

---

# **github3.py Documentation**

*Release 1.1.0*

**Ian Cordasco**

**May 25, 2018**



<b>1</b>	<b>Using Two Factor Authentication with github3.py</b>	<b>3</b>
<b>2</b>	<b>Using Tokens for Your Projects</b>	<b>5</b>
2.1	Requesting a token . . . . .	5
<b>3</b>	<b>Gist Code Examples</b>	<b>7</b>
3.1	Listing gists after authenticating . . . . .	7
3.2	Creating a gist after authenticating . . . . .	7
3.3	Creating an anonymous gist . . . . .	7
<b>4</b>	<b>Git Code Examples</b>	<b>9</b>
4.1	Creating a Blob Object . . . . .	9
4.2	Creating a Tag Object . . . . .	9
<b>5</b>	<b>GitHub Examples</b>	<b>11</b>
5.1	Assumptions . . . . .	11
5.2	Adding a new key to your account . . . . .	12
5.3	Deleting the key we just created . . . . .	12
5.4	Creating a new repository . . . . .	13
5.5	Create a commit to change an existing file . . . . .	13
5.6	Follow another user on GitHub . . . . .	13
5.7	Changing your user information . . . . .	13
<b>6</b>	<b>Issue Code Examples</b>	<b>15</b>
6.1	Administering Issues . . . . .	15
6.2	Example issue to comment on . . . . .	15
6.3	Importing an issue . . . . .	16
<b>7</b>	<b>Taking Advantage of GitHubIterator</b>	<b>17</b>
7.1	The First Approach . . . . .	17
7.2	The Second Approach . . . . .	18
<b>8</b>	<b>Using Logging with github3.py</b>	<b>19</b>
<b>9</b>	<b>A Conversation With Octocat</b>	<b>21</b>
<b>10</b>	<b>Installation</b>	<b>23</b>

<b>11 User Guide</b>	<b>25</b>
11.1 User Guide for github3.py . . . . .	25
<b>12 API Reference Documentation</b>	<b>29</b>
12.1 API Reference . . . . .	29
<b>13 Version History</b>	<b>241</b>
13.1 Release Notes and History . . . . .	241
<b>14 Contributing</b>	<b>259</b>
14.1 Contributor Friendly Work . . . . .	259
14.2 Running the Unittests . . . . .	259
<b>15 Contact</b>	<b>265</b>
<b>Python Module Index</b>	<b>267</b>

Release v1.1.0.

github3.py is 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. As an example, let's get information about a user:

```
from github3 import login

gh = login('sigmavirus24', password='<password>')

sigmavirus24 = gh.me()
# <AuthenticatedUser [sigmavirus24:Ian Stapleton Cordasco]>

print(sigmavirus24.name)
# Ian Stapleton Cordasco
print(sigmavirus24.login)
# sigmavirus24
print(sigmavirus24.followers_count)
# 4

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

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

print(kennethreitz.name)
print(kennethreitz.login)
print(kennethreitz.followers_count)

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

There are several examples of different aspects of using github3.py



---

## Using Two Factor Authentication with github3.py

---

GitHub recently added support for Two Factor Authentication to `github.com` and shortly thereafter added support for it on `api.github.com`. In version 0.8, `github3.py` also added support for it and you can use it right now.

To use Two Factor Authentication, you must define your own function that will return your one time authentication code. You then provide that function when logging in with `github3.py`.

For example:

```
import github3

try:
    # Python 2
    prompt = raw_input
except NameError:
    # Python 3
    prompt = input

def my_two_factor_function():
    code = ''
    while not code:
        # The user could accidentally press Enter before being ready,
        # let's protect them from doing that.
        code = prompt('Enter 2FA code: ')
    return code

g = github3.login('sigmavirus24', 'my_password',
                 two_factor_callback=my_two_factor_function)
```

Then each time the API tells `github3.py` it requires a Two Factor Authentication code, `github3.py` will call `my_two_factor_function` which prompts you for it.





---

## 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.
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 **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`.

### 2.1 Requesting a token

If you're not doing a web application, you are more than welcome to use `github3.py` (otherwise work with `redirects`). Let's say your application needs access to public and private repositories, and the users but not to gists. Your `scopes` should be `['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.

```
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'
```

(continues on next page)

(continued from previous page)

```
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 + '\n')
    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

token = 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
```

Examples with Gists

### 3.1 Listing gists after authenticating

```
from github3 import login

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

### 3.2 Creating a gist after authenticating

```
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)
```

### 3.3 Creating an anonymous gist

```
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)
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)
```

In the above examples 'spam.txt' is the file name. GitHub will auto-detect file type based on extension provided. 'What... is the air-speed velocity of an unladen swallow?' is the file's content or body. 'Answer this to cross the bridge' is the gists's description. While required by github3.py, it is allowed to be empty, e.g., '' is accepted by GitHub.

Note that anonymous gists are always public.

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.

## 4.1 Creating a Blob Object

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

```
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'VGvzdGluZyBibG9iIGNyZWf0aW9u\n'
blob.decoded
# u'Testing blob creation'
blob.encoding
# u'base64'
```

## 4.2 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')
```

(continues on next page)

(continued from previous page)

```
tag
# <Tag [cdba84b4fede2c69cb1ee246b33f49f19475abfa]>
tag.object.sha
# u'24ea44d302c6394a0372dcde8fd8aed899c0034b'
tag.object.type
# u'commit'
```

Examples using the *GitHub* object.

## 5.1 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 {}: '.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:

```
sigma = 'sigmavirus24'  
github3 = 'github3.py'  
todopy = 'Todo.txt-python'  
kr = 'kennethreitz'  
requests = 'requests'
```

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

## 5.2 Adding a new key to your account

```
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.')
```

## 5.3 Deleting the key we just created

Assuming we still have key from the previous example:

```
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:

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



## 5.4 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))
```

## 5.5 Create a commit to change an existing file

```
repo.contents('/README.md').update('commit message', 'file content'.encode('utf-8'))
```

## 5.6 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)
```

## 5.7 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:

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

---

## Issue Code Examples

---

Examples using Issues

### 6.1 Administering 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`.

```
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 + '\n-----\n**Update:** Text to append')
```

#### 6.1.1 Closing and Commenting on Issues

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

### 6.2 Example issue to comment on

If you would like to test the above, see [issue #108](#). Just follow the code there and fill in your username, password (or token), and comment message. Then run the script and watch as the issue opens in your browser focusing on the comment **you** just created.

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

```
import webbrowser
import tempfile
import github3

template = """<html><head></head><body>{0}</body></html>"""

i = github3.issue('kennethreitz', 'requests', 868)

with tempfile.NamedTemporaryFile() as tmpfd:
    tmpfd.write(template.format(i.body_html))
    webbrowser.open('file://' + tmpfd.name)
```

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

```
import github3
import textwrap

i = github3.issue('kennethreitz', 'requests', 868)
for line in textwrap.wrap(i.body_text, 78, replace_whitespace=False):
    print line
```

## 6.3 Importing an issue

Not only can you create new issues, but you can import existing ones. When importing, you preserve the timestamp creation date; you can preserve the timestamp(s) for comment(s) too.

```
import github3
gh = github3.login(token=token)
issue = {
    'title': 'Documentation issue',
    'body': 'Missing links in index.html',
    'created_at': '2011-03-11T17:00:40Z'
}

repository = gh.repository(user, repo)
repository.import_issue(**issue)
```

### 6.3.1 Status of imported issue

Here's how to check the status of the imported issue.

```
import github3
issue = repository.imported_issue(issue_num)
print issue.status
```

---

## 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:

```
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:

```
i = g.iter_all_users():

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

### 7.1 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:

```
import time

while True:
    i.refresh(True)
    for u in i:
        add_user_to_database(u)
```

(continues on next page)

(continued from previous page)

```
time.sleep(120) # Sleep for 2 minutes
```

## 7.2 The Second Approach

```
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.

---

## 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.

```
import github3
import logging

# Set up a file to have all the logs written to
file_handler = logging.FileHandler('github_script.log')

# Send the logs to stderr as well
stream_handler = logging.StreamHandler()

# Format the log output and include the log level's name and the time it was
# generated
formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)s')

# Use that Formatter on both handlers
file_handler.setFormatter(formatter)
stream_handler.setFormatter(formatter)

# Get the logger used by github3.py internally by referencing its name
# directly
logger = logging.getLogger('github3')
# Add the handlers to it
logger.addHandler(file_handler)
logger.addHandler(stream_handler)
# Set the level which determines what you see
logger.setLevel(logging.DEBUG)

# Make a library call and see the information posted
r = github3.repository('sigmavirus24', 'github3.py')
print('{0} - {0.html_url}'.format(r))
```

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:

```
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

```
import github3

print("Hey Octocat")
print(github3.octocat("Hey Ian"))
print("What do you think about github3.py?")
print(github3.octocat("github3.py rocks!"))
print("Thanks Octocat, that means a lot coming from you.")
print("FIN.")
print("""Epilog:
    The preceding conversation was entirely fictional. If you didn't realize
    that, you need to get out more.
""")
```

### What you should see

```
Hey Octocat

      MMM.          .MMM
      MMMMMMMMMMMMMMMMMMMMM
      MMMMMMMMMMMMMMMMMMMMM
      MMMMMMMMMMMMMMMMMMMMM | _____ |
      MMMMMMMMMMMMMMMMMMMMM | Hey Ian |
      MMMMMMMMMMMMMMMMMMMMM |_   _____|
      MMMM::- -:::~::~:~ -:::MMMM | /
      MM~::~:~ ~:::~::~:~ ~::~:MM
.. MMMMM:::~ .:::~::~:~ .:::MMMM ..
      .MM:::~ ._. .:::~MM.
      MMMM;:::~;MMMM
-MM      MMMMMMMM
^  M+      MMMMMMMM
      MMMMMMMM MM MM MM
      MM MM MM MM
      MM MM MM MM
      . ~ ~MM~MM~MM~MM~MM~ .
```

(continues on next page)

(continued from previous page)

```

~~~~MM:~MM~~~MM~:MM~~~~
~~~~~=====~~~~~
~~~~~=====~~~~~
      :~=====~~~~~

```

What do you think about github3.py?

```

      MMM.          .MMM
      MMMMMMMMMMMMMMMMMMM
      MMMMMMMMMMMMMMMMMMM
      MMMMMMMMMMMMMMMMMMM | _____ |
      MMMMMMMMMMMMMMMMMMM | github3.py rocks! |
      MMMMMMMMMMMMMMMMMMM |_ _____|
      MMMM::- -:::~::~- -:::MMMM | /
      MM~::~ ~:::~::~ ~::~MM
.. MMMMM:.. .:::~::~. .:::MMMM ..
      .MM:~::~ ._. :~::~MM.
      MMMM;~::~;MMMM
-MM      MMMMMMMM
^ M+      MMMMMMMM
      MMMMMMM MM MM MM
      MM MM MM MM
      MM MM MM MM
      .~MM~MM~MM~MM~.
      ~~~~MM:~MM~~~MM~:MM~~~~
      ~~~~~=====~~~~~
      ~~~~~=====~~~~~
      :~=====~~~~~

```

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.)

## CHAPTER 10

---

### Installation

---

```
$ pip install github3.py
```



### 11.1 User Guide for github3.py

This section of our documentation is intended to guide you, the user, through various ways of using the library and to introduce you to some high-level concepts in the library.

#### 11.1.1 Getting Started

This chapter in our documentation will teach you how to get started using github3.py after you've installed the library.

##### Using the library

To get started using the library, it's important to note that the module that is provided by this library is called `github3`. To use it you can run:

```
import github3
```

where necessary.

##### Logging into GitHub using github3.py

Once you've imported the module, you can get started using the API. It's recommended that you authenticate with GitHub to avoid running into [their ratelimits](#). To do so you have a few options.

First, you can use your username and password. We advise you not to type your password into your shell or python console directly as others can view that after the fact. For the sake of an example, let's assume that you have two variables bound as `username` and `password` that contain your username and password. You can then do:

```
import github3

github = github3.login(username=username, password=password)
```

Second, you can [generate an access token](#) and use that. Let's presume you have a variable bound as `token` that contains your access token.

```
import github3

github = github3.login(token=token)
```

Third, if you're using a GitHub Enterprise installation you can use similar methods above, but you'll need to use `enterprise_login()`, e.g.,

```
import github3

githubent = github3.enterprise_login(
    url='https://github.myenterprise.example.com',
    username=username,
    password=password,
)

githubent = github3.enterprise_login(
    url='https://github.myenterprise.example.com',
    token=token,
)
```

### Two-Factor Authentication and github3.py

GitHub has long supported the use of a second-factor authentication (a.k.a, 2FA) mechanism for logging in. This provides some extra security, especially around administrative actions on the website. If you choose to login with simply your username and password and you have to provide github3.py with a mechanism for obtaining your token and providing it to Github.

An example mechanism is as follows:

```
# This assumes Python 3
import github3

def second_factor_retrieval():
    """Provide a way to retrieve the code from the user."""
    code = ''
    while not code:
        code = input('Enter 2FA code: ')
    return code

github = github3.login(username, password,
                      two_factor_callback=second_factor_retrieval)
```

This means that for every API call made, GitHub will force us to prompt you for a new 2FA code. This is obviously not ideal. In those situations, you almost certainly want to obtain an access token.

### 11.1.2 Using the Repository APIs

Now that we have [learned](#) how to set up a client for use with our APIs, let's begin to review how github3.py implements the [Repositories API](#).

## Retrieving Repositories

Once you've *logged in* you will have an instance of *GitHub* or *GitHubEnterprise*. Let's assume either one is bound to a variable called `github`. To retrieve a single *repository* that we know the owner and name of, we would do the following:

```
repository = github.repository(owner, repository_name)
```

For example, let's retrieve the repository of the `uritemplate` package that `github3.py` relies on:

```
uritemplate = github.repository('python-hyper', 'uritemplate')
```

It's also possible for us to retrieve multiple repositories owned by the same user or organization:

```
for short_repository in github.repositories_by('python-hyper'):
    ...
```

When listing repositories, like listing other objects, the GitHub API doesn't return the full representation of the object. In this case, `github3.py` returns a different object to represent a *short repository*. This object has fewer attributes, but can be converted into a full repository like so:

```
for short_repository in github.repositories_by('python-hyper'):
    full_repository = short_repository.refresh()
```

We now have two separate objects for the repository based on how GitHub represents them. Both objects have the same methods attached to them. There's just a different set of attributes on each.

## Interacting with Repositories

Repositories are central to many things in GitHub as well as in the API and as result they have many attributes and methods. It's possible to list branches, collaborators, commits, contributors, deployments, forks, issues, projects, pull requests, refs, and more.

For example, we could build a tiny function that checks if a contributor has deleted their fork:

```
uritemplate = github.repository('python-hyper', 'uritemplate')
contributors_without_forks = (set(uritemplate.contributors()) -
                             set(fork.owner for fork in uritemplate.forks()))
print(f'The following contributors deleted their forks of {uritemplate!r}')
for contributor in sorted(contributors_without_forks, key=lambda c: c.login):
    print(f' * {contributor.login}')
```

The output should look like

