event, issue=None)
The IssueEvent object. This specifically deals with events described in the Issues>Events section of the GitHub API.

comments = None
Number of comments
commit_id = None
    SHA of the commit.
created_at = None
    datetime object representing when the event was created.
event = None
    The type of event, e.g., closed
classmethod from_json(json)
    Return an instance of cls formed from json.
issue = None
    Issue where this comment was made.
pull_request = None
    Dictionary of links for the pull request
ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.
refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header ('Last-Modified', or 'ETag') on the object and send that as described in the Conditional Requests section of the docs

    Returns self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None's and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

to_json()
    Return the json representing this object.

class github3.issues.Milestone(mile, session=None)
    The Milestone object. This is a small class to handle information about milestones on repositories and issues.

closed_issues = None
    The number of closed issues associated with this milestone.
created_at = None
    datetime object representing when the milestone was created.
creator = None
    User object representing the creator

delete()
    Delete this milestone.

    Returns bool
description = None
    Description of this milestone.

due_on = None
    datetime representing when this milestone is due.

classmethod from_json(json)
    Return an instance of cls formed from json.

iter_labels(number=-1)
    Iterate over the labels for every issue associated with this milestone.

    Parameters number (int) – (optional), number of labels to return. Default: -1 returns all available labels.

    Returns generator of Labels

number = None
    Identifying number associated with milestone.

open_issues = None
    Number of issues associated with this milestone which are still

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

    Returns self

The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos(‘kennethreitz’)]

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos(‘kennethreitz’)]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

state = None
    State of the milestone, e.g., open or closed.

title = None
    Title of the milestone, e.g., 0.2.

to_json()
    Return the json representing this object.

update(title='', state='', description='', due_on='')
    Update this milestone.

    state, description, and due_on are optional

    Parameters
        • title (str) – (required), new title of the milestone
        • state (str) – (optional), (‘open’, ‘closed’)

48 Chapter 2. Modules
• **description** (*str*) – (optional)

• **due_on** (*str*) – (optional), ISO 8601 time format: YYYY-MM-DDTHH:MM:SSZ

**Returns** bool

class `github3.issues.Label`(*label*, session=None)

The **Label** object. Succinctly represents a label that exists in a repository.

**color** = None

Color of the label, e.g., 626262

**delete**()

Delete this label.

**Returns** bool

classmethod **from_json**(json)

Return an instance of cls formed from json.

**name** = None

Name of the label, e.g., ‘bug’

**ratelimit_remaining**

Number of requests before GitHub imposes a ratelimit.

**refresh**(conditional=False)

Re-retrieve the information for this object and returns the refreshed instance.

**Parameters** conditional (*bool*) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

**Returns** self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

**to_json**()

Return the json representing this object.

**update**(name, color)

Update this label.

**Parameters**

• **name** (*str*) – (required), new name of the label

• **color** (*str*) – (required), color code, e.g., 626262, no leading ‘#’

**Returns** bool

2.7. Issue
2.8 Legacy

This part of the documentation covers the legacy objects used in the Search section of the API.

2.8.1 Legacy Objects

class github3.legacy.LegacyIssue(issue, session=None)

The LegacyIssue object. This object is only every used in conjuction with the search_issues. Unfortunately, GitHub hasn’t updated the search functionality to use the objects as they exist now.

    body = None
    Body of the issue

    comments = None
    Number of comments on the issue

    created_at = None
    datetime object representing the creation of the issue

    classmethod from_json(json)
    Return an instance of cls formed from json.

    gravatar_id = None
    id of the gravatar account

    html_url = None
    URL of the issue

    labels = None
    list of labels applied to this issue

    number = None
    Issue number

    position = None
    Position

    ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

    refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

    Returns self

    The reasoning for the return value is the following example:

    repos = [r.refresh() for r in g.iter_repos(‘kennethreitz’)]

    Without the return value, that would be an array of None’s and you would otherwise have to do:

    repos = [r for i in g.iter_repos(‘kennethreitz’)]
    [r.refresh() for r in repos]

    Which is really an anti-pattern. Changed in version 0.5.
state = None
State of the issue, i.e., open or closed

title = None
Title of the issue

to_json()
Return the json representing this object.

updated_at = None
datetime object representing the last time the issue was updated

user = None
User’s login

votes = None
Number of votes on this issue. Probably effectively deprecated

class github3.legacy.LegacyRepo (repo, session=None)
The LegacyRepo object. This wraps data returned using the search_repos

created = None
datetime object representing the date of creation of this repo

created_at = None
datetime object representing the date of creation of this repo

description = None
description of this repository

followers = None
Number of followers

forks = None
Number of forks of this repository

classmethod from_json (json)
Return an instance of cls formed from json.

has_downloads = None
Boolean - whether the repository has downloads or not

has_issues = None
Boolean - whether the repository has issues or not

has_wiki = None
Boolean - whether the repository has a wiki or not

homepage = None
URL of the website for this repository

is_private()
Checks if this repository is private

language = None
Language used in this repository

name = None
Name of this repository

open_issues = None
Number of open issues
owner = None
Owner of this repository

private = None
Whether the repository is private or not

pushed = None
datetime object representing the last time the repo was pushed to

pushed_at = None
datetime object representing the last time the repo was pushed to

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh (conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters
conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns
self

The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

score = None
Score

size = None
Size of the repo

to_json ()
Return the json representing this object.

type = None
Type of object

url = None
URL of the project on GitHub

user = None
User’s login

watchers = None
Number of people watching this project

class github3.legacy.LegacyUser (user, session=None)
The LegacyUser object. This handles information returned by search_users.

created = None
datetime object representing when the account was created
created_at = None
datetime object representing when the account was created

followers = None
Number of followers

followers_count = None
Number of followers

classmethod from_json(json)
Return an instance of cls formed from json.

fullname = None
Full name of this user

gravatar_id = None
Gravatar id for this user

id = None
Unique id of this user

language = None
Language

location = None
Location of this user

login = None
username for the user

name = None
Full name of this user

public_repo_count = None
Number of public repos owned by this user

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh (conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header ('Last-Modified', or 'ETag') on the object and send that as described in the Conditional Requests section of the docs

Returns self

The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None's and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

repos = None
Number of repos owned by the user

score = None
Score
to_json()
    Return the json representing this object.

type = None
    Type of user

username = None
    User’s login

2.9 Models

This part of the documentation covers a lot of lower-level objects that are never directly seen or used by the user (developer). They are documented for future developers of this library.

2.9.1 Objects

class github3.models.GitHubCore (json, ses=None)
    The GitHubCore object. This class provides some basic attributes to other classes that are very useful to have.

    @classmethod
    def from_json (json)
        Return an instance of cls formed from json.

    ratelimit_remaining
        Number of requests before GitHub imposes a ratelimit.

    refresh (conditional=False)
        Re-retrieve the information for this object and returns the refreshed instance.

            Parameters
            ----------
            conditional : bool, optional
                If True, then we will search for a stored header ('Last-Modified', or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

            Returns
            -------
            self

        The reasoning for the return value is the following example:

        repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

        Without the return value, that would be an array of None’s and you would otherwise have to do:

        repos = [r for i in g.iter_repos('kennethreitz')]
        [r.refresh() for r in repos]

        Which is really an anti-pattern. Changed in version 0.5.

to_json()
    Return the json representing this object.

class github3.models.BaseAccount (acct, session)
    The BaseAccount object. This is used to do the heavy lifting for Organization and User objects.

    avatar_url = None
        URL of the avatar at gravatar

    bio = None
        Markdown formatted biography
blog = None
    URL of the blog

company = None
    Name of the company

created_at = None
    datetime object representing the date the account was created

email = None
    E-mail address of the user/org

followers = None
    Number of followers

following = None
    Number of people the user is following

classmethod from_json(json)
    Return an instance of cls formed from json.

html_url = None
    URL of the user/org’s profile

id = None
    Unique ID of the account

location = None
    Location of the user/org

login = None
    login name of the user/org

name = None
    Real name of the user/org

public_repos = None
    Number of public repos owned by the user/org

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

    Returns self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

to_json()
    Return the json representing this object.
type = None
    Tells you what type of account this is

class github3.models.BaseComment (comment, session)
The BaseComment object. A basic class for Gist, Issue and Pull Request Comments.

    body = None
        Body of the comment. (As written by the commenter)

    body_html = None
        Body of the comment formatted as html.

    body_text = None
        Body of the comment formatted as plain-text. (Stripped of markdown,

    created_at = None
        datetime object representing when the comment was created.

    delete ()
        Delete this comment.

        Returns bool

    edit (body)
        Edit this comment.

        Parameters body (str) – (required), new body of the comment, Markdown formatted

        Returns bool

    classmethod from_json (json)
        Return an instance of cls formed from json.

    html_url = None
        The url of this comment at GitHub

    id = None
        Unique ID of the comment.

    pull_request_url = None
        The url of the pull request, if it exists

    ratelimit_remaining
        Number of requests before GitHub imposes a ratelimit.

    refresh (conditional=False)
        Re-retrieve the information for this object and returns the refreshed instance.

        Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-
            Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests
            section of the docs

        Returns self

        The reasoning for the return value is the following example:

        repos = [r.refresh() for r in g.iter_repos(‘kennethreitz’)]

        Without the return value, that would be an array of None’s and you would otherwise have to do:

        repos = [r for i in g.iter_repos(‘kennethreitz’)]
        [r.refresh() for r in repos]
Which is really an anti-pattern. Changed in version 0.5.

to_json()
    Return the json representing this object.

updated_at = None
    datetime object representing when the comment was updated.

class github3.models.BaseCommit (commit, session)
The BaseCommit object. This serves as the base for the various types of commit objects returned by the API.

classmethod from_json(json)
    Return an instance of cls formed from json.

message = None
    Commit message

parents = None
    List of parents to this commit.

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

    Returns self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

sha = None
    SHA of this commit.

to_json()
    Return the json representing this object.

2.10 Organization

This section of the documentation covers:

* Organizations
* Teams
2.10.1 Organization Objects

class github3.orgs.Organization (org, session=None)
   The Organization object.

    add_member (login, team)
   Add login to team and thereby to this organization.

   Any user that is to be added to an organization, must be added to a team as per the GitHub api.

   Note: This method is of complexity O(n). This iterates over all teams in your organization and only adds
   the user when the team name matches the team parameter above. If you want constant time, you should
   retrieve the team and call add_member on that team directly.

    Parameters
     • login (str) – (required), login name of the user to be added
     • team (str) – (required), team name

    Returns bool

    add_repo (repo, team)
   Add repo to team.

   Note: This method is of complexity O(n). This iterates over all teams in your organization and only adds
   the repo when the team name matches the team parameter above. If you want constant time, you should
   retrieve the team and call add_repo on that team directly.

    Parameters
     • repo (str) – (required), form: ‘user/repo’
     • team (str) – (required), team name

    conceal_member (login)
   Conceal login’s membership in this organization.

    Returns bool

    create_repo (name, description='', homepage='', private=False, has_issues=True, has_wiki=True,
                has_downloads=True, team_id=0, auto_init=False, gitignore_template='')
   Create a repository for this organization if the authenticated user is a member.

    Parameters
     • name (str) – (required), name of the repository
     • description (str) – (optional)
     • homepage (str) – (optional)
     • private (bool) – (optional), If True, create a private repository. API default: False
     • has_issues (bool) – (optional), If True, enable issues for this repository. API default: True
     • has_wiki (bool) – (optional), If True, enable the wiki for this repository. API default: True
• **has_downloads** (*bool*) – (optional), If True, enable downloads for this repository. API default: True

• **team_id** (*int*) – (optional), id of the team that will be granted access to this repository

• **auto_init** (*bool*) – (optional), auto initialize the repository.

• **gitignore_template** (*str*) – (optional), name of the template; this is ignored if auto_int = False.

`create_team` *(name, repo_names=[], permissions='')*  
Assuming the authenticated user owns this organization, create and return a new team.

Parameters

• **name** (*str*) – (required), name to be given to the team

• **repo_names** (*list*) – (optional) repositories, e.g. ['github/dotfiles']

• **permissions** (*str*) – (optional), options:
  - **pull** – (default) members can not push or administer repositories accessible by this team
  - **push** – members can push and pull but not administer repositories accessible by this team
  - **admin** – members can push, pull and administer repositories accessible by this team

`edit` *(billing_email=None, company=None, email=None, location=None, name=None)*  
Edit this organization.

Parameters

• **billing_email** (*str*) – (optional) Billing email address (private)

• **company** (*str*) – (optional)

• **email** (*str*) – (optional) Public email address

• **location** (*str*) – (optional)

• **name** (*str*) – (optional)

Returns **bool**

`from_json` *(json)*  
Return an instance of cls formed from json.

`is_member` *(login)*  
Check if the user with login login is a member.

Returns **bool**

`is_public_member` *(login)*  
Check if the user with login login is a public member.

Returns **bool**

`iter_events` *(number=-1, etag=None)*  
Iterate over events for this org.

Parameters
• **number (int)** – (optional), number of events to return. Default: -1 iterates over all events available.

• **etag (str)** – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Events**

**iter_members (number=-1, etag=None)**
Iterate over members of this organization.

**Parameters**

• **number (int)** – (optional), number of members to return. Default: -1 will return all available.

• **etag (str)** – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Users**

**iter_public_members (number=-1, etag=None)**
Iterate over public members of this organization.

**Parameters**

• **number (int)** – (optional), number of members to return. Default: -1 will return all available.

• **etag (str)** – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Users**

**iter_repos (type='', number=-1, etag=None)**
Iterate over repos for this organization.

**Parameters**

• **type (str)** – (optional), accepted values: ('all', 'public', 'member', 'private', 'forks', 'sources'), API default: ‘all’

• **number (int)** – (optional), number of members to return. Default: -1 will return all available.

• **etag (str)** – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Repository**

**iter_teams (number=-1, etag=None)**
Iterate over teams that are part of this organization.

**Parameters**

• **number (int)** – (optional), number of teams to return. Default: -1 returns all available teams.

• **etag (str)** – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Teams**

**private_repos = None**
Number of private repositories.

**publicize_member (login)**
Make login’s membership in this organization public.

**Returns** bool
**ratelimit_remaining**
Number of requests before GitHub imposes a ratelimit.

**refresh**(conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters **conditional** (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self
The reasoning for the return value is the following example:

```python
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]
```

Without the return value, that would be an array of None’s and you would otherwise have to do:

```python
repos = [r for i in g.iter_repos('kennethreitz')] [r.refresh() for r in repos]
```

Which is really an anti-pattern. Changed in version 0.5.

**remove_member**(login)
Remove the user with login login from this organization.

Returns bool

**remove_repo**(repo, team)
Remove repo from team.

Parameters

- **repo** (str) – (required), form: ‘user/repo’
- **team** (str) – (required)

Returns bool

**team**(team_id)
Returns Team object with information about team specified by team_id.

Parameters **team_id** (int) – (required), unique id for the team

Returns Team

**to_json**()
Return the json representing this object.

class github3.orgs.Team(team, session=None)

**add_member**(login)
Add login to this team.

Returns bool

**add_repo**(repo)
Add repo to this team.

Parameters **repo** (str) – (required), form: ‘user/repo’

Returns bool
delete()
Delete this team.

Returns bool

edit(name, permission='')
Edit this team.

Parameters
• name (str) – (required)
• permission (str) – (optional), ('pull', 'push', 'admin')

Returns bool

classmethod from_json(json)
Return an instance of cls formed from json.

has_repo(repo)
Checks if this team has access to repo

Parameters repo (str) – (required), form: 'user/repo'

Returns bool

id = None
Unique ID of the team.

is_member(login)
Check if login is a member of this team.

Parameters login (str) – (required), login name of the user

Returns bool

iter_members(number=-1, etag=None)
Iterate over the members of this team.

Parameters
• number (int) – (optional), number of users to iterate over. Default: -1 iterates over all values
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Users

iter_repos(number=-1, etag=None)
Iterate over the repositories this team has access to.

Parameters
• number (int) – (optional), number of repos to iterate over. Default: -1 iterates over all values
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Repository objects

members_count = None
Number of members in this team.

name = None
This team’s name.
permission = None
    Permission level of the group

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh (conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.
    
    Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

    Returns self

The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')] [r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

remove_member (login)
    Remove login from this team.

    Parameters login (str) – (required), login of the member to remove

    Returns bool

remove_repo (repo)
    Remove repo from this team.

    Parameters repo (str) – (required), form: ‘user/repo’

    Returns bool

repos_count = None
    Number of repos owned by this team.

to_json ()
    Return the json representing this object.

## 2.11 Pull Request

This section of the documentation covers:

- PullRequest
- ReviewComment
- PullDestination
- PullFile
2.11.1 Pull Request Objects

class github3.pulls.PullRequest (pull, session=None)
The PullRequest object.

    base = None
    Base of the merge

    body = None
    Body of the pull request message

    body_html = None
    Body of the pull request as HTML

    body_text = None
    Body of the pull request as plain text

    close()
    Closes this Pull Request without merging.

        Returns
        bool

    closed_at = None
    datetime object representing when the pull was closed

    created_at = None
    datetime object representing when the pull was created

    diff()
    Return the diff

    diff_url = None
    URL to view the diff associated with the pull

    classmethod from_json(json)
    Return an instance of cls formed from json.

    head = None
    The new head after the pull request

    html_url = None
    The URL of the pull request

    id = None
    The unique id of the pull request

    is_merged()
    Checks to see if the pull request was merged.