```
The following contributors deleted their forks of <Repository [python-hyper/
->uritemplate]>
 * eugene-eeo
 * jpotts18
 * sigmavirus24
 * thierryba
```





## 12.1 API Reference

### 12.1.1 Anonymous Functional 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.

#### 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 may use `github3.login()`.

`github3.login(username=None, password=None, token=None, two_factor_callback=None)`  
Construct and return an authenticated GitHub session.

---

**Note:** To allow you to specify either a username and password combination or a token, none of the parameters are required. If you provide none of them, you will receive `None`.

---

#### Parameters

- **username** (*str*) – login name
- **password** (*str*) – password for the login
- **token** (*str*) – OAuth token
- **two\_factor\_callback** (*func*) – (optional), function you implement to provide the Two Factor Authentication code to GitHub when necessary

**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 also create a new `GitHub` object, e.g.,

```
from github3 import GitHub

gh = GitHub()
```

Or you can use the following functions:

### Anonymous Functions

`github3.authorize(username, password, scopes, note="", note_url="", client_id="", client_secret="", two_factor_callback=None, github=None)`

Obtain an authorization token for the GitHub API.

#### Parameters

- **username** (*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
- **two\_factor\_callback** (*func*) – (optional), function to call when a Two-Factor Authentication code needs to be provided by the user.
- **github** (`GitHub`) – (optional), GitHub (or GitHubEnterprise) object for login.

**Returns** Authorization

### Deprecated Functions

**Warning:** Due to GitHub's anonymous ratelimits, it's strongly advised that you don't use these functions.

`github3.create_gist(description, files)`

Create 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)`

Retrieve the gist identified by `id_num`.

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

**Returns** `Gist`

`github3.gitignore_template` (*language*)

Return the template for language.

**Returns** `str`

`github3.gitignore_templates` ()

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

Iterate over issues on `owner/repository`.

Changed in version 0.9.0: The `state` parameter now accepts ‘all’ in addition to ‘open’ and ‘closed’.

• **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: (‘all’, ‘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** (*datetime or string*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, 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 `ShortIssues`

`github3.all_repositories` (*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*

`github3.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*

`github3.all_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 *Event*

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

List the followers of `username`.

**Parameters**

- **username** (*str*) – (required), login of the person to list the followers of
- **number** (*int*) – (optional), number of followers to return, Default: -1, return all of them
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of *User*

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

List the people `username` follows.

**Parameters**

- **username** (*str*) – (required), login of the user
- **number** (*int*) – (optional), number of users being followed by `username` to return. Default: -1, return all of them
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of *User*

`github3.public_gists(number=-1, etag=None)`

Iterate over all public gists.

New in version 1.0: This was split from `github3.iter_gists` before 1.0.

**Parameters**

- **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 *Gist*

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

Iterate over gists created by the provided username.

#### Parameters

- **username** (*str*) – (required), 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 `Gist`

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

List the organizations with `username` as a member.

#### 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 `ShortOrganization`

`github3.repositories_by` (*username*, *type=None*, *sort=None*, *direction=None*, *number=-1*, *etag=None*)

List public repositories for the specified `username`.

New in version 0.6.

---

**Note:** This replaces `github3.iter_repos`

---

#### Parameters

- **username** (*str*) – (required)
- **type** (*str*) – (optional), accepted values: ('all', 'owner', '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.starred_by` (*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.subscriptions_for(username, number=-1, etag=None)`  
Iterate over repositories subscribed to by `username`.

**Parameters**

- **username** (*str*) – 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

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

`github3.octocat(say=None)`  
Return 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(name)`  
Return an `Organization` object for the login name.

**Parameters** `username` (*str*) – (required), login name of the org

**Returns** the organization

**Return type** `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.rate_limit()`  
Return a dictionary with information from `/rate_limit`.

The dictionary has two keys: `resources` and `rate`. In `resources` you can access information about `core` or `search`.

Note: the `rate` key will be deprecated before version 3 of the GitHub API is finalized. Do not rely on that key. Instead, make your code future-proof by using `core` in `resources`, e.g.,

```
rates = g.rate_limit()
rates['resources']['core'] # => your normal ratelimit info
rates['resources']['search'] # => your search ratelimit info
```

New in version 0.8.

**Returns** ratelimit mapping

**Return type** dict

`github3.repository` (*owner, repository*)

Retrieve the desired repository.

**Parameters**

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

**Returns** the repository

**Return type** *Repository*

`github3.search_code` (*query, sort=None, order=None, per\_page=None, text\_match=False, number=-1, etag=None*)

Find code via the code search API.

**Warning:** You will only be able to make 5 calls with this or other search functions. To raise the rate-limit on this set of endpoints, create an authenticated *GitHub* Session with `login`.

The query can contain any combination of the following supported qualifiers:

- `in` Qualifies which fields are searched. With this qualifier you can restrict the search to just the file contents, the file path, or both.
- `language` Searches code based on the language it's written in.
- `fork` Specifies that code from forked repositories should be searched. Repository forks will not be searchable unless the fork has more stars than the parent repository.
- `size` Finds files that match a certain size (in bytes).
- `path` Specifies the path that the resulting file must be at.
- `extension` Matches files with a certain extension.
- `user` or `repo` Limits searches to a specific user or repository.

For more information about these qualifiers, see: <http://git.io/-DvAuA>

**Parameters**

- **query** (*str*) – (required), a valid query as described above, e.g., `addClass in:file language:js repo:jquery/jquery`
- **sort** (*str*) – (optional), how the results should be sorted; option(s): `indexed`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`

- **per\_page** (*int*) – (optional)
- **text\_match** (*bool*) – (optional), if True, return matching search terms. See <http://git.io/4ct1eQ> for more information
- **number** (*int*) – (optional), number of repositories to return. Default: -1, returns all available repositories
- **etag** (*str*) – (optional), previous ETag header value

**Returns** generator of *CodeSearchResult*

`github3.search_issues(query, sort=None, order=None, per_page=None, text_match=False, number=-1, etag=None)`

Find issues by state and keyword

**Warning:** You will only be able to make 5 calls with this or other search functions. To raise the rate-limit on this set of endpoints, create an authenticated *GitHub* Session with `login`.

The query can contain any combination of the following supported qualifiers:

- **type** With this qualifier you can restrict the search to issues or pull request only.
- **in** Qualifies which fields are searched. With this qualifier you can restrict the search to just the title, body, comments, or any combination of these.
- **author** Finds issues created by a certain user.
- **assignee** Finds issues that are assigned to a certain user.
- **mentions** Finds issues that mention a certain user.
- **commenter** Finds issues that a certain user commented on.
- **involves** Finds issues that were either created by a certain user, assigned to that user, mention that user, or were commented on by that user.
- **state** Filter issues based on whether they're open or closed.
- **labels** Filters issues based on their labels.
- **language** Searches for issues within repositories that match a certain language.
- **created** or **updated** Filters issues based on times of creation, or when they were last updated.
- **comments** Filters issues based on the quantity of comments.
- **user** or **repo** Limits searches to a specific user or repository.

For more information about these qualifiers, see: <http://git.io/d1oELA>

#### Parameters

- **query** (*str*) – (required), a valid query as described above, e.g., `windows label:bug`
- **sort** (*str*) – (optional), how the results should be sorted; options: `created`, `comments`, `updated`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`
- **per\_page** (*int*) – (optional)
- **text\_match** (*bool*) – (optional), if True, return matching search terms. See <http://git.io/QLQuSQ> for more information



- **number** (*int*) – (optional), number of issues to return. Default: -1, returns all available issues
- **etag** (*str*) – (optional), previous ETag header value

**Returns** generator of *IssueSearchResult*

`github3.search_repositories` (*query, sort=None, order=None, per\_page=None, text\_match=False, number=-1, etag=None*)

Find repositories via various criteria.

**Warning:** You will only be able to make 5 calls with this or other search functions. To raise the rate-limit on this set of endpoints, create an authenticated *GitHub* Session with `login`.

The query can contain any combination of the following supported qualifiers:

- **in** Qualifies which fields are searched. With this qualifier you can restrict the search to just the repository name, description, readme, or any combination of these.
- **size** Finds repositories that match a certain size (in kilobytes).
- **forks** Filters repositories based on the number of forks, and/or whether forked repositories should be included in the results at all.
- **created** or **pushed** Filters repositories based on times of creation, or when they were last updated. Format: YYYY-MM-DD. Examples: `created:<2011`, `pushed:<2013-02`, `pushed:>=2013-03-06`
- **user** or **repo** Limits searches to a specific user or repository.
- **language** Searches repositories based on the language they're written in.
- **stars** Searches repositories based on the number of stars.

For more information about these qualifiers, see: <http://git.io/4Z8AkA>

#### Parameters

- **query** (*str*) – (required), a valid query as described above, e.g., `tetris language:assembly`
- **sort** (*str*) – (optional), how the results should be sorted; options: `stars`, `forks`, `updated`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`
- **per\_page** (*int*) – (optional)
- **text\_match** (*bool*) – (optional), if True, return matching search terms. See <http://git.io/4ct1eQ> for more information
- **number** (*int*) – (optional), number of repositories to return. Default: -1, returns all available repositories
- **etag** (*str*) – (optional), previous ETag header value

**Returns** generator of *Repository*

`github3.search_users` (*query, sort=None, order=None, per\_page=None, text\_match=False, number=-1, etag=None*)

Find users via the Search API.

**Warning:** You will only be able to make 5 calls with this or other search functions. To raise the rate-limit on this set of endpoints, create an authenticated *GitHub* Session with `login`.

The query can contain any combination of the following supported qualifiers:

- `type` With this qualifier you can restrict the search to just personal accounts or just organization accounts.
- `in` Qualifies which fields are searched. With this qualifier you can restrict the search to just the username, public email, full name, or any combination of these.
- `repos` Filters users based on the number of repositories they have.
- `location` Filter users by the location indicated in their profile.
- `language` Search for users that have repositories that match a certain language.
- `created` Filter users based on when they joined.
- `followers` Filter users based on the number of followers they have.

For more information about these qualifiers see: <http://git.io/wjVYJw>

#### Parameters

- **query** (*str*) – (required), a valid query as described above, e.g., `tom repos:>42 followers:>1000`
- **sort** (*str*) – (optional), how the results should be sorted; options: `followers`, `repositories`, or `joined`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`
- **per\_page** (*int*) – (optional)
- **text\_match** (*bool*) – (optional), if `True`, return matching search terms. See [http://git.io/\\_V1zRwa](http://git.io/_V1zRwa) for more information
- **number** (*int*) – (optional), number of search results to return; Default: `-1` returns all available
- **etag** (*str*) – (optional), ETag header value of the last request.

**Returns** generator of *UserSearchResult*

`github3.user(username)`

Retrieve a *User* object for the specified user name.

**Parameters** `username` (*str*) – name of the user

**Returns** the user

**Return type** *User*

`github3.zen()`

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

**Returns** *str*

## 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.

```
from github3 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')
```

### 12.1.2 Authorizations API Classes

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

**class** `github3.auths.Authorization` (*json, session*)  
Representation of an OAuth Authorization.

See also: [https://developer.github.com/v3/oauth\\_authorizations/](https://developer.github.com/v3/oauth_authorizations/)

This object has the following attributes:

**app**

Details about the application the authorization was created for.

**created\_at**

A `datetime` representing when this authorization was created.

**fingerprint**

New in version 1.0.

The optional parameter that is used to allow an OAuth application to create multiple authorizations for the same user. This will help distinguish two authorizations for the same app.

**hashed\_token**

New in version 1.0.

This is the base64 of the SHA-256 digest of the token.

**See also:**

**Removing Authorization Tokens** The blog post announcing the removal of `token`.

**id**

The unique identifier for this authorization.

**note\_url**

The URL that points to a longer description about the purpose of this authorization.

**note**

The short note provided when this authorization was created.

**scopes**

The list of scopes assigned to this token.

**See also:**

**Scopes for OAuth Applications** GitHub's documentation around available scopes and what they mean

**token**

If this authorization was created, this will contain the full token. Otherwise, this attribute will be an empty string.

**token\_last\_eight**

New in version 1.0.

The last eight characters of the token. This allows users to identify a token after the initial retrieval.

**updated\_at**

A `datetime` representing when this authorization was most recently updated.

**add\_scopes** (*scopes*, *note=None*, *note\_url=None*)

Add the scopes to this authorization.

New in version 1.0.

**Parameters**

- **scopes** (*list*) – Adds these scopes to the ones present on this authorization
- **note** (*str*) – (optional), Note about the authorization
- **note\_url** (*str*) – (optional), URL to link to when the user views the authorization

**Returns** True if successful, False otherwise

**Return type** bool

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete** ()

Delete this authorization.

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from `json_dict`.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from `json`.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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`

**remove\_scopes** (*scopes, note=None, note\_url=None*)

Remove the scopes from this authorization.

New in version 1.0.

**Parameters**

- **scopes** (*list*) – Remove these scopes from the ones present on this authorization
- **note** (*str*) – (optional), Note about the authorization
- **note\_url** (*str*) – (optional), URL to link to when the user views the authorization

**Returns** True if successful, False otherwise

**Return type** `bool`

**replace\_scopes** (*scopes, note=None, note\_url=None*)

Replace the scopes on this authorization.

New in version 1.0.

**Parameters**

- **scopes** (*list*) – Use these scopes instead of the previous list
- **note** (*str*) – (optional), Note about the authorization
- **note\_url** (*str*) – (optional), URL to link to when the user views the authorization

**Returns** True if successful, False otherwise

**Return type** `bool`

### 12.1.3 Events API Classes

This part of the documentation covers the objects that represent data returned by the Events API.

#### The Event Object

**class** github3.events.**Event** (*json, session*)

Represents an event as returned by the API.

It structures and handles the data returned by via the [Events](#) section of the GitHub API.

Two events can be compared like so:

```
e1 == e2
e1 != e2
```

And that is equivalent to:

```
e1.id == e2.id
e1.id != e2.id
```

#### **actor**

A [EventUser](#) that represents the user whose action generated this event.

#### **created\_at**

A `datetime` representing when this event was created.

#### **id**

The unique identifier for this event.

#### **org**

If present, a [EventOrganization](#) representing the organization on which this event occurred.

#### **type**

The type of event this is.

#### **See also:**

[Event Types Documentation](#) GitHub's documentation of different event types

#### **payload**

The payload of the event which has all of the details relevant to this event.

#### **repo**

The string representation of the repository this event pertains to.

Changed in version 1.0.0: This restores the behaviour of the API. To get a tuple, representation, use `self.repo.split('/', 1)`

#### **public**

A boolean representing whether the event is publicly viewable or not.

#### **as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from *json*.

**static list\_types** ()

List available payload types.

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

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

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

### Event Related Objects

The following objects are returned as part of an *Event*. These objects all have methods to convert them to full representations of the object. For example, *EventUser* has `to_user()` and aliases `refresh()` to behave similarly.

**class** `github3.events.EventUser` (*json*, *session*)  
The class that represents the user information returned in Events.

---

**Note:** Refreshing this object will return a *User*.

---

**avatar\_url**

The URL of the avatar image this user chose.

**display\_login**

The login that is displayed as part of the event.

**gravatar\_id**

The unique ID for the user's gravatar, if they're using gravatar to host their avatar.

**id**

The user's unique ID in GitHub.

**login**

The user's login (or handle) on GitHub.

**class** `github3.events.EventOrganization` (*json*, *session*)  
Representation of the organization information returned in Events.

---

**Note:** Refreshing this object will return a *Organization*.

---

This object has the following attributes:

**avatar\_url**

The URL to this organization's avatar.

**gravatar\_id**

The unique identifier for this organization on Gravatar, if its avatar is hosted there.

**id**

This organization's unique identifier on GitHub.



**login**

The unique login for this organization.

**class** github3.events.**EventPullRequest** (*json, session*)

Representation of a Pull Request returned in Events.

---

**Note:** Refreshing this object returns a *PullRequest*.

---

This object has the following attributes:

**id**

The unique id of this pull request across all of GitHub.

**number**

The number of this pull request on its repository.

**state**

The state of this pull request during this event.

**title**

The title of this pull request during this event.

**locked**

A boolean attribute describing if this pull request was locked.

**class** github3.events.**EventReviewComment** (*json, session*)

Representation of review comments in events.

---

**Note:** Refreshing this object will return a new :class:`~github3.pulls.ReviewComment`

---

This object has the following attributes:

**id**

The unique id of this comment across all of GitHub.

**author\_association**

The association the author has with this project.

**body**

The markdown body of this review comment.

**commit\_id**

The identifier of the commit that this comment was left on.

**created\_at**

A *datetime* object representing the date and time this comment was created.

**diff\_hunk**

The section (or hunk) of the diff this comment was left on.

**html\_url**

The URL to view this comment in a browser.

**links**

A dictionary of links to various items about this comment.

**original\_commit\_id**

The identifier of original commit this comment was left on.

**original\_position**

The original position within the diff this comment was left.

**path**

The path to the file this comment was left on.

**position**

The current position within the diff this comment is placed.

**pull\_request\_url**

The URL to retrieve the pull request information from the API.

**updated\_at**

A `datetime` object representing the date and time this comment was updated.

**user**

A `ShortUser` representing the user who authored this comment.

**class** `github3.events.EventIssue` (*json, session*)

The class that represents the issue information returned in Events.

**class** `github3.events.EventIssueComment` (*json, session*)

Representation of a comment left on an issue.

See also: <http://developer.github.com/v3/issues/comments/>

This object has the following attributes:

**author\_association**

The association of the author (*user*) with the repository this issue belongs to.

**body**

The markdown formatted original text written by the author.

**created\_at**

A `datetime` object representing the date and time when this comment was created.

**html\_url**

The URL to view this comment in a browser.

**id**

The unique identifier for this comment.

**issue\_url**

The URL of the parent issue in the API.

**updated\_at**

A `datetime` object representing the date and time when this comment was most recently updated.

**user**

A `ShortUser` representing the author of this comment.

## 12.1.4 Gist API Objects

The Gists API has a rich set of objects it returns.

### Gist Representations

**class** `github3.gists.gist.ShortGist` (*json, session*)

Short representation of a gist.

GitHub's API returns different amounts of information about gists based upon how that information is retrieved. This object exists to represent the full amount of information returned for a specific gist. For example, you would receive this class when calling `all_gists()`. To provide a clear distinction between the types of gists, `github3.py` uses different classes with different sets of attributes.

This object only has the following attributes:

**url**

The GitHub API URL for this repository, e.g., `https://api.github.com/gists/6faaaeb956dec3f51a9bd630a3490291`.

**comments\_count**

Number of comments on this gist

**description**

Description of the gist as written by the creator

**html\_url**

The URL of this gist on GitHub, e.g., `https://gist.github.com/sigmavirus24/6faaaeb956dec3f51a9bd630a3490291`

**id**

The unique identifier for this gist.

**public**

This is a boolean attribute describing if the gist is public or private

**git\_pull\_url**

The git URL to pull this gist, e.g., `git://gist.github.com/sigmavirus24/6faaaeb956dec3f51a9bd630a3490291.git`

**git\_push\_url**

The git URL to push to gist, e.g., `git@gist.github.com:sgmavirus24/6faaaeb956dec3f51a9bd630a3490291.git`

**created\_at**

This is a datetime object representing when the gist was created.

**updated\_at**

**This is a datetime object representing the last time this gist was most recently updated.**

**owner**

This attribute is a *ShortUser* object representing the creator of the gist.

**files**

A dictionary mapping the filename to a *GistFile* object.

Changed in version 1.0.0: Previously this was a list but it has been converted to a dictionary to preserve the structure of the API.

**comments\_url**

The URL to retrieve the list of comments on the Gist via the API.

**class** `github3.gists.gist.GistFork` (*json, session*)

This object represents a forked Gist.

This has a subset of attributes of a *ShortGist*:

**created\_at**

The date and time when the gist was created.

**id**

The unique identifier of the gist.

**owner**

The user who forked the gist.

**updated\_at**

The date and time of the most recent modification of the fork.

**url**

The API URL for the fork.

**class** github3.gists.gist.**Gist** (*json, session*)

This object constitutes the full representation of a Gist.

GitHub's API returns different amounts of information about gists based upon how that information is retrieved. This object exists to represent the full amount of information returned for a specific gist. For example, you would receive this class when calling *gist()*. To provide a clear distinction between the types of gists, github3.py uses different classes with different sets of attributes.

This object has all the same attributes as *ShortGist* as well as:

**commits\_url**

The URL to retrieve gist commits from the GitHub API.

**original\_forks**

A list of *GistFork* objects representing each fork of this gist. To retrieve the most recent list of forks, use the *forks()* method.

**forks\_url**

The URL to retrieve the current listing of forks of this gist.

**history**

A list of *GistHistory* objects representing each change made to this gist.

**truncated**

This is a boolean attribute that indicates whether the content of this Gist has been truncated or not.

## Files in a Gist

Gists have files which have two representations:

**class** github3.gists.file.**GistFile** (*json, session*)

This represents the full file object returned by interacting with gists.

The object has all of the attributes as returned by the API for a *ShortGistFile* as well as:

**truncated**

A boolean attribute that indicates whether *original\_content* contains all of the file's contents.

**original\_content**

The contents of the file (potentially truncated) returned by the API. If the file was truncated use *content()* to retrieve it in its entirety.

**class** github3.gists.file.**ShortGistFile** (*json, session*)

This represents the file object returned by interacting with gists.

The object has the following attributes as returned by the API for a Gist:

**raw\_url**

This URL provides access to the complete, untruncated content of the file represented by this object.

**filename**

The string for the filename.

**language**

The GitHub detected language for the file, e.g., Erlang, Python, text.

**type**

The mime-type of the file. Related to *language*.

**size**

The file size in bytes.

## The History of a Gist

**class** `github3.gists.history.GistHistory` (*json, session*)

This object represents one version (or revision) of a gist.

The GitHub API returns the following attributes:

**url**

The URL to the revision of the gist retrievable through the API.

**version**

The commit ID of the revision of the gist.

**user**

The *ShortUser* representation of the user who owns this gist.

**committed\_at**

The date and time of the revision's commit.

**change\_status**

A dictionary with the number of deletions, additions, and total changes to the gist.

For convenience, github3.py also exposes the following attributes from the *change\_status*:

**additions**

The number of additions to the gist compared to the previous revision.

**deletions**

The number of deletions from the gist compared to the previous revision.

**totoal**

The total number of changes to the gist compared to the previous revision.

## 12.1.5 Git API Classes

This part of the documentation covers the module associated with the [Git Data](#) section of the GitHub API.

Like much of the GitHub API, many objects have different representations.

### Blob Object(s)

**class** `github3.git.Blob` (*json, session*)

This object provides an interface to the API representation of a blob.

See also: <http://developer.github.com/v3/git/blobs/>

Changed in version 1.0.0: The *content* is no longer forcibly coerced to bytes. The *decoded* is deprecated in favor of `decode_content()`.

This object has the following attributes

**content**

The raw content of the blob. This may be base64 encoded text. Use `decode_content()` to receive the non-encoded text.

**encoding**

The encoding that GitHub reports for this blob's content.

**size**

The size of this blob's content in bytes.

**sha**

The SHA1 of this blob's content.

### Commit Object(s)

**class** `github3.git.Commit` (*json, session*)

This represents a commit as returned by the git API.

This is distinct from *RepoCommit*. Primarily this object represents the commit data stored by git and it has no relationship to the repository on GitHub.

See also: <http://developer.github.com/v3/git/commits/>

This object has all of the attributes of a *ShortCommit* as well as the following attributes:

**parents**

The list of commits that are the parents of this commit. This may be empty if this is the initial commit, or it may have several if it is the result of an octopus merge. Each parent is represented as a dictionary with the API URL and SHA1.

**sha**

The unique SHA1 which identifies this commit.

**verification**

The GPG verification data about this commit. See <https://developer.github.com/v3/git/commits/#commit-signature-verification> for more information.

**class** `github3.git.ShortCommit` (*json, session*)

This represents a commit as returned by the git API.

This is distinct from *RepoCommit*. Primarily this object represents the commit data stored by git. This shorter representation of a *Commit* is most often found on a *RepoCommit* to represent the git data associated with it.

See also: <http://developer.github.com/v3/git/commits/>

This object has the following attributes:

**author**

This is a dictionary with at least the name and email of the author of this commit as well as the date it was authored.

**committer**

This is a dictionary with at least the name and email of the committer of this commit as well as the date it was committed.

**message**

The commit message that describes the changes as written by the author and committer.

**tree**

The git tree object this commit points to.

## Tree Object(s)

**class** `github3.git.CommitTree` (*json, session*)

This object represents the abbreviated tree data in a commit.

The API returns different representations of different objects. When representing a *ShortCommit* or *Commit*, the API returns an abbreviated representation of a git tree.

This object has the following attributes:

**sha**

The SHA1 of this tree in the git repository.

**class** `github3.git.Hash` (*json, session*)

This is used to represent the elements of a tree.

This provides the path to the object and the type of object it is. For a brief explanation of what these types are and represent, this StackOverflow question answers some of that: <https://stackoverflow.com/a/18605496/1953283>

See also: <http://developer.github.com/v3/git/trees/#create-a-tree>

This object has the following attributes:

**mode**

The mode of the file, directory, or link.

**path**

The path to the file, directory, or link.

**sha**

The SHA1 for this hash.

**size**

This attribute is only not None if the *type* is not a tree.

**type**

The type of git object this is representing, e.g., tree, blob, etc.

**class** `github3.git.Tree` (*json, session*)

This represents a tree object from a git repository.

Trees tend to represent directories and subdirectories.

See also: <http://developer.github.com/v3/git/trees/>

This object has the following attributes:

**sha**

The SHA1 of this tree in the git repository.

**tree**

A list that represents the nodes in the tree. If this list has members it will have instances of *Hash*.

## Git Object, Reference, and Tag Object(s)

Yes, we know, `GitObject` is a funky name.

**class** `github3.git.GitObject` (*json, session*)

This object represents an arbitrary ‘object’ in git.

This object is intended to be versatile and is usually found on one of the following:

- *Reference*

- *Tag*

This object has the following attributes:

**sha**

The SHA1 of the object this is representing.

**type**

The name of the type of object this is representing.

**class** `github3.git.Reference` (*json, session*)

Object representing a git reference associated with a repository.

This represents a reference (or ref) created on a repository via git.

See also: <http://developer.github.com/v3/git/refs/>

This object has the following attributes:

**object**

A *GitObject* that this reference points to.

**ref**

The string path to the reference, e.g., 'refs/heads/sc/feature-a'.

**class** `github3.git.Tag` (*json, session*)

This represents an annotated tag.

Tags are a special kind of git reference and annotated tags have more information than lightweight tags.

See also: <http://developer.github.com/v3/git/tags/>

This object has the following attributes:

**message**

This is the message that was written to accompany the creation of the annotated tag.

**object**

A *GitObject* that represents the underlying git object.

**sha**

The SHA1 of this tag in the git repository.

**tag**

The “lightweight” tag (or reference) that backs this annotated tag.

**tagger**

The person who created this tag.

### 12.1.6 GitHub Object

The GitHub objects is the point at which most usage of github3.py works.

#### GitHub.com Object

**class** `github3.github.GitHub` (*username=”, password=”, token=”, session=None*)

Stores all the session information.

There are two ways to log into the GitHub API



```

from github3 import login
g = login(user, password)
g = login(token=token)
g = login(user, token=token)

```

or