        Returns
        bool

    issue_url = None
    The URL of the associated issue

    iter_comments (number=-1, etag=None)
    Iterate over the comments on this pull request.

        Parameters

        • number (int) – (optional), number of comments to return. Default: -1 returns all available comments.

        • etag (str) – (optional), ETag from a previous request to the same endpoint

        Returns generator of ReviewComments
iter_commits(\textit{number}=-1, \textit{etag}=None)
Iterates over the commits on this pull request.

Parameters

\begin{itemize}
  \item \textbf{number} (\textit{int}) – (optional), number of commits to return. Default: -1 returns all available commits.
  \item \textbf{etag} (\textit{str}) – (optional), ETag from a previous request to the same endpoint
\end{itemize}

Returns generator of \texttt{Commit}s

iter_files(\textit{number}=-1, \textit{etag}=None)
Iterate over the files associated with this pull request.

Parameters

\begin{itemize}
  \item \textbf{number} (\textit{int}) – (optional), number of files to return. Default: -1 returns all available files.
  \item \textbf{etag} (\textit{str}) – (optional), ETag from a previous request to the same endpoint
\end{itemize}

Returns generator of \texttt{PullFiles}s

\texttt{links} = None
Dictionary of _links

merge(\textit{commit\_message}='')
Merge this pull request.

Parameters \textbf{commit\_message} (\textit{str}) – (optional), message to be used for the merge commit

Returns bool

merge_commit_sha = None
SHA of the merge commit

mergeable = None
Whether the pull is deemed mergeable by GitHub

mergeable_state = None
Whether it would be a clean merge or not

merged_at = None
datetime object representing when the pull was merged

merged_by = None
User who merged this pull

number = None
Number of the pull/issue on the repository

patch()
Return the patch

patch_url = None
The URL of the patch

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh(\textit{conditional}=False)
Re-retrieve the information for this object and returns the refreshed instance.
Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs.

Returns self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

reopen()
Re-open a closed Pull Request.

Returns bool

repository = None
Returns (‘owner’, ‘repository’) this issue was filed on.

state = None
The state of the pull

title = None
The title of the request

to_json()
Return the json representing this object.

update(title=None, body=None, state=None)
Update this pull request.

Parameters

• title (str) – (optional), title of the pull
• body (str) – (optional), body of the pull request
• state (str) – (optional), (‘open’, ‘closed’)

Returns bool

updated_at = None
datetime object representing the last time the object was changed

user = None
User object representing the creator

class github3.pulls.ReviewComment (comment, session=None)

The ReviewComment object. This is used to represent comments on pull requests.

commit_id = None
SHA of the commit the comment is on

delete()
Delete this comment.

Returns bool
edit(body)
    Edit this comment.

    Parameters  body (str) – (required), new body of the comment, Markdown formatted

    Returns  bool

classmethod from_json(json)
    Return an instance of cls formed from json.

original_position = None
    Original position inside the file

path = None
    Path to the file

position = None
    Position within the commit

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

        Parameters  conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

        Returns  self

    The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

    Without the return value, that would be an array of None’s and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

    Which is really an anti-pattern. Changed in version 0.5.

to_json()
    Return the json representing this object.

user = None
    User who made the comment


class github3.pulls.PullDestination(dest, direction)
    The PullDestination object.

direction = None
    Direction of the merge with respect to this destination

classmethod from_json(json)
    Return an instance of cls formed from json.

label = None
    label of the destination

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.
ref = None
Full reference string of the object

refresh (conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header ('Last-Modified', or 'ETag') on the object and send that as described in the Conditional Requests section of the docs

Returns self
The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None's and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

sha = None
SHA of the commit at the head

to_json ()
Return the json representing this object.

user = None
User representing the owner

class github3.pulls.PullFile (pfile)
The PullFile object.

additions = None
Number of additions on this file

blob_url = None
URL to view the blob for this file

changes = None
Number of changes made to this file

deletions = None
Number of deletions on this file

filename = None
Name of the file

classmethod from_json (json)
Return an instance of cls formed from json.

patch = None
Patch generated by this pull request

raw_url = None
URL to view the raw diff of this file

sha = None
SHA of the commit
status = None
    Status of the file, e.g., ‘added’

to_json()
    Return the json representing this object.

2.12 Repository

This part of the documentation covers:

- Repository
- Branch
- Contents
- Download
- Hook
- RepoTag
- RepoComment
- RepoCommit
- Comparison
- Status

None of these objects should be instantiated directly by the user (developer). These are here for reference only.

When listing repositories in any context, GitHub refuses to return a number of attributes, e.g., source and parent. If you require these, call the refresh method on the repository object to make a second call to the API and retrieve those attributes.

More information for about this class can be found in the official documentation and in various other sections of the GitHub documentation.

2.12.1 Repository Objects

class github3.repos.Repository (repo, session=None)
    The Repository object. It represents how GitHub sends information about repositories.

    add_collaborator (login)
        Add login as a collaborator to a repository.

        Parameters login (str) – (required), login of the user

        Returns bool – True if successful, False otherwise

    archive (format, path='', ref='master')
        Get the tarball or zipball archive for this repo at ref.

        Parameters

        - format (str) – (required), accepted values: (‘tarball’, ‘zipball’)

        - path (str, file) – (optional), path where the file should be saved to, default is the
          filename provided in the headers and will be written in the current directory. it can
          take a file-like object as well
ref (str) – (optional)

Returns bool – True if successful, False otherwise

blob (sha)
Get the blob indicated by sha.

Parameters sha (str) – (required), sha of the blob

Returns Blob if successful, otherwise None

branch (name)
Get the branch name of this repository.

Parameters name (str) – (required), branch name

Returns Branch

clone_url = None
URL used to clone via HTTPS.

commit (sha)
Get a single (repo) commit. See git_commit() for the Git Data Commit.

Parameters sha (str) – (required), sha of the commit

Returns RepoCommit if successful, otherwise None

commit_comment (comment_id)
Get a single commit comment.

Parameters comment_id (int) – (required), id of the comment used by GitHub

Returns RepoComment if successful, otherwise None

compare_commits (base, head)
Compare two commits.

Parameters

• base (str) – (required), base for the comparison

• head (str) – (required), compare this against base

Returns Comparison if successful, else None

contents (path, ref=None)
Get the contents of the file pointed to by path.

Parameters

• path (str) – (required), path to file, e.g. github3/repo.py

• ref (str) – (optional), the string name of a commit/branch/tag. default: master

Returns Contents if successful, else None

create_blob (content, encoding)
Create a blob with content.

Parameters

• content (str) – (required), content of the blob

• encoding (str) – (required), (‘base64’, ‘utf-8’)

Returns string of the SHA returned
create_comment (body, sha, path='', position=1, line=1)
Create a comment on a commit.

Parameters

- **body** (str) – (required), body of the message
- **sha** (str) – (required), commit id
- **path** (str) – (optional), relative path of the file to comment on
- **position** (str) – (optional), line index in the diff to comment on
- **line** (int) – (optional), line number of the file to comment on, default: 1

Returns: RepoComment if successful else None

create_commit (message, tree, parents, author={}, committer={})
Create a commit on this repository.

Parameters

- **message** (str) – (required), commit message
- **tree** (str) – (required), SHA of the tree object this commit points to
- **parents** (list) – (required), SHAs of the commits that were parents of this commit. If empty, the commit will be written as the root commit. Even if there is only one parent, this should be an array.
- **author** (dict) – (optional), if omitted, GitHub will use the authenticated user’s credentials and the current time. Format: {'name': 'Committer Name', 'email': 'name@example.com', 'date': 'YYYY-MM-DDTHH:MM:SS+HH:00'}
- **committer** (dict) – (optional), if ommitted, GitHub will use the author parameters. Should be the same format as the author parameter.

Returns: Commit if successful, else None

create_fork (organization=None)
Create a fork of this repository.

Parameters **organization** (str) – (required), login for organization to create the fork under

Returns: Repository if successful, else None

create_hook (name, config, events=['push'], active=True)
Create a hook on this repository.

Parameters

- **name** (str) – (required), name of the hook
- **config** (dict) – (required), key-value pairs which act as settings for this hook
- **events** (list) – (optional), events the hook is triggered for
- **active** (bool) – (optional), whether the hook is actually triggered

Returns: Hook if successful, else None

create_issue (title, body=None, assignee=None, milestone=None, labels=None)
Creates an issue on this repository.

Parameters

- **title** (str) – (required), title of the issue
• **body** (*str*) – (optional), body of the issue
• **assignee** (*str*) – (optional), login of the user to assign the issue to
• **milestone** (*int*) – (optional), id number of the milestone to attribute this issue to (e.g. `m` is a `Milestone` object, `m.number` is what you pass here.)
• **labels** (*list of strings*) – (optional), labels to apply to this issue

**Returns** `Issue` if successful, else None

**create_key** *(title, key)*

Create a deploy key.

**Parameters**

• **title** (*str*) – (required), title of key
• **key** (*str*) – (required), key text

**Returns** `Key` if successful, else None

**create_label** *(name, color)*

Create a label for this repository.

**Parameters**

• **name** (*str*) – (required), name to give to the label
• **color** (*str*) – (required), value of the color to assign to the label

**Returns** `Label` if successful, else None

**create_milestone** *(title, state=None, description=None, due_on=None)*

Create a milestone for this repository.

**Parameters**

• **title** (*str*) – (required), title of the milestone
• **state** (*str*) – (optional), state of the milestone, accepted values: (`open`, `closed`), default: `open`
• **description** (*str*) – (optional), description of the milestone
• **due_on** (*str*) – (optional), ISO 8601 formatted due date

**Returns** `Milestone` if successful, else None

**create_pull** *(title, base, head, body=None)*

Create a pull request using commits from `head` and comparing against `base`.

**Parameters**

• **title** (*str*) – (required)
• **base** (*str*) – (required), e.g., `username:branch`, or a sha
• **head** (*str*) – (required), e.g., `master`, or a sha
• **body** (*str*) – (optional), markdown formatted description

**Returns** `PullRequest` if successful, else None

**create_pull_from_issue** *(issue, base, head)*

Create a pull request from issue #“issue”.

**Parameters**
• **issue** *(int)* – (required), issue number
• **base** *(str)* – (required), e.g., ‘username:branch’, or a sha
• **head** *(str)* – (required), e.g., ‘master’, or a sha

Returns **PullRequest** if successful, else None

**create_ref**(ref, sha)
Create a reference in this repository.

Parameters

• **ref** *(str)* – (required), fully qualified name of the reference, e.g. refs/heads/master. If it doesn’t start with refs and contain at least two slashes, GitHub’s API will reject it.
• **sha** *(str)* – (required), SHA1 value to set the reference to

Returns **Reference** if successful else None

**create_status**(sha, state, target_url='', description='')
Create a status object on a commit.

Parameters

• **sha** *(str)* – (required), SHA of the commit to create the status on
• **state** *(str)* – (required), state of the test; only the following are accepted: ‘pending’, ‘success’, ‘error’, ‘failure’
• **target_url** *(str)* – (optional), URL to associate with this status.
• **description** *(str)* – (optional), short description of the status

**create_tag**(tag, message, sha, obj_type, tagger, lightweight=False)
Create a tag in this repository.

Parameters

• **tag** *(str)* – (required), name of the tag
• **message** *(str)* – (required), tag message
• **sha** *(str)* – (required), SHA of the git object this is tagging
• **obj_type** *(str)* – (required), type of object being tagged, e.g., ‘commit’, ‘tree’, ‘blob’
• **tagger** *(dict)* – (required), containing the name, email of the tagger and the date it was tagged
• **lightweight** *(bool)* – (optional), if False, create an annotated tag, otherwise create a lightweight tag (a Reference).

Returns If lightweight == False: **Tag** if successful, else None. If lightweight == True: **Reference**

**create_tree**(tree, base_tree='')
Create a tree on this repository.

Parameters

• **tree** *(list)* – (required), specifies the tree structure. Format: [{'path': 'path/file', 'mode': 'filemode', 'type': 'blob or tree', 'sha': '44bfc6d...'}]
• **base_tree** *(str)* – (optional), SHA1 of the tree you want to update with new data

Returns **Tree** if successful, else None
created_at = None
datetime object representing when the Repository was created.

delete()
Delete this repository.

Returns bool – True if successful, False otherwise

delete_key(key_id)
Delete the key with the specified id from your deploy keys list.

Returns bool – True if successful, False otherwise
description = None
Description of the repository.
download(id_num)
Get a single download object by its id.

Warning: On 2012-03-11, GitHub will be deprecating the Downloads API. This method will no longer work.

Parameters id_num (int) – (required), id of the download

Returns Download if successful, else None
edit(name, description=None, homepage=None, private=None, has_issues=None, has_wiki=None, has_downloads=None, default_branch=None)
Edit this repository.

Parameters

• name (str) – (required), name of the repository

• description (str) – (optional), If not None, change the description for this repository. API default: None - leave value unchanged.

• homepage (str) – (optional), If not None, change the homepage for this repository. API default: None - leave value unchanged.

• private (bool) – (optional), If True, make the repository private. If False, make the repository public. API default: None - leave value unchanged.

• has_issues (bool) – (optional), If True, enable issues for this repository. If False, disable issues for this repository. API default: None - leave value unchanged.

• has_wiki (bool) – (optional), If True, enable the wiki for this repository. If False, disable the wiki for this repository. API default: None - leave value unchanged.

• has_downloads (bool) – (optional), If True, enable downloads for this repository. If False, disable downloads for this repository. API default: None - leave value unchanged.

• default_branch (str) – (optional), If not None, change the default branch for this repository. API default: None - leave value unchanged.

Returns bool – True if successful, False otherwise

fork = None
Is this repository a fork?

forks = None
The number of forks made of this repository.
classmethod from_json(json)
    Return an instance of cls formed from json.

full_name = None
    Full name as login/name

git_commit(sha)
    Get a single (git) commit.

        Parameters sha (str) – (required), sha of the commit
        Returns Commit if successful, otherwise None

git_url = None
    Plain git url for an anonymous clone.

has_downloads = None
    Whether or not this repository has downloads enabled

has_issues = None
    Whether or not this repository has an issue tracker

has_wiki = None
    Whether or not this repository has the wiki enabled

homepage = None
    URL of the home page for the project.

hook(id_num)
    Get a single hook.

        Parameters id_num (int) – (required), id of the hook
        Returns Hook if successful, else None

html_url = None
    URL of the project at GitHub.

id = None
    Unique id of the repository.

is_assignee(login)
    Check if the user is a possible assignee for an issue on this repository.

        Returns bool

is_collaborator(login)
    Check to see if login is a collaborator on this repository.

        Parameters login (str) – (required), login for the user
        Returns bool – True if successful, False otherwise

issue(number)
    Get the issue specified by number.

        Parameters number (int) – (required), number of the issue on this repository
        Returns Issue if successful, else None

iter_assignees(number=-1, etag=None)
    Iterate over all available assignees to which an issue may be assigned.

        Parameters
• **number** (*int*) – (optional), number of assignees to return. Default: -1 returns all available assignees

• **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** list of Users

**iter_branches** (*number=-1, etag=None*)
Iterate over the branches in this repository.

**Parameters**

• **number** (*int*) – (optional), number of branches to return. Default: -1 returns all branches

• **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** list of Branches

**iter_comments** (*number=-1, etag=None*)
Iterate over comments on all commits in the repository.

**Parameters**

• **number** (*int*) – (optional), number of comments to return. Default: -1 returns all comments

• **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** list of RepoComments

**iter_comments_on_commit** (**sha**, *number=1, etag=None*)
Iterate over comments for a single commit.

**Parameters**

• **sha** (*str*) – (required), sha of the commit to list comments on

• **number** (*int*) – (optional), number of comments to return. Default: -1 returns all comments

• **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** list of RepoComments

**iter_commits** (**sha=None, path=None, author=None, number=-1, etag=None**)  
Iterate over commits in this repository.

**Parameters**

• **sha** (*str*) – (optional), sha or branch to start listing commits from

• **path** (*str*) – (optional), commits containing this path will be listed

• **author** (*str*) – (optional), GitHub login, real name, or email to filter commits by (using commit author)

• **number** (*int*) – (optional), number of comments to return. Default: -1 returns all comments

• **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** list of RepoCommits

**iter_contributors** (**anon=False, number=-1, etag=None**)  
Iterate over the contributors to this repository.

**Parameters**
• anon (bool) – (optional), True lists anonymous contributors as well
• number (int) – (optional), number of contributors to return. Default: -1 returns all contributors
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns list of Users

iter_downloads (number=-1, etag=None)
Iterate over available downloads for this repository.

Parameters
• number (int) – (optional), number of downloads to return. Default: -1 returns all available downloads
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns list of Downloads

iter_events (number=-1, etag=None)
Iterate over events on this repository.

Parameters
• number (int) – (optional), number of events to return. Default: -1 returns all available events
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns list of Events

iter_forks (sort='', number=-1, etag=None)
Iterate over forks of this repository.

Parameters
• sort (str) – (optional), accepted values: (‘newest’, ‘oldest’, ‘watchers’), API default: ‘newest’
• number (int) – (optional), number of forks to return. Default: -1 returns all forks
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns list of Repository

iter_hooks (number=-1, etag=None)
Iterate over hooks registered on this repository.