```

from github3 import GitHub
g = GitHub(user, password)
g = GitHub(token=token)
g = GitHub(user, token=token)

```

This is simple backward compatibility since originally there was no way to call the GitHub object with authentication parameters.

**add\_email\_addresses** (*addresses=[]*)

Add the addresses to the authenticated user's account.

**Parameters** **addresses** (*list*) – (optional), email addresses to be added

**Returns** list of email objects

**Return type** [*Email*]

**all\_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

**Return type** *Event*

**all\_organizations** (*number=-1, since=None, etag=None, per\_page=None*)

Iterate over every organization in the order they were created.

**Parameters**

- **number** (*int*) – (optional), number of organizations to return. Default: -1, returns all of them
- **since** (*int*) – (optional), last organization id seen (allows restarting an iteration)
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint
- **per\_page** (*int*) – (optional), number of organizations to list per request

**Returns** generator of organizations

**Return type** *ShortOrganization*

**all\_repositories** (*number=-1, since=None, etag=None, per\_page=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
- **since** (*int*) – (optional), last repository id seen (allows restarting an iteration)

- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint
- **per\_page** (*int*) – (optional), number of repositories to list per request

**Returns** generator of repositories

**Return type** *ShortRepository*

**all\_users** (*number=-1, etag=None, per\_page=None, since=None*)

Iterate over every user in the order they signed up for GitHub.

Changed in version 1.0.0: Inserted the `since` parameter after the `number` parameter.

**Parameters**

- **number** (*int*) – (optional), number of users to return. Default: -1, returns all of them
- **since** (*int*) – (optional), ID of the last user that you've seen.
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint
- **per\_page** (*int*) – (optional), number of users to list per request

**Returns** generator of users

**Return type** *ShortUser*

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**authorization** (*id\_num*)

Get information about authorization `id`.

**Parameters** **id\_num** (*int*) – (required), unique id of the authorization

**Returns** *Authorization*

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

Iterate over authorizations for the authenticated user.

---

**Note:** 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

**Return type** *Authorization*

**authorize** (*username, password, scopes=None, note="", note\_url="", client\_id="", client\_secret=""*)  
Obtain an authorization token.

The retrieved token will allow future consumers to use the API without a username and password.

#### Parameters

- **username** (*str*) – (required)
- **password** (*str*) – (required)
- **scopes** (*list*) – (optional), 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** created authorization

**Return type** *Authorization*

**check\_authorization** (*access\_token*)

Check an authorization created by a registered application.

OAuth applications can use this method to check token validity without hitting normal rate limits because of failed login attempts. If the token is valid, it will return True, otherwise it will return False.

**Returns** True if token is valid, False otherwise

**Return type** bool

**create\_gist** (*description, files, public=True*)

Create a new gist.

Changed in version 1.1.0: Per [GitHub’s recent announcement](#) authentication is now required for creating gists.

#### 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** the created gist if successful, otherwise None

**Return type** created gist

**Return type** *Gist*

**create\_issue** (*owner, repository, title, body=None, assignee=None, milestone=None, labels=[], assignees=None*)  
Create an issue on the repository.

---

**Note:** `body, assignee, assignees, milestone, labels` are all optional.

---

**Warning:** This method retrieves the repository first and then uses it to create an issue. If you're making several issues, you should use `repository` and then use `create_issue`

#### 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
- **assignees** – (optional), logins of the users to assign the issue to
- **milestone** (*int*) – (optional), id number of the milestone to attribute this issue to (e.g. if `m` is a `Milestone` object, `m.number` is what you pass here.)
- **labels** (*list*) – (optional), List of label names.

**Returns** created issue

**Return type** `ShortIssue`

**create\_key** (*title, key, read\_only=False*)  
Create a new key for the authenticated user.

#### Parameters

- **title** (*str*) – (required), key title
- **key** (*str*) – (required), actual key contents, accepts path as a string or file-like object
- **read\_only** (*bool*) – (optional), restrict key access to read-only, default to `False`

**Returns** created key

**Return type** `Key`

**create\_repository** (*name, description="", homepage="", private=False, has\_issues=True, has\_wiki=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`
- **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** created repository

**Return type** *Repository*

**delete\_email\_addresses** (*addresses=[]*)

Delete the specified addresses the authenticated user's account.

**Parameters** **addresses** (*list*) – (optional), email addresses to be removed

**Returns** `True` if successful, `False` otherwise

**Return type** `bool`

**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 emails

**Return type** *Email*

**emojis** ()

Retrieve a dictionary of all of the emojis that GitHub supports.

**Returns**

dictionary where the key is what would be in between the colons and the value is the URL to the image, e.g.,

```
{
    '+1': 'https://github.global.ssl.fastly.net/images/...',
    # ...
}
```

**feeds** ()

List GitHub's timeline resources in Atom format.

**Returns** dictionary parsed to include URITemplates

**Return type** `dict`

**follow** (*username*)

Make the authenticated user follow the provided username.

**Parameters** **username** (*str*) – (required), user to follow

**Returns** `True` if successful, `False` otherwise

**Return type** `bool`

**followed\_by** (*username*, *number=-1*, *etag=None*)

Iterate over users being followed by *username*.

New in version 1.0.0: This replaces `iter_following('sigmavirus24')`.

**Parameters**

- **username** (*str*) – (required), 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

**Return type** *ShortUser*

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

Iterate over followers of the authenticated user.

New in version 1.0.0: This replaces `iter_followers()`.

**Parameters**

- **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 followers

**Return type** *ShortUser*

**followers\_of** (*username*, *number=-1*, *etag=None*)

Iterate over followers of *username*.

New in version 1.0.0: This replaces `iter_followers('sigmavirus24')`.

**Parameters**

- **username** (*str*) – (required), 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 followers

**Return type** *ShortUser*

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

Iterate over users the authenticated user is following.

New in version 1.0.0: This replaces `iter_following()`.

**Parameters**

- **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

**Return type** *ShortUser*

**classmethod** `from_dict` (*json\_dict*, *session*)

Return an instance of this class formed from `json_dict`.

**classmethod** `from_json` (*json*, *session*)

Return an instance of this class formed from `json`.

**gist** (*id\_num*)

Retrieve the gist using the specified id number.

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

**Returns** the gist identified by `id_num`

**Return type** `Gist`

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

Retrieve the authenticated user's gists.

New in version 1.0.

**Parameters**

- **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 short gists

**Return type** :class:`~github3.gists.ShortGist`

**gists\_by** (*username*, *number=-1*, *etag=None*)

Iterate over the gists owned by a user.

New in version 1.0.

**Parameters**

- **username** (*str*) – login of the user who owns the gists
- **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 short gists owned by the specified user

**Return type** `ShortGist`

**gitignore\_template** (*language*)

Return the template for language.

**Returns** the template string

**Return type** `str`

**gitignore\_templates** ()

Return the list of available templates.

**Returns** list of template names

**Return type** `[str]`

**is\_following** (*username*)

Check if the authenticated user is following login.

**Parameters** **username** (*str*) – (required), login of the user to check if the authenticated user is checking

**Returns** True if following, False otherwise

**Return type** bool

**is\_starred** (*username, repo*)

Check if the authenticated user starred username/repo.

**Parameters**

- **username** (*str*) – (required), owner of repository
- **repo** (*str*) – (required), name of repository

**Returns** True if starred, False otherwise

**Return type** bool

**issue** (*username, repository, number*)

Fetch issue from owner/repository.

**Parameters**

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

**Returns** the issue

**Return type** *Issue*

**issues** (*filter=”, state=”, labels=”, sort=”, direction=”, since=None, number=-1, etag=None*)

List all of the authenticated user’s (and organization’s) issues.

Changed in version 0.9.0: The `state` parameter now accepts ‘all’ in addition to ‘open’ and ‘closed’.

• **Parameters**

- **filter** (*str*) – accepted values: (‘assigned’, ‘created’, ‘mentioned’, ‘subscribed’) api-default: ‘assigned’
- **state** (*str*) – accepted values: (‘all’, ‘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** (*datetime* or *str*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, 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

**Return type** *ShortIssue*

**issues\_on** (*username, 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.

Changed in version 0.9.0: The `state` parameter now accepts ‘all’ in addition to ‘open’ and ‘closed’.



- **Parameters**

- **username** (*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: ('all', '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** (*datetime* or *str*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, 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

**Return type** `ShortIssue`

**key** (*id\_num*)

Get the authenticated user's key specified by *id\_num*.

**Parameters** **id\_num** (*int*) – (required), unique id of the key

**Returns** created key

**Return type** `Key`

**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

**Return type** `Key`

**license** (*name*)

Retrieve the license specified by the name.

**Parameters** **name** (*string*) – (required), name of license

**Returns** the specified license

**Return type** `License`

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

Iterate over open source licenses.

**Returns** generator of short license objects

**Return type** `ShortLicense`

**login** (*username=None, password=None, token=None, two\_factor\_callback=None*)

Log the user into GitHub for protected API calls.

**Parameters**

- **username** (*str*) – login name
- **password** (*str*) – password for the login
- **token** (*str*) – OAuth token
- **two\_factor\_callback** (*func*) – (optional), function you implement to provide the Two Factor Authentication code to GitHub when necessary

**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** the HTML formatted markdown text

**Return type** `str` (or `unicode` on Python 2)

**me** ()

Retrieve the info for the authenticated user.

New in version 1.0: This was separated from the `user` method.

**Returns** the representation of the authenticated user.

**Return type** `AuthenticatedUser`

**membership\_in** (*organization*)

Retrieve the user’s membership in the specified organization.

**Parameters** **organization** (`Organization`) – the organization or organization login to retrieve the authorized user’s membership in

**Returns** the user’s membership

**Return type** `Membership`

**meta** ()

Retrieve a dictionary with arrays of addresses in CIDR format.

The addresses in CIDR format specify the addresses that the incoming service hooks will originate from.

New in version 0.5.

**Returns** CIDR addresses

**Return type** `dict`

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**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 threads

**Return type** *Thread*

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

Return an easter egg of the API.

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

**Returns** ascii art of Octocat

**Return type** *str* (or unicode on Python 2)

**organization** (*username*)

Return an Organization object for the login name.

**Parameters** **username** (*str*) – (required), login name of the org

**Returns** the organization

**Return type** *Organization*

**organization\_issues** (*name, filter="", state="", labels="", sort="", direction="", since=None, number=-1, etag=None*)

Iterate over the organization's issues.

---

**Note:** This only works if the authenticated user belongs to it.

---

**Parameters**

- **name** (*str*) – (required), name of the organization
- **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** (*datetime* or *str*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, 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 issues

**Return type** `ShortIssue`

**organization\_memberships** (*state=None, number=-1, etag=None*)

List organizations of which the user is a current or pending member.

**Parameters** **state** (*str*) – (option), state of the membership, i.e., active, pending

**Returns** iterator of memberships

**Return type** `Membership`

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

Iterate over all organizations the authenticated user belongs to.

This will display both the private memberships and the publicized memberships.

**Parameters**

- **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

**Return type** `ShortOrganization`

**organizations\_with** (*username, number=-1, etag=None*)

Iterate over organizations with `username` as a public member.

New in version 1.0.0: Replaces `iter_orgs('sigmavirus24')`.

**Parameters**

- **username** (*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

**Return type** `ShortOrganization`

**project** (*number*)

Return the Project with id `number`.

**Parameters** **number** (*int*) – id of the project

**Returns** the project

**Return type** `Project`

**project\_card** (*number*)

Return the ProjectCard with id `number`.

**Parameters** **number** (*int*) – id of the project card

**Returns** `ProjectCard`

**project\_column** (*number*)

Return the ProjectColumn with id `number`.

**Parameters** **number** (*int*) – id of the project column

**Returns** `ProjectColumn`

**public\_gists** (*number=-1, etag=None*)

Retrieve all public gists and iterate over them.

New in version 1.0.

**Parameters**

- **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 short gists

**Return type** *ShortGist*

**pubsubhubbub** (*mode, topic, callback, secret=""*)

Create or 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** True if successful, False otherwise

**Return type** 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** PullRequest

**rate\_limit** ()

Return a dictionary with information from /rate\_limit.

The dictionary has two keys: `resources` and `rate`. In `resources` you can access information about `core` or `search`.

Note: the `rate` key will be deprecated before version 3 of the GitHub API is finalized. Do not rely on that key. Instead, make your code future-proof by using `core` in `resources`, e.g.,