Parameters
• number (int) – (optional), number of hooks to return. Default: -1 returns all hooks
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns list of Hooks

iter_issue_events (number=-1, etag=None)
Iterates over issue events on this repository.

Parameters
• **number** *(int)* – (optional), number of events to return. Default: -1 returns all available events

• **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

Returns generator of `IssueEvents`

**iter_issues** *(milestone=None, state=None, assignee=None, mentioned=None, labels=None, sort=None, direction=None, since=None, number=-1, etag=None)*

Iterate over issues on this repo based upon parameters passed.

Parameters

• **milestone** *(int)* – (optional), ‘none’, or ‘*’

• **state** *(str)* – (optional), accepted values: (‘open’, ‘closed’)

• **assignee** *(str)* – (optional), ‘none’, ‘*’, or login name

• **mentioned** *(str)* – (optional), user’s login name

• **labels** *(str)* – (optional), comma-separated list of labels, e.g. ‘bug,ui,@high’

• **sort** *(str)* – (optional), accepted values: (‘created’, ‘updated’, ‘comments’, ‘created’)

• **direction** *(str)* – (optional), accepted values: (‘asc’, ‘desc’)

• **since** *(str)* – (optional), ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ

• **number** *(int)* – (optional), Number of issues to return. By default all issues are returned

• **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

Returns list of `Issues`

**iter_keys** *(number=-1, etag=None)*

Iterates over deploy keys on this repository.

Parameters

• **number** *(int)* – (optional), number of keys to return. Default: -1 returns all available keys

• **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

Returns generator of `Keys`

**iter_labels** *(number=-1, etag=None)*

Iterates over labels on this repository.

Parameters

• **number** *(int)* – (optional), number of labels to return. Default: -1 returns all available labels

• **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

Returns generator of `Labels`

**iter_languages** *(number=-1, etag=None)*

Iterate over the programming languages used in the repository.

Parameters

• **number** *(int)* – (optional), number of languages to return. Default: -1 returns all used languages

• **etag** *(str)* – (optional), ETag from a previous request to the same endpoint
Returns  list of tuples

iter_milestones (state=None, sort=None, direction=None, number=-1, etag=None)
Iterates over the milestones on this repository.

Parameters

- **state** (str) – (optional), state of the milestones, accepted values: (‘open’, ‘closed’)
- **sort** (str) – (optional), how to sort the milestones, accepted values: (‘due_date’, ‘completeness’)
- **direction** (str) – (optional), direction to sort the milestones, accepted values: (‘asc’, ‘desc’)
- **number** (int) – (optional), number of milestones to return. Default: -1 returns all milestones
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns  generator of Milestones

iter_network_events (number=-1, etag=None)
Iterates over events on a network of repositories.

Parameters

- **number** (int) – (optional), number of events to return. Default: -1 returns all available events
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns  generator of Events

iter_notifications (all=False, participating=False, since='', number=-1, etag=None)
Iterates over the notifications for this repository.

Parameters

- **all** (bool) – (optional), show all notifications, including ones marked as read
- **participating** (bool) – (optional), show only the notifications the user is participating in directly
- **since** (str) – (optional), filters out any notifications updated before the given time. The time should be passed in as UTC in the ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ. Example: “2012-10-09T23:39:01Z”.
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns  generator of Thread

iter_pulls (state=None, number=-1, etag=None)
List pull requests on repository.

Parameters

- **state** (str) – (optional), accepted values: (‘open’, ‘closed’)
- **number** (int) – (optional), number of pulls to return. Default: -1 returns all available pull requests
- **etag** (str) – (optional), ETag from a previous request to the same endpoint

Returns  generator of PullRequests
**iter_refs** *(subspace='', number=-1, etag=None)*
Iterates over references for this repository.

**Parameters**
- **subspace** *(str)* – (optional), e.g. ‘tags’, ‘stashes’, ‘notes’
- **number** *(int)* – (optional), number of refs to return. Default: -1 returns all available refs
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **References**

**iter_stargazers** *(number=-1, etag=None)*
List users who have starred this repository.

**Parameters**
- **number** *(int)* – (optional), number of stargazers to return. Default: -1 returns all subscribers available
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Users**

**iter_statuses** *(sha, number=-1, etag=None)*
Iterates over the statuses for a specific SHA.

**Parameters**
- **sha** *(str)* – SHA of the commit to list the statuses of
- **number** *(int)* – (optional), return up to number statuses. Default: -1 returns all available statuses.
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Status**

**iter_subscribers** *(number=-1, etag=None)*
Iterates over users subscribed to this repository.

**Parameters**
- **number** *(int)* – (optional), number of subscribers to return. Default: -1 returns all subscribers available
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **User**

**iter_tags** *(number=-1, etag=None)*
Iterates over tags on this repository.

**Parameters**
- **number** *(int)* – (optional), return up to at most number tags. Default: -1 returns all available tags.
- **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **RepoTags**

**iter_teams** *(number=-1, etag=None)*
Iterates over teams with access to this repository.

**Parameters**
• **number** *(int)* – (optional), return up to number Teams. Default: -1 returns all Teams.
• **etag** *(str)* – (optional), ETag from a previous request to the same endpoint

Returns generator of **Teams**

**key**(id_num)
Get the specified deploy key.

Parameters  **id_num** *(int)* – (required), id of the key

Returns **Key** if successful, else None

**label**(name)
Get the label specified by name

Parameters  **name** *(str)* – (required), name of the label

Returns **Label** if successful, else None

**language** = None
Language property.

**mark_notifications**(last_read='')
Mark all notifications in this repository as read.

Parameters  **last_read** *(str)* – (optional), Describes the last point that notifications were checked. Anything updated since this time will not be updated. Default: Now. Expected in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ. Example: “2012-10-09T23:39:01Z”.

Returns **bool**

**master_branch** = None
default branch for the repository

**merge**(base, head, message=' ')
Perform a merge from head into base.

Parameters

• **base** *(str)* – (required), where you’re merging into
• **head** *(str)* – (required), where you’re merging from
• **message** *(str)* – (optional), message to be used for the commit

Returns **RepoCommit**

**milestone**(number)
Get the milestone indicated by number.

Parameters  **number** *(int)* – (required), unique id number of the milestone

Returns **Milestone**

**mirror_url** = None
Mirror property.

**name** = None
Name of the repository.

**open_issues** = None
Number of open issues on the repository.

**owner** = None
**User** object representing the
parent = None
    Parent of this fork, if it exists Repository

private = None
    Is this repository private?

pull_request (number)
    Get the pull request indicated by number.

        Parameters number (int) – (required), number of the pull request.

        Returns PullRequest

pushed_at = None
    datetime object representing the last time commits were pushed

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

readme ()
    Get the README for this repository.

        Returns Contents

ref (ref)
    Get a reference pointed to by ref.

    The most common will be branches and tags. For a branch, you must specify `heads/branchname` and for
    a tag, `tags/tagname`. Essentially, the system should return any reference you provide it in the namespace,
    including notes and stashes (provided they exist on the server).

        Parameters ref (str) – (required)

        Returns Reference

refresh (conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

        Parameters conditional (bool) – If True, then we will search for a stored header (`'Last-
        Modified'`, or `'ETag'`) on the object and send that as described in the Conditional Requests
        section of the docs

        Returns self

The reasoning for the return value is the following example:

repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None's and you would otherwise have to do:

repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

remove_collaborator (login)
    Remove collaborator login from the repository.

        Parameters login (str) – (required), login name of the collaborator

        Returns bool

set_subscription (subscribed, ignored)
    Set the user’s subscription for this repository

        Parameters
• **subscribed** (*bool*) – (required), determines if notifications should be received from this repository.

• **ignored** (*bool*) – (required), determines if notifications should be ignored from this repository.

**Returns**: `Subscription`<Subscription>

```python
size = None
Size of the repository.

source = None
Parent of this fork, if it exists. `Repository`

ssh_url = None
URL to clone the repository via SSH.

subscription()
Return subscription for this Repository.

**Returns**: `Subscription`

svn_url = None
If it exists, url to clone the repository via SVN.

tag(*sha*)
Get an annotated tag.

http://learn.github.com/p/tagging.html

**Parameters**: `sha` (*str*) – (required), sha of the object for this tag

**Returns**: `Tag`

to_json()
Return the json representing this object.

tree(*sha*)
Get a tree.

**Parameters**: `sha` (*str*) – (required), sha of the object for this tree

**Returns**: `Tree`

update_label(*name, color, new_name='')*
Update the label name.

**Parameters**

• `name` (*str*) – (required), name of the label

• `color` (*str*) – (required), color code

• `new_name` (*str*) – (optional), new name of the label

**Returns**: `bool`

updated_at = None
datetime object representing the last time the repository was watched.

watchers = None
Number of users watching the repository.