```
rates = g.rate_limit()
rates['resources']['core'] # => your normal ratelimit info
rates['resources']['search'] # => your search ratelimit info
```

New in version 0.8.

**Returns** ratelimit mapping

**Return type** dict

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**repositories** (*type=None, sort=None, direction=None, number=-1, etag=None*)

List repositories for the authenticated user, filterable by *type*.

Changed in version 0.6: Removed the login parameter for correctness. Use `repositories_by` instead

**Parameters**

- **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 repositories

**Return type** *ShortRepository*

**repositories\_by** (*username, type=None, sort=None, direction=None, number=-1, etag=None*)

List public repositories for the specified *username*.

New in version 0.6.

**Parameters**

- **username** (*str*) – (required), username
- **type** (*str*) – (optional), accepted values: ('all', 'owner', '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 repositories

**Return type** *ShortRepository*

**repository** (*owner, repository*)

Retrieve the desired repository.

**Parameters**

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

**Returns** the repository

**Return type** *Repository*

**repository\_with\_id** (*number*)

Retrieve the repository with the globally unique id.

**Parameters** **number** (*int*) – id of the repository

**Returns** the repository

**Return type** *Repository*

**revoke\_authorization** (*access\_token*)

Revoke specified authorization for an OAuth application.

Revoke all authorization tokens created by your application. This will only work if you have already called `set_client_id`.

**Parameters** **access\_token** (*str*) – (required), the `access_token` to revoke

**Returns** True if successful, False otherwise

**Return type** bool

**revoke\_authorizations** ()

Revoke all authorizations for an OAuth application.

Revoke all authorization tokens created by your application. This will only work if you have already called `set_client_id`.

**Parameters** **client\_id** (*str*) – (required), the `client_id` of your application

**Returns** True if successful, False otherwise

**Return type** bool

**search\_code** (*query, sort=None, order=None, per\_page=None, text\_match=False, number=-1, etag=None*)

Find code via the code search API.

The query can contain any combination of the following supported qualifiers:

- `in` Qualifies which fields are searched. With this qualifier you can restrict the search to just the file contents, the file path, or both.
- `language` Searches code based on the language it's written in.

- `fork` Specifies that code from forked repositories should be searched. Repository forks will not be searchable unless the fork has more stars than the parent repository.
- `size` Finds files that match a certain size (in bytes).
- `path` Specifies the path that the resulting file must be at.
- `extension` Matches files with a certain extension.
- `user` or `repo` Limits searches to a specific user or repository.

For more information about these qualifiers, see: <http://git.io/-DvAuA>

#### Parameters

- **query** (*str*) – (required), a valid query as described above, e.g., `addClass in:file language:js repo:jquery/jquery`
- **sort** (*str*) – (optional), how the results should be sorted; option(s): `indexed`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`
- **per\_page** (*int*) – (optional)
- **text\_match** (*bool*) – (optional), if `True`, return matching search terms. See <http://git.io/iRmJxg> for more information
- **number** (*int*) – (optional), number of repositories to return. Default: `-1`, returns all available repositories
- **etag** (*str*) – (optional), previous ETag header value

**Returns** generator of code search results

**Return type** `CodeSearchResult`

**search\_issues** (*query*, *sort=None*, *order=None*, *per\_page=None*, *text\_match=False*, *number=-1*, *etag=None*)

Find issues by state and keyword.

The query can contain any combination of the following supported qualifiers:

- `type` With this qualifier you can restrict the search to issues or pull request only.
- `in` Qualifies which fields are searched. With this qualifier you can restrict the search to just the title, body, comments, or any combination of these.
- `author` Finds issues created by a certain user.
- `assignee` Finds issues that are assigned to a certain user.
- `mentions` Finds issues that mention a certain user.
- `commenter` Finds issues that a certain user commented on.
- `involves` Finds issues that were either created by a certain user, assigned to that user, mention that user, or were commented on by that user.
- `state` Filter issues based on whether they're open or closed.
- `labels` Filters issues based on their labels.
- `language` Searches for issues within repositories that match a certain language.
- `created` or `updated` Filters issues based on times of creation, or when they were last updated.
- `comments` Filters issues based on the quantity of comments.



- `user` or `repo` Limits searches to a specific user or repository.

For more information about these qualifiers, see: <http://git.io/d1oELA>

#### Parameters

- **query** (*str*) – (required), a valid query as described above, e.g., `windows label:bug`
- **sort** (*str*) – (optional), how the results should be sorted; options: `created`, `comments`, `updated`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`
- **per\_page** (*int*) – (optional)
- **text\_match** (*bool*) – (optional), if True, return matching search terms. See <http://git.io/QLQuSQ> for more information
- **number** (*int*) – (optional), number of issues to return. Default: `-1`, returns all available issues
- **etag** (*str*) – (optional), previous ETag header value

**Returns** generator of issue search results

**Return type** `IssueSearchResult`

**search\_repositories** (*query*, *sort=None*, *order=None*, *per\_page=None*, *text\_match=False*, *number=-1*, *etag=None*)

Find repositories via various criteria.

The query can contain any combination of the following supported qualifiers:

- `in` Qualifies which fields are searched. With this qualifier you can restrict the search to just the repository name, description, readme, or any combination of these.
- `size` Finds repositories that match a certain size (in kilobytes).
- `forks` Filters repositories based on the number of forks, and/or whether forked repositories should be included in the results at all.
- `created` or `pushed` Filters repositories based on times of creation, or when they were last updated. Format: `YYYY-MM-DD`. Examples: `created:<2011`, `pushed:<2013-02`, `pushed:>=2013-03-06`
- `user` or `repo` Limits searches to a specific user or repository.
- `language` Searches repositories based on the language they're written in.
- `stars` Searches repositories based on the number of stars.

For more information about these qualifiers, see: <http://git.io/4Z8AkA>

#### Parameters

- **query** (*str*) – (required), a valid query as described above, e.g., `tetris language:assembly`
- **sort** (*str*) – (optional), how the results should be sorted; options: `stars`, `forks`, `updated`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`
- **per\_page** (*int*) – (optional)

- **text\_match** (*bool*) – (optional), if True, return matching search terms. See <http://git.io/4ct1eQ> for more information
- **number** (*int*) – (optional), number of repositories to return. Default: -1, returns all available repositories
- **etag** (*str*) – (optional), previous ETag header value

**Returns** generator of repository search results

**Return type** RepositorySearchResult

**search\_users** (*query*, *sort=None*, *order=None*, *per\_page=None*, *text\_match=False*, *number=-1*, *etag=None*)

Find users via the Search API.

The query can contain any combination of the following supported qualifiers:

- **type** With this qualifier you can restrict the search to just personal accounts or just organization accounts.
- **in** Qualifies which fields are searched. With this qualifier you can restrict the search to just the username, public email, full name, or any combination of these.
- **repos** Filters users based on the number of repositories they have.
- **location** Filter users by the location indicated in their profile.
- **language** Search for users that have repositories that match a certain language.
- **created** Filter users based on when they joined.
- **followers** Filter users based on the number of followers they have.

For more information about these qualifiers see: <http://git.io/wjVYJw>

#### Parameters

- **query** (*str*) – (required), a valid query as described above, e.g., `tom repos:>42 followers:>1000`
- **sort** (*str*) – (optional), how the results should be sorted; options: `followers`, `repositories`, or `joined`; default: `best match`
- **order** (*str*) – (optional), the direction of the sorted results, options: `asc`, `desc`; default: `desc`
- **per\_page** (*int*) – (optional)
- **text\_match** (*bool*) – (optional), if True, return matching search terms. See [http://git.io/\\_V1zRwa](http://git.io/_V1zRwa) for more information
- **number** (*int*) – (optional), number of search results to return; Default: -1 returns all available
- **etag** (*str*) – (optional), ETag header value of the last request.

**Returns** generator of user search results

**Return type** UserSearchResult

**set\_client\_id** (*id*, *secret*)

Allow the developer to set their OAuth application credentials.

#### Parameters

- **id** (*str*) – 20-character hexadecimal `client_id` provided by GitHub

- **secret** (*str*) – 40-character hexadecimal client\_secret provided by GitHub

**set\_user\_agent** (*user\_agent*)

Allow the user to set their own user agent string.

**Parameters** **user\_agent** (*str*) – string used to identify your application. Library default: “github3.py/{version}”, e.g., “github3.py/1.0.0”

**star** (*username, repo*)

Star a repository.

**Parameters**

- **username** (*str*) – (required), owner of the repo
- **repo** (*str*) – (required), name of the repo

**Returns** True if successful, False otherwise

**Return type** bool

**starred** (*sort=None, direction=None, number=-1, etag=None*)

Iterate over repositories starred by the authenticated user.

Changed in version 1.0.0: This was split from `iter_starred` and requires authentication.

**Parameters**

- **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 repositories

**Return type** `ShortRepository>`

**starred\_by** (*username, sort=None, direction=None, number=-1, etag=None*)

Iterate over repositories starred by username.

New in version 1.0: This was split from `iter_starred` and requires the login parameter.

**Parameters**

- **username** (*str*) – 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 repositories

**Return type** `ShortRepository`

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

Iterate over repositories subscribed to by the authenticated user.

**Parameters**

- **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 repositories

**Return type** *ShortRepository*

**subscriptions\_for** (*username, number=-1, etag=None*)

Iterate over repositories subscribed to by *username*.

**Parameters**

- **username** (*str*) – 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 subscribed repositories

**Return type** *ShortRepository*

**unfollow** (*username*)

Make the authenticated user stop following *username*.

**Parameters** **username** (*str*) – (required)

**Returns** True if successful, False otherwise

**Return type** bool

**unstar** (*username, repo*)

Unstar *username/repo*.

**Parameters**

- **username** (*str*) – (required), owner of the repo
- **repo** (*str*) – (required), name of the repo

**Returns** True if successful, False otherwise

**Return type** bool

**update\_me** (*name=None, email=None, blog=None, company=None, location=None, hireable=False, bio=None*)

Update the profile of the authenticated user.

**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** True if successful, False otherwise

**Return type** bool

**user** (*username*)

Retrieve a User object for the specified user name.

**Parameters** **username** (*str*) – name of the user

**Returns** the user

**Return type** *User*

**user\_issues** (*filter=""*, *state=""*, *labels=""*, *sort=""*, *direction=""*, *since=None*, *per\_page=None*, *number=-1*, *etag=None*)

List only the authenticated user's issues.

Will not list organization's issues. See [organization\\_issues\(\)](#).

Changed in version 1.0: *per\_page* parameter added before *number*

Changed in version 0.9.0: The *state* parameter now accepts 'all' in addition to 'open' and 'closed'.

- **Parameters**

- **filter** (*str*) – accepted values: ('assigned', 'created', 'mentioned', 'subscribed') api-default: 'assigned'
- **state** (*str*) – accepted values: ('all', '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** (*datetime* or *str*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, 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

**Return type** *ShortIssue*

**user\_teams** (*number=-1*, *etag=None*)

Get the authenticated user's teams across all of organizations.

List all of the teams across all of the organizations to which the authenticated user belongs. This method requires user or repo scope when authenticating via OAuth.

**Returns** generator of teams

**Return type** *ShortTeam*

**user\_with\_id** (*number*)

Get the user's information with id *number*.

**Parameters** **number** (*int*) – the user's id number

**Returns** the user

**Return type** *User*

**zen** ()

Return a quote from the Zen of GitHub.

Yet another API Easter Egg

**Returns** the zen of GitHub

**Return type** str (on Python 3, unicode on Python 2)

## Examples

There are some examples of how to get started with this object [here](#).

## 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, username="", password="", token="", verify=True,
                                     session=None)
```

An interface to a specific GitHubEnterprise instance.

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.

If you have a self signed SSL for your local github enterprise you can override the validation by passing `verify=False`.

**admin\_stats** (*option*)

Retrieve statistics about this GitHub Enterprise instance.

**Parameters** **option** (*str*) – (required), accepted values: ('all', 'repos', 'hooks', 'pages', 'orgs', 'users', 'pulls', 'issues', 'milestones', 'gists', 'comments')

**Returns** the statistics

**Return type** dict

**create\_user** (*login, email*)

Create a new user.

---

**Note:** This is only available for administrators of the instance.

---

### Parameters

- **login** (*str*) – (required), The user's username.
- **email** (*str*) – (required), The user's email address.

**Returns** created user

**Return type** `ShortUser`

## GitHubStatus Object

```
class github3.github.GitHubStatus (session=None)
```

A sleek interface to the GitHub System Status API.

This will only ever return the JSON objects returned by the API.

**api ()**  
Retrieve API status.

**last\_message ()**  
Retrieve the last message.

**messages ()**  
Retrieve all messages.

**status ()**  
Retrieve overall status.

### 12.1.7 Issues API Objects

The following objects represent data returned by the [Issues API](#)

#### Issues

**class** `github3.issues.issue.ShortIssue` (*json, session*)

Object for the shortened representation of an Issue.

GitHub's API returns different amounts of information about issues based upon how that information is retrieved. Often times, when iterating over several issues, GitHub will return less information. To provide a clear distinction between the types of issues, github3.py uses different classes with different sets of attributes.

New in version 1.0.0.

This object has the following attributes:

**assignee**

Deprecated since version 1.0.0: While the API still returns this attribute, it's not as useful in the context of multiple assignees.

If a user is assigned to this issue, then it will be represented as a *ShortUser*.

**assignees**

If users are assigned to this issue, then they will be represented as a list of *ShortUser*.

**body**

The markdown formatted text of the issue as written by the user who opened the issue.

**closed\_at**

If this issue is closed, this will be a `datetime` object representing the date and time this issue was closed. Otherwise it will be `None`.

**comments\_count**

The number of comments on this issue.

**comments\_url**

The URL to retrieve the comments on this issue from the API.

**created\_at**

A `datetime` object representing the date and time this issue was created.

**events\_url**

The URL to retrieve the events related to this issue from the API.

**html\_url**

The URL to view this issue in a browser.

**id**

The unique identifier for this issue in GitHub.

**labels\_url**

A `URITemplate` object that can expand to a URL to retrieve the labels on this issue from the API.

**locked**

A boolean attribute representing whether or not this issue is locked.

**milestone**

A `Milestone` object representing the milestone to which this issue was assigned.

**number**

The number identifying this issue on its parent repository.

**original\_labels**

If any are assigned to this issue, the list of `Label` objects representing the labels returned by the API for this issue.

**pull\_request\_urls**

If present, a dictionary of URLs for retrieving information about the associated pull request for this issue.

**state**

The current state of this issue, e.g., 'closed' or 'open'.

**title**

The title for this issue.

**updated\_at**

A `datetime` object representing the date and time when this issue was last updated.

**user**

A `ShortUser` representing the user who opened this issue.

**add\_labels (\*args)**

Add labels to this issue.

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

**Returns** list of labels

**Return type** `Label`

**as\_dict ()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json ()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string



**Return type** str

**assign** (*username*)

Assign user *username* to this issue.

This is a short cut for `edit()`.

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

**Returns** True if successful, False, otherwise

**Return type** bool

**close** ()

Close this issue.

**Returns** True if successful, False otherwise

**Return type** bool

**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** the comment identified by *id\_num*

**Return type** *IssueComment*

**comments** (*number=-1, sort=", direction=", since=None*)

Iterate over the comments on this issue.

**Parameters**

- **number** (*int*) – (optional), number of comments to iterate over Default: -1 returns all comments
- **sort** (*str*) – accepted values: ('created', 'updated') api-default: created
- **direction** (*str*) – accepted values: ('asc', 'desc') Ignored without the sort parameter
- **since** (*datetime or string*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, e.g., 2012-05-20T23:10:27Z

**Returns** iterator of comments

**Return type** *IssueComment*

**create\_comment** (*body*)

Create a comment on this issue.

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

**Returns** the created comment

**Return type** *IssueComment*

**edit** (*title=None, body=None, assignee=None, state=None, milestone=None, labels=None, assignees=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, or 0 to remove the milestone

---

**Note:** This is not the milestone's globally unique identifier, it's value in *number*.

---

- **labels** (*list*) – list of labels to apply this to
- **assignees** (*list of strings*) – (optional), login of the users to assign the issue to

**Returns** True if successful, False otherwise

**Return type** bool

**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 events on this issues

**Return type** *IssueEvent*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**is\_closed** ()

Check if the issue is closed.

**Returns** True if successful, False otherwise

**Return type** bool

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

Iterate over the labels associated with this issue.

**Parameters**

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

**Returns** generator of labels on this issue

**Return type** *Label*

**lock** ()

Lock an issue.

**Returns** True if successful, False otherwise

**Return type** bool

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**pull\_request()**

Retrieve the pull request associated with this issue.

**Returns** the pull request associated with this issue

**Return type** `PullRequest`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh(conditional=False)**

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**remove\_all\_labels()**

Remove all labels from this issue.

**Returns** the list of current labels (empty) if successful

**Return type** list

**remove\_label(name)**

Remove label name from this issue.

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

**Returns** list of removed labels

**Return type** `Label`

**reopen()**

Re-open a closed issue.

---

**Note:** This is a short cut to using `edit()`.

---

**Returns** True if successful, False otherwise

**Return type** bool

**replace\_labels** (*labels*)

Replace all labels on this issue with *labels*.

**Parameters** **labels** (*list*) – label names

**Returns** list of labels

**Return type** *Label*

**unlock** ()

Unlock an issue.

**Returns** True if successful, False otherwise

**Return type** bool

**class** github3.issues.issue.**Issue** (*json, session*)

Object for the full representation of an Issue.

GitHub’s API returns different amounts of information about issues based upon how that information is retrieved. This object exists to represent the full amount of information returned for a specific issue. For example, you would receive this class when calling *issue()*. To provide a clear distinction between the types of issues, github3.py uses different classes with different sets of attributes.

Changed in version 1.0.0.

This object has all of the same attributes as a *ShortIssue* as well as the following:

**body\_html**

The HTML formatted body of this issue.

**body\_text**

The plain-text formatted body of this issue.

**closed\_by**

If the issue is closed, a *ShortUser* representing the user who closed the issue.

**add\_labels** (*\*args*)

Add labels to this issue.

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

**Returns** list of labels

**Return type** *Label*

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object’s attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**assign** (*username*)

Assign user *username* to this issue.

This is a short cut for `edit()`.

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

**Returns** True if successful, False, otherwise

**Return type** bool

**close** ()

Close this issue.

**Returns** True if successful, False otherwise

**Return type** bool

**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** the comment identified by *id\_num*

**Return type** *IssueComment*

**comments** (*number=-1, sort=", direction=", since=None*)

Iterate over the comments on this issue.

**Parameters**

- **number** (*int*) – (optional), number of comments to iterate over Default: -1 returns all comments
- **sort** (*str*) – accepted values: ('created', 'updated') api-default: created
- **direction** (*str*) – accepted values: ('asc', 'desc') Ignored without the sort parameter
- **since** (*datetime or string*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, e.g., 2012-05-20T23:10:27Z

**Returns** iterator of comments

**Return type** *IssueComment*

**create\_comment** (*body*)

Create a comment on this issue.

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

**Returns** the created comment

**Return type** *IssueComment*

**edit** (*title=None, body=None, assignee=None, state=None, milestone=None, labels=None, assignees=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, or 0 to remove the milestone

---

**Note:** This is not the milestone's globally unique identifier, it's value in *number*.

---

- **labels** (*list*) – list of labels to apply this to
- **assignees** (*list of strings*) – (optional), login of the users to assign the issue to

**Returns** True if successful, False otherwise

**Return type** bool

**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 events on this issues

**Return type** *IssueEvent*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**is\_closed** ()

Check if the issue is closed.

**Returns** True if successful, False otherwise

**Return type** bool

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

Iterate over the labels associated with this issue.

**Parameters**

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

**Returns** generator of labels on this issue

**Return type** *Label*

**lock()**

Lock an issue.