```
```
class github3.repos.Contents(content)

The Contents object. It holds the information concerning any content in a repository requested via the API.

content = None
Base64-encoded content of the file.

decoded = None
Decoded content of the file as a bytes object. If we try to decode to character set for you, we might encounter an exception which will prevent the object from being created. On python2 this is the same as a string, but on python3 you should call the decode method with the character set you wish to use, e.g., content.decoded.decode('utf-8').

encoding = None
Returns encoding used on the content.

git_url
API URL for this blob

html_url
URL pointing to the content on GitHub.

links = None
Dictionary of links

name = None
Name of the content.

path = None
Path to the content.

sha = None
SHA string.

size = None
Size of the content

type = None
Type of content.

class github3.repos.Download(download, session=None)

The Download object. It represents how GitHub sends information about files uploaded to the downloads section of a repository.

Warning: On 2013-03-11, this API will be deprecated by GitHub. There will also be a new version of github3.py to accompany this at that date.

content_type = None
Content type of the download.
**delete()**
Delete this download if authenticated

**description = None**
Description of the download.

**download_count = None**
How many times this particular file has been downloaded.

**html_url = None**
URL of the download at GitHub.

**id = None**
Unique id of the download on GitHub.

**name = None**
Name of the download.

**saveas(path='')**
Save this download to the path specified.

**Parameters**
- **path (str)** – (optional), if no path is specified, it will be saved in the current directory with the name specified by GitHub. It can take a file-like object as well

**Returns**
**bool**

**size = None**
Size of the download.

---

**class github3.repos.Hook(hook, session=None)**
The Hook object. This handles the information returned by GitHub about hooks set on a repository.

**active = None**
Whether or not this Hook is marked as active on GitHub

**config = None**
Dictionary containing the configuration for the Hook.

**created_at = None**
Datetime object representing the date the hook was created.

**delete()**
Delete this hook.

**Returns**
**bool**

**delete_subscription()**
Delete the user’s subscription to this repository.

**Returns**
**bool**

**edit(name, config, events=[], add_events=[], rm_events=[], active=True)**
Edit this hook.

**Parameters**
- **name (str)** – (required), name of the service being called
- **config (dict)** – (required), key-value pairs of settings for this hook
- **events (list)** – (optional), which events should this be triggered for
- **add_events (list)** – (optional), events to be added to the list of events that this hook triggers for

---

2.12. Repository
• **rm_events** *(list)* – (optional), events to be removed from the list of events that this hook triggers for
• **active** *(bool)* – (optional), should this event be active

**Returns** bool

```
events = None
Events which trigger the hook.

id = None
Unique id of the hook.

name = None
The name of the hook.

test()
Test this hook

Returns bool

updated_at = None
datetime object representing when this hook was last updated.
```

```python
class github3.repos.RepoTag(tag)
    The RepoTag object. This stores the information representing a tag that was created on a repository.
    
    commit = None
    Dictionary containing the SHA and URL of the commit.

    name = None
    Name of the tag.

    tarball_url = None
    URL for the GitHub generated tarball associated with the tag.

    zipball_url = None
    URL for the GitHub generated zipball associated with the tag.
```

More information about this class can be found in the official documentation about comments.

```python
class github3.repos.RepoComment(comment, session=None)
    The RepoComment object. This stores the information about a comment on a file in a repository.
    
    commit_id = None
    Commit id on which the comment was made.

    delete()
    Delete this comment.

    Returns bool

    edit(body)
    Edit this comment.

    Parameters body *(str)* – (required), new body of the comment, Markdown formatted

    Returns bool

    classmethod from_json(json)
    Return an instance of cls formed from json.
```
html_url = None
URL of the comment on GitHub.

line = None
The line number where the comment is located.

path = None
The path to the file where the comment was made.

position = None
The position in the diff where the comment was made.

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh (conditional=False)
Re-fetch the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self
The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

to_json()
Return the json representing this object.

update (body, sha, line, path, position)
Update this comment.

Parameters
• body (str) – (required)
• sha (str) – (required), sha id of the commit to comment on
• line (int) – (required), line number to comment on
• path (str) – (required), relative path of the file you’re commenting on
• position (int) – (required), line index in the diff to comment on

Returns bool

updated_at = None
datetime object representing when the comment was updated.

user = None
Login of the user who left the comment.

class github3.repos.RepoCommit (commit, session=None)
The RepoCommit object. This represents a commit as viewed by a Repository. This is different from a Commit object returned from the git data section.
additions = None
    The number of additions made in the commit.

author = None
    User who authored the commit.

commit = None
    Commit.

committer = None
    User who committed the commit.

deletions = None
    The number of deletions made in the commit.

diff()
    Return the diff

files = None
    The files that were modified by this commit.

patch()
    Return the patch

total = None
    Total number of changes in the files.

class github3.repos.Comparison(compare)
    The Comparison object. This encapsulates the information returned by GitHub comparing two commit objects in a repository.

    ahead_by = None
        Number of commits ahead by.

    base_commit = None
        RepoCommit object representing the base of

    behind_by = None
        Number of commits behind by.

    commits = None
        List of RepoCommit objects.

diff()
    Return the diff

diff_url = None
    URL to see the diff between the two commits.

files = None
    List of dicts describing the files modified.

html_url = None
    URL to view the comparison at GitHub

patch()
    Return the patch

patch_url = None
    Patch URL at GitHub for the comparison.
github3.py Documentation, Release 0.6.0.b

```python
permalink_url = None
    Permanent link to this comparison.

status = None
    Behind or ahead.

total_commits = None
    Number of commits difference in the comparison.
```

```python
class github3.repos.Status(status)
    The Status object. This represents information from the Repo Status API.

created_at = None
    datetime object representing the creation of the status object

creator = None
    User who created the object

description = None
    Short description of the Status

id = None
    GitHub ID for the status object

state = None
    State of the status, e.g., 'success', 'pending', 'failed', 'error'

target_url = None
    URL to view more information about the status

updated_at = None
    datetime object representing the last time the status was updated
```

2.13 Structures

2.13.1 Developed for github3.py

As of right now, there exists only one class in this section, and it is of only limited importance to users of github3.py. The GitHubIterator class is used to return the results of calls to almost all of the calls to iter_ methods on objects. When conditional refreshing was added to objects, there was a noticeable gap in having conditional calls to those iter_ methods. GitHub provides the proper headers on those calls, but there was no easy way to add that to what github3.py returned so it could be used properly. This was the best compromise - an object the behaves like an iterator regardless but can also be refreshed to get results since the last request conditionally.

2.13.2 Objects

```python
class github3.structs.GitHubIterator (count, url, cls, session, params=None, etag=None)
    The GitHubIterator class powers all of the iter_* methods.

cls = None
    Class being used to cast all items to

count = None
    Number of items left in the iterator
```
etag = None
The ETag Header value returned by GitHub

classmethod from_json(json)
Return an instance of cls formed from json.

headers = None
Headers generated for the GET request

original = None
Original number of items requested

params = None
Parameters of the query string

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

to_json()
Return the json representing this object.

url = None
URL the class used to make it’s first GET

2.14 User

This part of the documentation covers:

• User
• Key
• Plan

None of these objects should ever be instantiated by the user (developer).

When listing users, GitHub only sends a handful of the object’s attributes. To retrieve all of the object’s attributes, you must call the refresh() method. This unfortunately requires another call to the API, so use it sparingly if you have a low limit

2.14.1 User Modules

class github3.users.User (user, session=None)
The User object. This handles and structures information in the User section.

add_email_address (address)
Add the single email address to the authenticated user’s account.

Parameters address (str) – (required), email address to add

Returns list of email addresses

add_email_addresses (addresses=[])
Add the email addresses in addresses to the authenticated user’s account.

Parameters addresses (list) – (optional), email addresses to be added

Returns list of email addresses

delete_email_address (address)
Delete the email address from the user’s account.
Parameters address (str) – (required), email address to delete

Returns bool
delete_email_addresses (addresses=[])
Delete the email addresses in addresses from the authenticated user’s account.

Parameters addresses (list) – (optional), email addresses to be removed

Returns bool
disk_usage = None
How much disk consumed by the user
classmethod from_json(json)
Return an instance of cls formed from json.
gravatar_id = None
ID of the user’s image on Gravatar
hireable = None
True – for hire, False – not for hire
is_assignee_on (login, repository)
Checks if this user can be assigned to issues on login/repository.

Returns bool
iter_events (public=False, number=-1, etag=None)
Iterate over events performed by this user.

Parameters
• public (bool) – (optional), only list public events for the authenticated user
• number (int) – (optional), number of events to return. Default: -1 returns all available events.
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns list of Events
iter_followers (number=-1, etag=None)
Iterate over the followers of this user.

Parameters
• number (int) – (optional), number of followers to return. Default: -1 returns all available
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Users
iter_following (number=-1, etag=None)
Iterate over the users being followed by this user.

Parameters
• number (int) – (optional), number of users to return. Default: -1 returns all available users
• etag (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Users
iter_keys(number=-1, etag=None)
Iterate over the public keys of this user. New in version 0.5.

Parameters

- number (int) – (optional), number of keys to return. Default: -1 returns all available keys
- etag (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Keys

iter_org_events(org, number=-1, etag=None)
Iterate over events as they appear on the user’s organization dashboard. You must be authenticated to view this.

Parameters

- org (str) – (required), name of the organization
- number (int) – (optional), number of events to return. Default: -1 returns all available events
- etag (str) – (optional), ETag from a previous request to the same endpoint

Returns list of Events

iter_received_events(public=False, number=-1, etag=None)
Iterate over events that the user has received. If the user is the authenticated user, you will see private and public events, otherwise you will only see public events.

Parameters

- public (bool) – (optional), determines if the authenticated user sees both private and public or just public
- number (int) – (optional), number of events to return. Default: -1 returns all events available
- etag (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Events

iter_starred(sort=None, direction=None, number=-1, etag=None)
Iterate over repositories starred by this user. Changed in version 0.5: Added sort and direction parameters (optional) as per the change in GitHub’s API.

Parameters

- number (int) – (optional), number of starred repos to return. Default: -1, returns all available repos
- sort (str) – (optional), either ‘created’ (when the star was created) or ‘updated’ (when the repository was last pushed to)
- direction (str) – (optional), either ‘asc’ or ‘desc’. Default: ‘desc’
- etag (str) – (optional), ETag from a previous request to the same endpoint

Returns generator of Repository

iter_subscriptions(number=-1, etag=None)
Iterate over repositories subscribed to by this user.

Parameters

• **number** (*int*) – (optional), number of subscriptions to return. Default: -1, returns all available

• **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of **Repository**