**Returns** True if successful, False otherwise

**Return type** bool

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**pull\_request()**

Retrieve the pull request associated with this issue.

**Returns** the pull request associated with this issue

**Return type** *PullRequest*

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh(*conditional=False*)**

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**remove\_all\_labels()**

Remove all labels from this issue.

**Returns** the list of current labels (empty) if successful

**Return type** list

**remove\_label(*name*)**

Remove label name from this issue.

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

**Returns** list of removed labels

**Return type** *Label*

**reopen ()**  
Re-open a closed issue.

---

**Note:** This is a short cut to using `edit ()`.

---

**Returns** True if successful, False otherwise

**Return type** bool

**replace\_labels (labels)**  
Replace all labels on this issue with `labels`.

**Parameters** `labels (list)` – label names

**Returns** list of labels

**Return type** `Label`

**unlock ()**  
Unlock an issue.

**Returns** True if successful, False otherwise

**Return type** bool

## Issue Comments

**class** `github3.issues.comment.IssueComment (json, session)`  
Representation of a comment left on an issue.

See also: <http://developer.github.com/v3/issues/comments/>

This object has the following attributes:

**author\_association**  
The association of the author (`user`) with the repository this issue belongs to.

**body**  
The markdown formatted original text written by the author.

**body\_html**  
The HTML formatted comment body.

**body\_text**  
The plain-text formatted comment body.

**created\_at**  
A `datetime` object representing the date and time when this comment was created.

**html\_url**  
The URL to view this comment in a browser.

**id**  
The unique identifier for this comment.

**issue\_url**  
The URL of the parent issue in the API.

**updated\_at**  
A `datetime` object representing the date and time when this comment was most recently updated.



**user**

A *ShortUser* representing the author of this comment.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**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\_dict(*json\_dict*, *session*)**

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json(*json*, *session*)**

Return an instance of this class formed from *json*.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh(*conditional=False*)**

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

## Issue Events

**class** github3.issues.event.**IssueEvent** (*json, session*)

Representation of an event from a specific issue.

This object will be instantiated from calling `events()` which calls <https://developer.github.com/v3/issues/events/#list-events-for-an-issue>

See also: <http://developer.github.com/v3/issues/events>

This object has the following attributes:

**actor**

A *ShortUser* representing the user who generated this event.

**commit\_id**

The string SHA of a commit that referenced the parent issue. If there was no commit referencing this issue, then this will be `None`.

**commit\_url**

The URL to retrieve commit information from the API for the commit that references the parent issue. If there was no commit, this will be `None`.

**created\_at**

A *datetime* object representing the date and time this event occurred.

**event**

The issue-specific action that generated this event. Some examples are:

- closed
- reopened
- subscribed
- merged
- referenced
- mentioned
- assigned

See [this list of events](#) for a full listing.

**id**

The unique identifier for this event.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**classmethod from\_dict(json\_dict, session)**

Return an instance of this class formed from json\_dict.

**classmethod from\_json(json, session)**

Return an instance of this class formed from json.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh(conditional=False)**

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**class** github3.issues.event.**RepositoryIssueEvent** (*json, session*)

Representation of an issue event on the repository level.

This object will be instantiated from calling `issue_events()` or `issue_events()` which call <https://developer.github.com/v3/issues/events/#list-events-for-a-repository>

See also: <http://developer.github.com/v3/issues/events>

This object has all of the attributes of `IssueEvent` and the following:

**issue**

A `ShortIssue` representing the issue where this event originated from.

## Issue Labels

**class** github3.issues.label.**Label** (*json, session*)

A representation of a label object defined on a repository.

See also: <http://developer.github.com/v3/issues/labels/>

This object has the following attributes:

```
.. attribute:: color
```

The hexadecimal representation of the background color of this label.

**name**

The name (display label) for this label.

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete** ()

Delete this label.

**Returns** True if successfully deleted, False otherwise

**Return type** bool

**classmethod** **from\_dict** (*json\_dict, session*)

Return an instance of this class formed from `json_dict`.

**classmethod** `from_json` (*json*, *session*)

Return an instance of this class formed from `json`.

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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`

**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** `True` if successfully updated, `False` otherwise

**Return type** `bool`

## Milestone Objects

**class** `github3.issues.milestone.Milestone` (*json*, *session*)

Representation of milestones on a repository.

See also: <http://developer.github.com/v3/issues/milestones/>

This object has the following attributes:

**closed\_issues\_count**

The number of closed issues in this milestone.

**created\_at**

A `datetime` object representing the date and time when this milestone was created.

**creator**

If present, a *ShortUser* representing the user who created this milestone.

**description**

The written description of this milestone and its purpose.

**due\_on**

If set, a `datetime` object representing the date and time when this milestone is due.

**id**

The unique identifier of this milestone in GitHub.

**number**

The repository-local numeric identifier of this milestone. This starts at 1 like issues.

**open\_issues\_count**

The number of open issues still in this milestone.

**state**

The state of this milestone, e.g., 'open' or 'closed'.

**title**

The title of this milestone.

**updated\_at**

A `datetime` object representing the date and time when this milestone was last updated.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete this milestone.

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict(json\_dict, session)**

Return an instance of this class formed from `json_dict`.

**classmethod from\_json(json, session)**

Return an instance of this class formed from `json`.

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

Iterate over the labels of every associated issue.

Changed in version 0.9: Add etag parameter.

**Parameters**

- **number** (*int*) – (optional), number of labels to return. Default: -1 returns all available labels.
- **etag** (*str*) – (optional), ETag header from a previous response

**Returns** generator of labels

**Return type** *Label*

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** *int*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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*

**update** (*title=None, state=None, description=None, due\_on=None*)

Update this milestone.

All parameters are optional, but it makes no sense to omit all of them at once.

**Parameters**

- **title** (*str*) – (optional), new title of the milestone
- **state** (*str*) – (optional), ('open', 'closed')
- **description** (*str*) – (optional)
- **due\_on** (*str*) – (optional), ISO 8601 time format: YYYY-MM-DDTHH:MM:SSZ

**Returns** True if successful, False otherwise

**Return type** bool

## 12.1.8 Notifications

This part of the documentation covers the notifications module which contains all of the classes used to represent notification objects in github3.py.

### Notification Objects

**class** github3.notifications.**Thread**(*json, session*)

Object representing a notification thread.

Changed in version 1.0.0: The `comment`, `thread`, and `url` attributes are no longer present because GitHub stopped returning the comment that caused the notification.

The `is_unread` method was removed since it just returned the `unread` attribute.

This object has the following attributes:

**id**

The unique identifier for this notification across all GitHub notifications.

**last\_read\_at**

A `datetime` object representing the date and time when the authenticated user last read this thread.

**reason**

The reason the authenticated user is receiving this notification.

**repository**

A `ShortRepository` this thread originated on.

**subject**

A dictionary with the subject of the notification, for example, which issue, pull request, or diff this is in relation to.

**unread**

A boolean attribute indicating whether this thread has been read or not.

**updated\_at**

A `datetime` representing the date and time when this thread was last updated.

See also: <http://developer.github.com/v3/activity/notifications/>

**as\_dict**()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json**()

Return the json data for this object.

This is equivalent to calling:



```
json.dumps(obj.as_dict())
```

**Returns** this object’s attributes as a JSON string

**Return type** str

**delete\_subscription()**

Delete subscription for this thread.

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict(json\_dict, session)**

Return an instance of this class formed from json\_dict.

**classmethod from\_json(json, session)**

Return an instance of this class formed from json.

**mark()**

Mark the thread as read.

**Returns** True if successful, False otherwise

**Return type** bool

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh(conditional=False)**

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**set\_subscription(subscribed, ignored)**

Set the user’s subscription for this thread.

**Parameters**

- **subscribed** (*bool*) – (required), determines if notifications should be received from this thread.
- **ignored** (*bool*) – (required), determines if notifications should be ignored from this thread.

**Returns** new subscription

**Return type** *ThreadSubscription*

**subscription()**

Check the status of the user’s subscription to this thread.

**Returns** the subscription for this thread

**Return type** *ThreadSubscription*

**class** github3.notifications.**ThreadSubscription** (*json, session*)

This object provides a representation of a thread subscription.

See also: [developer.github.com/v3/activity/notifications/#get-a-thread-subscription](https://developer.github.com/v3/activity/notifications/#get-a-thread-subscription)

Changed in version 1.0.0: The `is_ignored` and `is_subscribed` methods were removed. Use the `:attr'ignored'` and `subscribed` attributes instead.

This object has the following attributes:

**created\_at**

A `datetime` object representing the date and time the user was subscribed to the thread.

**ignored**

A boolean attribute indicating whether the user ignored this.

**reason**

The reason the user is subscribed to the thread.

**subscribed**

A boolean attribute indicating whether the user is subscribed or not.

**thread\_url**

The URL of the thread resource in the GitHub API.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object’s attributes serialized to a dictionary

**Return type** `dict`

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object’s attributes as a JSON string

**Return type** `str`

**delete()**

Delete this subscription.

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**set** (*subscribed, ignored*)

Set the user's subscription for this subscription.

**Parameters**

- **subscribed** (*bool*) – (required), determines if notifications should be received from this thread.
- **ignored** (*bool*) – (required), determines if notifications should be ignored from this thread.

**class** github3.notifications.**RepositorySubscription** (*json, session*)

This object provides a representation of a thread subscription.

See also: [developer.github.com/v3/activity/notifications/#get-a-thread-subscription](https://developer.github.com/v3/activity/notifications/#get-a-thread-subscription)

Changed in version 1.0.0: The `is_ignored` and `is_subscribed` methods were removed. Use the `:attr'ignored'` and `subscribed` attributes instead.

This object has the following attributes:

**created\_at**

A `datetime` object representing the date and time the user was subscribed to the thread.

**ignored**

A boolean attribute indicating whether the user ignored this.

**reason**

The reason the user is subscribed to the thread.

**repository\_url**

The URL of the repository resource in the GitHub API.

**subscribed**

A boolean attribute indicating whether the user is subscribed or not.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete this subscription.

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict(json\_dict, session)**

Return an instance of this class formed from `json_dict`.

**classmethod from\_json(json, session)**

Return an instance of this class formed from `json`.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**set** (*subscribed, ignored*)

Set the user's subscription for this subscription.

**Parameters**

- **subscribed** (*bool*) – (required), determines if notifications should be received from this thread.
- **ignored** (*bool*) – (required), determines if notifications should be ignored from this thread.

## 12.1.9 Organizations and their Related Objects

This section of the documentation covers the objects that represent data returned by the API for organizations.

### Team Objects

**class** github3.orgs.**ShortTeam** (*json, session*)

Object representing a team in the GitHub API.

**id**

Unique identifier for this team across all of GitHub.

**members\_count**

The number of members in this team.

**members\_url**

A `URITemplate` instance to either retrieve all members in this team or to test if a user is a member.

**name**

The human-readable name provided to this team.

**permission**

The level of permissions this team has, e.g., push, pull, or admin.

**repos\_count**

The number of repositories this team can access.

**repositories\_url**

The URL of the resource to enumerate all repositories this team can access.

**slug**

The handle for this team or the portion you would use in an at-mention after the /, e.g., in @myorg/myteam the slug is myteam.

Please see GitHub's [Team Documentation](#) for more information.

**add\_member** (*username*)

Add *username* to this team.

**Parameters** **username** (*str*) – the username of the user you would like to add to this team.

**Returns** True if successfully added, False otherwise

**Return type** bool

**add\_repository** (*repository*, *permission=""*)

Add *repository* to this team.

If a permission is not provided, the team's default permission will be assigned, by GitHub.

**Parameters**

- **repository** (*str*) – (required), form: 'user/repo'
- **permission** (*str*) – (optional), ('pull', 'push', 'admin')

**Returns** True if successful, False otherwise

**Return type** bool

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete** ()

Delete this team.

**Returns** True if successful, False otherwise

**Return type** bool

**edit** (*name*, *permission=""*)

Edit this team.

**Parameters**

- **name** (*str*) – (required), the new name of this team
- **permission** (*str*) – (optional), one of ('pull', 'push', 'admin')

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**has\_repository** (*repository*)

Check if this team has access to *repository*.

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

**Returns** True if the team can access the repository, False otherwise

**Return type** bool

**invite** (*username*)

Invite the user to join this team.

This returns a dictionary like so:

```
{'state': 'pending', 'url': 'https://api.github.com/teams/...'}
```

**Parameters username** (*str*) – (required), login of user to invite to join this team.

**Returns** dictionary of the invitation response

**Return type** dict

**is\_member** (*username*)

Check if *login* is a member of this team.

**Parameters username** (*str*) – (required), username name of the user

**Returns** True if the user is a member, False otherwise

**Return type** bool

**members** (*role=None, number=-1, etag=None*)

Iterate over the members of this team.

**Parameters**

- **role** (*str*) – (optional), filter members returned by their role in the team. Can be one of: "member", "maintainer", "all". Default: "all".
- **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 the members of this team

**Return type** *ShortUser*

**membership\_for** (*username*)

Retrieve the membership information for the user.

**Parameters** `username` (*str*) – (required), name of the user

**Returns** dictionary with the membership

**Return type** dict

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**remove\_member** (*username*)

Remove `username` from this team.

**Parameters** `username` (*str*) – (required), username of the member to remove

**Returns** True if successful, False otherwise

**Return type** bool

**remove\_repository** (*repository*)

Remove `repository` from this team.

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

**Returns** True if successful, False otherwise

**Return type** bool

**repositories** (*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 repositories this team has access to

**Return type** `ShortRepository`

**revoke\_membership** (*username*)

Revoke this user's team membership.

**Parameters** **username** (*str*) – (required), name of the team member

**Returns** True if successful, False otherwise

**Return type** `bool`

**class** `github3.orgs.Team` (*json*, *session*)

Object representing a team in the GitHub API.

In addition to the attributes on a `ShortTeam` a `Team` has the following attribute:

**created\_at**

A `datetime` instance representing the time and date when this team was created.

**members\_count**

The number of members in this team.

**organization**

A `ShortOrganization` representing the organization this team belongs to.

**repos\_count**

The number of repositories this team can access.

**updated\_at**

A `datetime` instance representing the time and date when this team was updated.

Please see GitHub's [Team Documentation](#) for more information.

**add\_member** (*username*)

Add *username* to this team.

**Parameters** **username** (*str*) – the username of the user you would like to add to this team.

**Returns** True if successfully added, False otherwise

**Return type** `bool`

**add\_repository** (*repository*, *permission=""*)

Add *repository* to this team.

If a permission is not provided, the team's default permission will be assigned, by GitHub.

**Parameters**

- **repository** (*str*) – (required), form: 'user/repo'
- **permission** (*str*) – (optional), ('pull', 'push', 'admin')

**Returns** True if successful, False otherwise

**Return type** `bool`

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete this team.

**Returns** True if successful, False otherwise

**Return type** bool

**edit** (*name*, *permission=""*)

Edit this team.

**Parameters**

- **name** (*str*) – (required), the new name of this team
- **permission** (*str*) – (optional), one of ('pull', 'push', 'admin')

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from *json*.

**has\_repository** (*repository*)

Check if this team has access to *repository*.

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

**Returns** True if the team can access the repository, False otherwise

**Return type** bool

**invite** (*username*)

Invite the user to join this team.

This returns a dictionary like so:

```
{'state': 'pending', 'url': 'https://api.github.com/teams/...'}
```

**Parameters** **username** (*str*) – (required), login of user to invite to join this team.

**Returns** dictionary of the invitation response

**Return type** dict

**is\_member** (*username*)

Check if *login* is a member of this team.

**Parameters** `username` (*str*) – (required), username name of the user

**Returns** True if the user is a member, False otherwise

**Return type** bool

**members** (*role=None, number=-1, etag=None*)

Iterate over the members of this team.

**Parameters**

- **role** (*str*) – (optional), filter members returned by their role in the team. Can be one of: "member", "maintainer", "all". Default: "all".
- **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 the members of this team

**Return type** *ShortUser*

**membership\_for** (*username*)

Retrieve the membership information for the user.

**Parameters** `username` (*str*) – (required), name of the user

**Returns** dictionary with the membership

**Return type** dict

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**remove\_member** (*username*)

Remove *username* from this team.

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

**Returns** True if successful, False otherwise

**Return type** bool

**remove\_repository** (*repository*)

Remove *repository* from this team.

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

**Returns** True if successful, False otherwise

**Return type** bool

**repositories** (*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 repositories this team has access to

**Return type** ShortRepository

**revoke\_membership** (*username*)

Revoke this user's team membership.

**Parameters** **username** (*str*) – (required), name of the team member

**Returns** True if successful, False otherwise

**Return type** bool

**class** github3.orgs.Membership (*json, session*)

Object describing a user's membership in teams and organizations.

**organization**

A *ShortOrganization* instance representing the organization this membership is part of.

**organization\_url**

The URL of the organization resource in the API that this membership is part of.

**state**

The state of this membership, e.g., active or pending.

**active**

**Warning:** This is a computed attribute, it is not returned by the API.

A boolean attribute equivalent to `self.state.lower() == 'active'`.

**pending**

**Warning:** This is a computed attribute, it is not returned by the API.

A boolean attribute equivalent to `self.state.lower() == 'pending'`.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**edit(state)**

Edit the user's membership.

**Parameters** *state* (*str*) – (required), the state the membership should be in. Only accepts "active".

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict(json\_dict, session)**

Return an instance of this class formed from `json_dict`.

**classmethod from\_json(json, session)**

Return an instance of this class formed from `json`.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh(conditional=False)**

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

## Organization Objects

**class** github3.orgs.**ShortOrganization** (*json, session*)

Object for the shortened representation of an Organization.

GitHub's API returns different amounts of information about orgs based upon how that information is retrieved. Often times, when iterating over several orgs, GitHub will return less information. To provide a clear distinction between the types of orgs, github3.py uses different classes with different sets of attributes.

New in version 1.0.0.

**avatar\_url**

The URL of the avatar image for this organization.

**description**

The user-provided description of this organization.

**events\_url**

The URL to retrieve the events related to this organization.

**hooks\_url**

The URL for the resource to manage organization hooks.

**id**

The unique identifier for this organization across GitHub.

**issues\_url**

The URL to retrieve the issues across this organization's repositories.

**login**

The unique username of the organization.

**members\_url**

The URL to retrieve the members of this organization.

**public\_members\_url**

A `uritemplate.URITemplate` that can be expanded to either list the public members of this organization or verify a user is a public member.

**repos\_url**

The URL to retrieve the repositories in this organization.

**url**

The URL to retrieve this organization from the GitHub API.

**type**

Deprecated since version 1.0.0: This will be removed in a future release.

Previously returned by the API to indicate the type of the account.

**add\_member** (*username*, *team\_id*)

Add *username* to *team* and thereby to this organization.

**Warning:** This method is no longer valid. To add a member to a team, you must now retrieve the team directly, and use the `invite` method.

**Warning:** This method is no longer valid. To add a member to a team, you must now retrieve the team directly, and use the `invite` method.

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

Changed in version 1.0: The second parameter used to be `team` but has been changed to `team_id`. This parameter is now required to be an integer to improve performance of this method.

**Parameters**

- **username** (*str*) – (required), login name of the user to be added
- **team\_id** (*int*) – (required), team id

**Returns** True if successful, False otherwise

**Return type** bool

**add\_repository** (*repository*, *team\_id*)

Add *repository* to *team*.

Changed in version 1.0: The second parameter used to be `team` but has been changed to `team_id`. This parameter is now required to be an integer to improve performance of this method.

**Parameters**

- **repository** (*str*) – (required), form: ‘user/repo’
- **team\_id** (*int*) – (required), team id

**Returns** True if successful, False otherwise

**Return type** bool

**all\_events** (*username*, *number=-1*, *etag=None*)

Iterate over all org events visible to the authenticated user.

**Parameters**

- **username** (*str*) – (required), the username of the currently authenticated user.
- **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

**Return type** *Event*

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**conceal\_member(username)**

Conceal `username`'s membership in this organization.

**Parameters** `username` (*str*) – username of the organization member to conceal

**Returns** True if successful, False otherwise

**Return type** bool

**create\_project(name, body="")**

Create a project for this organization.

If the client is authenticated and a member of the organization, this will create a new project in the organization.

**Parameters**

- **name** (*str*) – (required), name of the project
- **body** (*str*) – (optional), the body of the project

**Returns** the new project

**Return type** Project

**create\_repository(name, description="", homepage="", private=False, has\_issues=True, has\_wiki=True, team\_id=0, auto\_init=False, gitignore\_template="", license\_template="")**

Create a repository for this organization.

If the client is authenticated and a member of the organization, this will create a new repository in the organization.

`name` should be no longer than 100 characters

**Parameters**

- **name** (*str*) – (required), name of the repository



**Warning:** this should be no longer than 100 characters

- **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`
- **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_init` is `False`.
- **license\_template** (*str*) – (optional), name of the license; this is ignored if `auto_init` is `False`.

**Returns** the created repository

**Return type** `Repository`

**create\_team** (*name*, *repo\_names*=[], *permission*='pull')

Create a new team and return it.

This only works if the authenticated user owns this organization.

#### Parameters

- **name** (*str*) – (required), name to be given to the team
- **repo\_names** (*list*) – (optional) repositories, e.g. [`'github/dotfiles'`]
- **permission** (*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

**Returns** the created team

**Return type** `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** True if successful, False otherwise

**Return type** bool

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

Iterate over public events for this org (deprecated).

Deprecated since version 1.0.0: Use `public_events()` instead.

**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

**Return type** *Event*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from `json_dict`.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from `json`.

**is\_member** (*username*)

Check if the user named `username` is a member.

**Parameters** **username** (*str*) – name of the user you'd like to check

**Returns** True if successful, False otherwise

**Return type** bool

**is\_public\_member** (*username*)

Check if the user named `username` is a public member.

**Parameters** **username** (*str*) – name of the user you'd like to check

**Returns** True if the user is a public member, False otherwise

**Return type** bool

**members** (*filter=None, role=None, number=-1, etag=None*)

Iterate over members of this organization.

**Parameters**

- **filter** (*str*) – (optional), filter members returned by this method. Can be one of: "2fa\_disabled", "all", . Default: "all". Filtering by "2fa\_disabled" is only available for organization owners with private repositories.
- **role** (*str*) – (optional), filter members returned by their role. Can be one of: "all", "admin", "member". 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 members of this organization

**Return type** *ShortUser*

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**project** (*id*, *etag=None*)

Return the organization project with the given ID.

**Parameters** **id** (*int*) – (required), ID number of the project

**Returns** requested project

**Return type** `Project`

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

Iterate over projects for 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 organization projects

**Return type** `Project`

**public\_events** (*number=-1*, *etag=None*)

Iterate over public 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 public events

**Return type** `Event`

**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 public members

**Return type** `ShortUser`

**publicize\_member** (*username*)

Make `username`'s membership in this organization public.

**Parameters** **username** (*str*) – the name of the user whose membership you wish to publicize

**Returns** True if successful, False otherwise

**Return type** `bool`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**remove\_member** (*username*)

Remove the user named *username* from this organization.

**Parameters** **username** (*str*) – name of the user to remove from the org

**Returns** True if successful, False otherwise

**Return type** bool

**remove\_repository** (*repository, team\_id*)

Remove *repository* from the team with *team\_id*.

**Parameters**

- **repository** (*str*) – (required), form: 'user/repo'
- **team\_id** (*int*) – (required), the unique identifier of the team

**Returns** True if successful, False otherwise

**Return type** bool

**repositories** (*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 repositories in this organization

**Return type** Repository

**team** (*team\_id*)

Return the team specified by `team_id`.

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

**Returns** the team identified by the id in this organization

**Return type** *Team*

**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 this organization’s teams

**Return type** *ShortTeam*

**class** `github3.orgs.Organization` (*json, session*)

Object for the full representation of a Organization.

GitHub’s API returns different amounts of information about orgs based upon how that information is retrieved. This object exists to represent the full amount of information returned for a specific org. For example, you would receive this class when calling `organization()`. To provide a clear distinction between the types of orgs, github3.py uses different classes with different sets of attributes.

Changed in version 1.0.0.

This object includes all attributes on `ShortOrganization` as well as the following:

**blog**

If set, the URL of this organization’s blog.

**company**

The name of the company that is associated with this organization.

**created\_at**

A `datetime` instance representing the time and date when this organization was created.

**email**

The email address associated with this organization.

**followers\_count**

The number of users following this organization. Organizations no longer have followers so this number will always be 0.

**following\_count**

The number of users this organization follows. Organizations no longer follow users so this number will always be 0.

**html\_url**

The URL used to view this organization in a browser.

**location**

The location of this organization, e.g., New York, NY.

**name**

The display name of this organization.

**public\_repos\_count**

The number of public repositories owned by thi sorganization.

**add\_member** (*username*, *team\_id*)

Add *username* to *team* and thereby to this organization.

**Warning:** This method is no longer valid. To add a member to a team, you must now retrieve the team directly, and use the `invite` method.

**Warning:** This method is no longer valid. To add a member to a team, you must now retrieve the team directly, and use the `invite` method.

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

Changed in version 1.0: The second parameter used to be `team` but has been changed to `team_id`. This parameter is now required to be an integer to improve performance of this method.

**Parameters**

- **username** (*str*) – (required), login name of the user to be added
- **team\_id** (*int*) – (required), team id

**Returns** True if successful, False otherwise

**Return type** bool

**add\_repository** (*repository*, *team\_id*)

Add *repository* to *team*.

Changed in version 1.0: The second parameter used to be `team` but has been changed to `team_id`. This parameter is now required to be an integer to improve performance of this method.

**Parameters**

- **repository** (*str*) – (required), form: ‘user/repo’
- **team\_id** (*int*) – (required), team id

**Returns** True if successful, False otherwise

**Return type** bool

**all\_events** (*username*, *number=-1*, *etag=None*)

Iterate over all org events visible to the authenticated user.

**Parameters**

- **username** (*str*) – (required), the username of the currently authenticated user.
- **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

**Return type** *Event*

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**conceal\_member** (*username*)

Conceal *username*'s membership in this organization.

**Parameters** **username** (*str*) – username of the organization member to conceal

**Returns** True if successful, False otherwise

**Return type** bool

**create\_project** (*name*, *body=""*)

Create a project for this organization.

If the client is authenticated and a member of the organization, this will create a new project in the organization.

**Parameters**

- **name** (*str*) – (required), name of the project
- **body** (*str*) – (optional), the body of the project

**Returns** the new project

**Return type** Project

**create\_repository** (*name*, *description=""*, *homepage=""*, *private=False*, *has\_issues=True*, *has\_wiki=True*, *team\_id=0*, *auto\_init=False*, *gitignore\_template=""*, *license\_template=""*)

Create a repository for this organization.

If the client is authenticated and a member of the organization, this will create a new repository in the organization.

*name* should be no longer than 100 characters

**Parameters**

- **name** (*str*) – (required), name of the repository

**Warning:** this should be no longer than 100 characters

- **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`
- **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_init` is `False`.
- **license\_template** (*str*) – (optional), name of the license; this is ignored if `auto_init` is `False`.

**Returns** the created repository

**Return type** `Repository`

**create\_team** (*name*, *repo\_names=[]*, *permission='pull'*)

Create a new team and return it.

This only works if the authenticated user owns this organization.

#### Parameters

- **name** (*str*) – (required), name to be given to the team
- **repo\_names** (*list*) – (optional) repositories, e.g. [`'github/dotfiles'`]
- **permission** (*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

**Returns** the created team

**Return type** `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** True if successful, False otherwise

**Return type** bool

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

Iterate over public events for this org (deprecated).

Deprecated since version 1.0.0: Use `public_events()` instead.

**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

**Return type** *Event*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from `json_dict`.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from `json`.

**is\_member** (*username*)

Check if the user named `username` is a member.

**Parameters** **username** (*str*) – name of the user you'd like to check

**Returns** True if successful, False otherwise

**Return type** bool

**is\_public\_member** (*username*)

Check if the user named `username` is a public member.

**Parameters** **username** (*str*) – name of the user you'd like to check

**Returns** True if the user is a public member, False otherwise

**Return type** bool

**members** (*filter=None, role=None, number=-1, etag=None*)

Iterate over members of this organization.

**Parameters**

- **filter** (*str*) – (optional), filter members returned by this method. Can be one of: "2fa\_disabled", "all", . Default: "all". Filtering by "2fa\_disabled" is only available for organization owners with private repositories.
- **role** (*str*) – (optional), filter members returned by their role. Can be one of: "all", "admin", "member". 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 members of this organization

**Return type** *ShortUser*

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**project** (*id*, *etag=None*)

Return the organization project with the given ID.

**Parameters** **id** (*int*) – (required), ID number of the project

**Returns** requested project

**Return type** `Project`

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

Iterate over projects for 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 organization projects

**Return type** `Project`

**public\_events** (*number=-1*, *etag=None*)

Iterate over public 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 public events

**Return type** `Event`

**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 public members

**Return type** `ShortUser`

**publicize\_member** (*username*)

Make *username*'s membership in this organization public.

**Parameters** **username** (*str*) – the name of the user whose membership you wish to publicize

**Returns** True if successful, False otherwise

**Return type** `bool`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**remove\_member** (*username*)

Remove the user named *username* from this organization.

**Parameters** **username** (*str*) – name of the user to remove from the org

**Returns** True if successful, False otherwise

**Return type** bool

**remove\_repository** (*repository, team\_id*)

Remove *repository* from the team with *team\_id*.

**Parameters**

- **repository** (*str*) – (required), form: 'user/repo'
- **team\_id** (*int*) – (required), the unique identifier of the team

**Returns** True if successful, False otherwise

**Return type** bool

**repositories** (*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 repositories in this organization

**Return type** Repository

**team** (*team\_id*)

Return the team specified by *team\_id*.