```
owned_private_repos = None
```

Number of private repos owned by this user

```
plan = None
```

Which plan this user is on

```
public_gists = None
```

Number of public gists

**ratelimit_remaining**

Number of requests before GitHub imposes a ratelimit.

```
refresh (conditional=False)
```

Re-retrieve the information for this object and returns the refreshed instance.

**Parameters**

conditional (**bool**) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

**Returns** self

The reasoning for the return value is the following example:

```
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]
```

Without the return value, that would be an array of None’s and you would otherwise have to do:

```
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]
```

Which is really an anti-pattern. Changed in version 0.5.

```
to_json ()
```

Return the json representing this object.

```
total_private_gists = None
```

Number of private gists owned by this user

```
total_private_repos = None
```

Total number of private repos

**update** (**name=None**, **email=None**, **blog=None**, **company=None**, **location=None**, **hireable=False**, **bio=None**)

If authenticated as this user, update the information with the information provided in the parameters.

**Parameters**

• **name** (*str*) – e.g., ‘John Smith’, not login name

• **email** (*str*) – e.g., ‘john.smith@example.com’

• **blog** (*str*) – e.g., ‘http://www.example.com/jsmith/blog’

• **company** (*str*) –

• **location** (*str*) –

• **hireable** (**bool**) – defaults to False
• *bio* (*str*) – GitHub flavored markdown

**Returns** bool

```python
class github3.users.Key(key, session=None)
The Key object. Please see GitHub’s Key Documentation for more information.

def delete()
    Delete this Key

classmethod from_json(json)
    Return an instance of cls formed from json.

id = None
    The unique id of the key at GitHub

key = None
    The text of the actual key

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header ('Last-
                               Modified', or 'ETag') on the object and send that as described in the Conditional Requests
                               section of the docs

    **Returns** self

    The reasoning for the return value is the following example:
    repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

    Without the return value, that would be an array of None’s and you would otherwise have to do:
    repos = [r for i in g.iter_repos('kennethreitz')]
    [r.refresh() for r in repos]

    Which is really an anti-pattern. Changed in version 0.5.

title = None
    The title the user gave to the key

to_json()
    Return the json representing this object.

update(title, key)
    Update this key.

    Parameters

    • *title* (*str*) – (required), title of the key

    • *key* (*str*) – (required), text of the key file

    **Returns** bool
```

```python
class github3.users.Plan(plan)
The Plan object. This makes interacting with the plan information about a user easier. Please see GitHub’s
Authenticated User documentation for more specifics.
```
collaborators = None
    Number of collaborators

classmethod from_json(json)
    Return an instance of cls formed from json.

is_free()
    Checks if this is a free plan.

    Returns bool

name = None
    Name of the plan

private_repos = None
    Number of private repos

space = None
    Space allowed

to_json()
    Return the json representing this object.

## 2.15 Internals

For objects you’re not likely to see in practice. This is useful if you ever feel the need to contribute to the project.

### 2.15.1 Models

This part of the documentation covers a lot of lower-level objects that are never directly seen or used by the user (developer). They are documented for future developers of this library.

**Objects**

class github3.models.GitHubCore(json, ses=None)
    The GitHubCore object. This class provides some basic attributes to other classes that are very useful to have.

classmethod from_json(json)
    Return an instance of cls formed from json.

ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.

    Parameters conditional (bool) – If True, then we will search for a stored header ('Last-Modified', or 'ETag') on the object and send that as described in the Conditional Requests section of the docs

    Returns self

    The reasoning for the return value is the following example:

    repos = [r.refresh() for r in g.iter_repos('kennethreitz')]
Without the return value, that would be an array of `None`'s and you would otherwise have to do:

```python
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]
```

Which is really an anti-pattern. Changed in version 0.5.

```python
to_json()
    Return the json representing this object.
```

class github3.models.BaseAccount(acct, session)
The `BaseAccount` object. This is used to do the heavy lifting for Organization and User objects.

**Properties**

- `avatar_url`: URL of the avatar at gravatar
- `bio`: Markdown formatted biography
- `blog`: URL of the blog
- `company`: Name of the company
- `created_at`: Datetime object representing the date the account was created
- `email`: E-mail address of the user/org
- `followers`: Number of followers
- `following`: Number of people the user is following

```python
classmethod from_json(json)
    Return an instance of cls formed from json.
```

- `html_url`: URL of the user/org’s profile
- `id`: Unique ID of the account
- `location`: Location of the user/org
- `login`: Login name of the user/org
- `name`: Real name of the user/org
- `public_repos`: Number of public repos owned by the user/org

```python
ratelimit_remaining
    Number of requests before GitHub imposes a ratelimit.
```

```python
refresh(conditional=False)
    Re-retrieve the information for this object and returns the refreshed instance.
```
Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self

The reasoning for the return value is the following example:

```python
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]
```

Without the return value, that would be an array of `None`’s and you would otherwise have to do:

```python
repos = [r for i in g.iter_repos('kennethreitz')] [r.refresh() for r in repos]
```

Which is really an anti-pattern. Changed in version 0.5.

to_json()

Return the json representing this object.

type = None

Tells you what type of account this is

class github3.models.BaseComment (comment, session)

The BaseComment object. A basic class for Gist, Issue and Pull Request Comments.

body = None

Body of the comment. (As written by the commenter)

body_html = None

Body of the comment formatted as html.

body_text = None

Body of the comment formatted as plain-text. (Stripped of markdown,

created_at = None

datetime object representing when the comment was created.

delete ()

Delete this comment.

Returns bool

edit (body)

Edit this comment.

Parameters body (str) – (required), new body of the comment, Markdown formatted

Returns bool

classmethod from_json (json)

Return an instance of cls formed from json.

html_url = None

The url of this comment at GitHub

id = None

Unique ID of the comment.

pull_request_url = None

The url of the pull request, if it exists
ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self
The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]

Which is really an anti-pattern. Changed in version 0.5.

to_json()
Return the json representing this object.

updated_at = None
datetime object representing when the comment was updated.

class github3.models.BaseCommit (commit, session)
The BaseCommit object. This serves as the base for the various types of commit objects returned by the API.

classmethod from_json (json)
Return an instance of cls formed from json.

message = None
Commit message

parents = None
List of parents to this commit.

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

refresh(conditional=False)
Re-retrieve the information for this object and returns the refreshed instance.

Parameters conditional (bool) – If True, then we will search for a stored header (‘Last-Modified’, or ‘ETag’) on the object and send that as described in the Conditional Requests section of the docs

Returns self
The reasoning for the return value is the following example:
repos = [r.refresh() for r in g.iter_repos('kennethreitz')]

Without the return value, that would be an array of None’s and you would otherwise have to do:
repos = [r for i in g.iter_repos('kennethreitz')]
[r.refresh() for r in repos]
Which is really an anti-pattern. Changed in version 0.5.

```python
sha = None
    SHA of this commit.

to_json()
    Return the json representing this object.
```

## 2.15.2 Decorators

This part of the documentation covers the decorators module which contains all of the decorators used in github3.py.

### Contents

```python
github3.decorators.requires_auth(x)
```
$ pip install github3.py
# OR:
$ git clone git://github.com/sigmavirus24/github3.py.git github3.py
$ cd github3.py
$ python setup.py install

3.1 Dependencies

- requests by Kenneth Reitz
CHAPTER FOUR

CONTRIBUTING

I’m maintaining two public copies of the project. The first can be found on GitHub and the second on BitBucket. I would prefer pull requests to take place on GitHub, but feel free to do them via BitBucket. Please make sure to add yourself to the list of contributors in AUTHORS.rst, especially if you’re going to be working on the list below.

4.1 Contributor Friendly Work

In order of importance:

Documentation

I know I’m not the best at writing documentation so if you want to clarify or correct something, please do so.

Examples

Have a clever example that takes advantage of github3.py? Feel free to share it.

4.2 Running the Unittests

```bash
mkdir -p /path/to/virtualenv/github3.py
cd /path/to/virtualenv/github3.py
virtualenv .
cd /path/to/github3.py_repo/requirements.txt
pip install -r requirements.txt
# Or you could run make test-deps
make tests
```
CONTACT

- Twitter: @sigmavirus24
- Private email: graffatcolmingov [at] gmail
- Mailing list: github3.py [at] librelist.com
HISTORY/CHANGELOG

6.1 0.6.0: 2013-xx-xx

• Add sort and order parameters to github3.GitHub.search_users and github3.GitHub.search_repos.
• Add iter_commits to github3.gists.Gist as a means of re-requesting just the history from GitHub and iterating over it.
• Add minimal logging (e.g., logging.getLogger('github3'))
• Re-organize the library.
• Calling refresh(True) on a github3.structs.GitHubIterator actually works as expected now.
• API iter_ methods now accept the etag argument as the GitHub.iter_ methods do.
• github3.octocat and github3.github.GitHub.octocat both support sending messages to make the Octocat say things. (Think cowsay)
• Remove vendored dependency of PySO8601.

6.2 0.5.3: 2013-03-19

• Add missing optional parameter to Repository.contents. Thanks @tpetr

6.3 0.5.2: 2013-03-02

• Stop trying to decode the byte strings returned by b64decode. Fixes #72

6.4 0.5.1: 2013-02-21

• Hot fix an issue when a user doesn’t have a real name set
6.5 0.5: 2013-02-16

- 100% (mock) test coverage
- Add support for the announced meta endpoint.
- Add support for conditional refreshing, e.g.,

```python
import github3

u = github3.user('sigmavirus24')

# some time later

u.refresh()  # Will ALWAYS send a GET request and lower your ratelimit
u.refresh(True)  # Will send the GET with a header such that if nothing
# has changed, it will not count against your ratelimit
# otherwise you’ll get the updated user object.
```

- Add support for conditional iterables. What this means is that you can do:

```python
import github3

i = github3.iter_all_repos(10)

for repo in i:
    # do stuff

i = github3.iter_all_repos(10, etag=i.etag)
```

And the second call will only give you the new repositories since the last request. This mimics behavior in pengwynn/octokit

- Add support for sortable stars.
- In github3.users.User, iter_keys now allows you to iterate over any user’s keys. No name is returned for each key. This is the equivalent of visiting: github.com/:user.keys
- In github3.repos.Repository, pubsubhubbub has been removed. Use github3.github.Github.pubsubhubbub instead
- In github3.api, iter_repo_issues’s signature has been corrected.
- Remove list_{labels, comments, events} methods from github3.issues.Issue
- Remove list_{comments, commits, files} methods from github3.pulls.PullRequest
- In github3.gists.Gist:
  - the user attribute was changed by GitHub and is now the owner attribute
  - the public attribute and the is_public method return the same information. The method will be removed in the next version.
  - the is_starred method now requires authentication
  - the default refresh method is no longer over-ridden. In a change made in before, a generic refresh method was added to most objects. This was overridden in the Gist object and would cause otherwise unexpected results.
- github3.events.Event.is_public() and github3.events.Event.public now return the same information. In the next version, the former will be removed.
• In github3.issues.Issue
  – add_labels now returns the list of Labels on the issue instead of a boolean.
  – remove_label now returns a boolean.
  – remove_all_labels and replace_labels now return lists. The former should return an empty list on a successful call. The latter should return a list of github3.issues.Label objects.

• Now we won’t get spurious GitHubErrors on 404s, only on other expected errors whilst accessing the json in a response. All methods that return an object can now actually return None if it gets a 404 instead of just raising an exception. (Inspired by #49)

• GitHubStatus API now works.

### 6.6 0.4: 2013-01-16

• In github3.legacy.LegacyRepo
  – has_{downloads,issues,wiki} are now attributes.
  – is_private() and the private attribute return the same thing is_private() will be deprecated in the next release.

• In github3.repos.Repository
  – is_fork() is now deprecated in favor of the fork attribute
  – is_private() is now deprecated in favor of the private attribute

• In github3.repos.Hook
  – is_active() is now deprecated in favor of the active attribute

• In github3.pulls.PullRequest
  – is_mergeable() is now deprecated in favor of the mergeable attribute

• In github3.notifications.Thread
  – is_unread() is now deprecated in favor of the unread

• pubsubhubbub() is now present on the GitHub object and will be removed from the Repository object in the next release

• 70% test coverage

### 6.7 0.3: 2013-01-01

• In github3.repos.Repository
  – is_fork() and fork return the same thing
  – is_private() and private return the same thing as well
  – has_downloads, has_issues, has_wiki are now straight attributes

• In github3.repos.Hook
  – is_active() and active return the same value

• In github3.pulls.PullRequest
– is_mergeable() and mergeable are now the same
– repository now returns a tuple of the login and name of the repository it belongs to

• In github3notifications.Thread
  – is_unread() and unread are now the same

• In github3.gists
  – GistFile.filename and GistFile.name return the same information
  – Gist.history now lists the history of the gist
  – GistHistory is an object representing one commit or version of the history
  – You can retrieve gists at a specific version with GistHistory.get_gist()

• github3orgs.Organization.iter_repos now accepts all types
• list_* methods on Organization objects that were missed are now deleted
• Some objects now have __str__ methods. You can now do things like:

```python
import github3
u = github3.user('sigmavirus24')
r = github3.repository(u, 'github3.py')
```

And

```python
import github3
r = github3.repository('sigmavirus24', 'github3.py')
```

template = """Some kind of template where you mention this repository
[0]"""

```python
print(template.format(r))
# Some kind of template where you mention this repository
# sigmavirus24/github3.py
```

Current list of objects with this feature:
– github3users.User (uses the login name)
– github3users.Key (uses the key text)
– github3users.Repository (uses the login/name pair)
– github3users.RepoTag (uses the tag name)
– github3users.Contents (uses the decoded content)

• 60% test coverage with mock
• Upgrade to requests 1.0.x

6.8 0.2: 2012-11-21

• MAJOR API CHANGES:
  – GitHub.iter_subscribed -> GitHub.iter_subscriptions
Broken list_* functions in github3.api have been renamed to the correct iter_* methods on GitHub.

Removed list_* functions from Repository, Gist, Organization, and User objects

- Added zen of GitHub method.
- More tests
  - Changed the way Repository.edit works courtesy of Kristian Glass (@doismellburning)
  - Changed Repository.contents behaviour when acting on a 404.
- 50% test coverage via mock tests

### 6.9 0.1: 2012-11-13

- Add API for GitHub Enterprise customers.

### 6.10 0.1b2: 2012-11-10

- Handle 500 errors better, courtesy of Kristian Glass (@doismellburning)
- Handle sending json with % symbols better, courtesy of Kristian Glass
- Correctly handle non-GitHub committers and authors courtesy of Paul Swartz (@paulswartz)
- Correctly display method signatures in documentation courtesy of (@seveas)

### 6.11 0.1b1: 2012-10-31

- unit tests implemented using mock instead of hitting the GitHub API (#37)
- removed list_* functions from GitHub object
- Notifications API coverage

### 6.12 0.1b0: 2012-10-06

- Support for the complete GitHub API (accomplished)
  - Now also includes the Statuses API
  - Also covers the auto_init parameters to the Repository creation methodology
  - Limited implementation of iterators in the place of list functions.
- 98% coverage by unit tests
g

- `github3`, 22
- `github3.api`, 13
- `github3.auths`, 19
- `github3.decorators`, 99
- `github3.events`, 21
- `github3.gists`, 22
- `github3.git`, 24
- `github3.github`, 30
- `github3.issues`, 43
- `github3.legacy`, 49
- `github3.models`, 95
- `github3.orgs`, 57
- `github3.pulls`, 63
- `github3.repos`, 69
- `github3.structs`, 89
- `github3.users`, 90