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

**Returns** the team identified by the id in this organization

**Return type** *Team*

**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 this organization's teams

**Return type** *ShortTeam*

## 12.1.10 Pull Requests and their Associated Objects

This section of the documentation covers the representations of various objects related to the [Pull Requests API](#).

### Pull Request Objects

**class** `github3.pulls.ShortPullRequest` (*json, session*)

Object for the shortened representation of a PullRequest.

GitHub's API returns different amounts of information about prs based upon how that information is retrieved. Often times, when iterating over several prs, GitHub will return less information. To provide a clear distinction between the types of Pull Requests, github3.py uses different classes with different sets of attributes.

New in version 1.0.0.

The attributes available on this object are:

**url**

The URL that describes this exact pull request.

**assignee**

Deprecated since version 1.0.0: Use *assignees* instead.

The assignee of the pull request, if present, represented as an instance of *ShortUser*

**assignees**

New in version 1.0.0.

A list of the assignees of the pull request. If not empty, a list of instances of *ShortUser*.

**base**

A *Base* object representing the base pull request destination.

**body**

The Markdown formatted body of the pull request message.

**body\_html**

The HTML formatted body of the pull request message.

**body\_text**

The plain-text formatted body of the pull request message.

**closed\_at**

A *datetime* object representing the date and time when this pull request was closed.

**comments\_url**

The URL to retrieve the comments on this pull request from the API.

**commits\_url**

The URL to retrieve the commits in this pull request from the API.

**created\_at**

A `datetime` object representing the date and time when this pull request was created.

**diff\_url**

The URL to retrieve the diff for this pull request via the API.

**head**

A `Head` object representing the head pull request destination.

**html\_url**

The URL one would use to view this pull request in the browser.

**id**

The unique ID of this pull request across all of GitHub.

**issue\_url**

The URL of the resource that represents this pull request as an issue.

**links**

A dictionary provided by `_links` in the API response.

New in version 1.0.0.

**merge\_commit\_sha**

If unmerged, holds the sha of the commit to test mergability. If merged, holds commit sha of the merge commit, squashed commit on the base branch or the commit that the base branch was updated to after rebasing the PR.

**merged\_at**

A `datetime` object representing the date and time this pull request was merged. If this pull request has not been merged then this attribute will be `None`.

**number**

The number of the pull request on the repository.

**patch\_url**

The URL to retrieve the patch for this pull request via the API.

**repository**

A `ShortRepository` from the `base` instance.

**review\_comment\_urlt**

A `URITemplate` instance that expands to provide the review comment URL provided a number.

**review\_comments\_url**

The URL to retrieve all review comments on this pull request from the API.

**state**

The current state of this pull request.

**title**

The title of this pull request.

**updated\_at**

A `datetime` instance representing the date and time when this pull request was last updated.

**user**

A `ShortUser` instance representing who opened this pull request.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**close()**

Close this Pull Request without merging.

**Returns** True if successful, False otherwise

**Return type** bool

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

Iterate over the commits on this pull request.

**Parameters**

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

**Returns** generator of repository commit objects

**Return type** *ShortCommit*

**create\_comment** (*body*)

Create a comment on this pull request's issue.

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

**Returns** the comment that was created on the pull request

**Return type** *IssueComment*

**create\_review** (*body, commit\_id=None, event=None, comments=None*)

Create a review comment on this pull request.

**Warning:** If you do not specify `event`, GitHub will default it to `PENDING` which means that your review will need to be submitted after creation. (See also `submit()`.)

**Parameters**

- **body** (*str*) – The comment text itself, required when using COMMENT or REQUEST\_CHANGES.
- **commit\_id** (*str*) – The SHA of the commit to comment on
- **event** (*str*) – The review action you want to perform. Actions include APPROVE, REQUEST\_CHANGES or COMMENT. By leaving this blank you set the action to PENDING and will need to submit the review. Leaving blank may result in a 422 error response which will need to be handled.
- **comments** (*list*) – Array of draft review comment objects. Please see Github’s [Create a pull request review documentation](#) for details on review comment objects. At the time of writing these were a dictionary with 3 keys: *path*, *position* and *body*.

**Returns** The created review.

**Return type** PullReview

**create\_review\_comment** (*body*, *commit\_id*, *path*, *position*)

Create a review comment on this pull request.

---

**Note:** All parameters are required by the GitHub API.

---

#### Parameters

- **body** (*str*) – The comment text itself
- **commit\_id** (*str*) – The SHA of the commit to comment on
- **path** (*str*) – The relative path of the file to comment on
- **position** (*int*) – The line index in the diff to comment on.

**Returns** The created review comment.

**Return type** ReviewComment

**diff** ()

Return the diff.

**Returns** representation of the diff

**Return type** bytes

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

Iterate over the files associated with this pull request.

#### Parameters

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

**Returns** generator of pull request files

**Return type** PullFile

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from *json*.

**is\_merged()**

Check to see if the pull request was merged.

Changed in version 1.0.0: This now always makes a call to the GitHub API. To avoid that, check `merged` before making this call.

**Returns** True if merged, False otherwise

**Return type** bool

**issue()**

Retrieve the issue associated with this pull request.

**Returns** the issue object that this pull request builds upon

**Return type** `Issue`

**issue\_comments(number=-1, etag=None)**

Iterate over the issue comments on this pull request.

In this case, GitHub leaks implementation details. Pull Requests are really just Issues with a diff. As such, comments on a pull request that are not in-line with code, are technically issue comments.

**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 non-review comments on this pull request

**Return type** `IssueComment`

**merge(commit\_message=None, sha=None, merge\_method='merge')**

Merge this pull request.

Changed in version 1.0.0: The boolean `squash` parameter has been replaced with `merge_method` which requires a string.

**Parameters**

- **commit\_message** (*str*) – (optional), message to be used for the merge commit
- **sha** (*str*) – (optional), SHA that pull request head must match to merge.
- **merge\_method** (*str*) – (optional), Change the merge method. Either 'merge', 'squash' or 'rebase'. Default is 'merge'.

**Returns** True if successful, False otherwise

**Return type** bool

**Returns** bool

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**patch()**

Return the patch.

**Returns** bytestring representation of the patch

**Return type** bytes



**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**reopen** ()

Re-open a closed Pull Request.

**Returns** True if successful, False otherwise

**Return type** bool

**review\_comments** (*number=-1, etag=None*)

Iterate over the review 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 review comments

**Return type** *ReviewComment*

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

Iterate over the reviews associated with this pull request.

**Parameters**

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

**Returns** generator of reviews for this pull request

**Return type** *PullReview*

**update** (*title=None, body=None, state=None, base=None, maintainer\_can\_modify=None*)

Update this pull request.

**Parameters**

- **title** (*str*) – (optional), title of the pull
- **body** (*str*) – (optional), body of the pull request
- **state** (*str*) – (optional), one of ('open', 'closed')
- **base** (*str*) – (optional), Name of the branch on the current repository that the changes should be pulled into.
- **maintainer\_can\_modify** (*bool*) – (optional), Indicates whether a maintainer is allowed to modify the pull request or not.

**Returns** True if successful, False otherwise

**Return type** bool

**class** github3.pulls.**PullRequest** (*json, session*)

Object for the full representation of a PullRequest.

GitHub's API returns different amounts of information about prs based upon how that information is retrieved. This object exists to represent the full amount of information returned for a specific pr. For example, you would receive this class when calling `pull_request()`. To provide a clear distinction between the types of prs, github3.py uses different classes with different sets of attributes.

Changed in version 1.0.0.

This object has all of the same attributes as `ShortPullRequest` as well as the following:

**additions\_count**

The number of lines of code added in this pull request.

**deletions\_count**

The number of lines of code deleted in this pull request.

**comments\_count**

The number of comments left on this pull request.

**commits\_count**

The number of commits included in this pull request.

**mergeable**

A boolean attribute indicating whether GitHub deems this pull request is mergeable.

**mergeable\_state**

A string indicating whether this would be a 'clean' or 'dirty' merge.

**merged**

A boolean attribute indicating whether the pull request has been merged or not.

**merged\_by**

An instance of `ShortUser` to indicate the user who merged this pull request. If this hasn't been merged or if `mergeable` is still being decided by GitHub this will be `None`.

**review\_comments\_count**

The number of review comments on this pull request.

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**close()**

Close this Pull Request without merging.

**Returns** True if successful, False otherwise

**Return type** bool

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

Iterate over the commits on this pull request.

**Parameters**

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

**Returns** generator of repository commit objects

**Return type** *ShortCommit*

**create\_comment** (*body*)

Create a comment on this pull request's issue.

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

**Returns** the comment that was created on the pull request

**Return type** *IssueComment*

**create\_review** (*body, commit\_id=None, event=None, comments=None*)

Create a review comment on this pull request.

**Warning:** If you do not specify `event`, GitHub will default it to `PENDING` which means that your review will need to be submitted after creation. (See also `submit()`.)

**Parameters**

- **body** (*str*) – The comment text itself, required when using `COMMENT` or `REQUEST_CHANGES`.
- **commit\_id** (*str*) – The SHA of the commit to comment on
- **event** (*str*) – The review action you want to perform. Actions include `APPROVE`, `REQUEST_CHANGES` or `COMMENT`. By leaving this blank you set the action to `PENDING` and will need to submit the review. Leaving blank may result in a 422 error response which will need to be handled.

- **comments** (*list*) – Array of draft review comment objects. Please see Github’s [Create a pull request review documentation](#) for details on review comment objects. At the time of writing these were a dictionary with 3 keys: *path*, *position* and *body*.

**Returns** The created review.

**Return type** *PullReview*

**create\_review\_comment** (*body, commit\_id, path, position*)

Create a review comment on this pull request.

---

**Note:** All parameters are required by the GitHub API.

---

#### Parameters

- **body** (*str*) – The comment text itself
- **commit\_id** (*str*) – The SHA of the commit to comment on
- **path** (*str*) – The relative path of the file to comment on
- **position** (*int*) – The line index in the diff to comment on.

**Returns** The created review comment.

**Return type** *ReviewComment*

**diff** ()

Return the diff.

**Returns** representation of the diff

**Return type** bytes

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

Iterate over the files associated with this pull request.

#### Parameters

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

**Returns** generator of pull request files

**Return type** *PullFile*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**is\_merged** ()

Check to see if the pull request was merged.

Changed in version 1.0.0: This now always makes a call to the GitHub API. To avoid that, check *merged* before making this call.

**Returns** True if merged, False otherwise

**Return type** bool

**issue()**

Retrieve the issue associated with this pull request.

**Returns** the issue object that this pull request builds upon

**Return type** `Issue`

**issue\_comments** (*number=-1, etag=None*)

Iterate over the issue comments on this pull request.

In this case, GitHub leaks implementation details. Pull Requests are really just Issues with a diff. As such, comments on a pull request that are not in-line with code, are technically issue comments.

**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 non-review comments on this pull request

**Return type** `IssueComment`

**merge** (*commit\_message=None, sha=None, merge\_method='merge'*)

Merge this pull request.

Changed in version 1.0.0: The boolean `squash` parameter has been replaced with `merge_method` which requires a string.

**Parameters**

- **commit\_message** (*str*) – (optional), message to be used for the merge commit
- **sha** (*str*) – (optional), SHA that pull request head must match to merge.
- **merge\_method** (*str*) – (optional), Change the merge method. Either 'merge', 'squash' or 'rebase'. Default is 'merge'.

**Returns** True if successful, False otherwise

**Return type** `bool`

**Returns** `bool`

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**patch()**

Return the patch.

**Returns** bytestring representation of the patch

**Return type** `bytes`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**reopen** ()

Re-open a closed Pull Request.

**Returns** True if successful, False otherwise

**Return type** bool

**review\_comments** (*number=-1, etag=None*)

Iterate over the review 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 review comments

**Return type** *ReviewComment*

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

Iterate over the reviews associated with this pull request.

**Parameters**

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

**Returns** generator of reviews for this pull request

**Return type** *PullReview*

**update** (*title=None, body=None, state=None, base=None, maintainer\_can\_modify=None*)

Update this pull request.

**Parameters**

- **title** (*str*) – (optional), title of the pull
- **body** (*str*) – (optional), body of the pull request
- **state** (*str*) – (optional), one of ('open', 'closed')
- **base** (*str*) – (optional), Name of the branch on the current repository that the changes should be pulled into.

- **maintainer\_can\_modify** (*bool*) – (optional), Indicates whether a maintainer is allowed to modify the pull request or not.

**Returns** True if successful, False otherwise

**Return type** bool

**class** github3.pulls.**PullDestination** (*json, session*)

The object that represents a pull request destination.

This is the base class for the *Head* and *Base* objects. Each has identical attributes to this object.

Please see GitHub’s [Pull Request Documentation](#) for more information.

**ref**

The full reference string for the git object

**label**

The label for the destination (e.g., ‘master’, ‘mybranch’)

**user**

If provided, a *ShortUser* instance representing the owner of the destination

**sha**

The SHA of the commit at the head of the destination

**repository**

A *ShortRepository* representing the repository containing this destination

**repo**

A tuple containing the login and repository name, e.g., (‘sigmavirus24’, ‘github3.py’)

This attribute is generated by github3.py and may be deprecated in the future.

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object’s attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object’s attributes as a JSON string

**Return type** str

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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`

**class** `github3.pulls.Head` (*json, session*)

An object representing the Head destination of a pull request.

See [PullDestination](#) for more details.

**class** `github3.pulls.Base` (*json, session*)

An object representing the Base destination of a pull request.

See [PullDestination](#) for more details.

**class** `github3.pulls.PullFile` (*json, session*)

The object that represents a file in a pull request.

Please see GitHub's [Pull Request Files Documentation](#) for more information.

**additions\_count**

The number of additions made to this file

**blob\_url**

The API resource used to retrieve the blob for this file

**changes\_count**

The number of changes made to this file

**contents\_url**

The API resource to view the raw contents of this file

**deletions\_count**

The number of deletions made to this file



**filename**

The name of this file

**patch**

The patch generated by this

---

**Note:** If the patch is larger than a specific size it may be missing from GitHub's response. The attribute will be set to `None` in this case.

---

**raw\_url**

The API resource to view the raw diff of this file

**sha**

The SHA of the commit that this file belongs to

**status**

The string with the status of the file, e.g., 'added'

**as\_dict ()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json ()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**contents ()**

Return the contents of the file.

**Returns** An object representing the contents of this file

**Return type** *Contents*

**classmethod from\_dict (json\_dict, session)**

Return an instance of this class formed from `json_dict`.

**classmethod from\_json (json, session)**

Return an instance of this class formed from `json`.

**new\_session ()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

## Review Objects

**class** github3.pulls.**ReviewComment** (*json, session*)

Object representing review comments left on a pull request.

Please see GitHub's [Pull Comments Documentation](#) for more information.

**id**

The unique identifier for this comment across all GitHub review comments.

**author\_association**

The role of the author of this comment on the repository.

**body**

The Markdown formatted body of this comment.

**body\_html**

The HTML formatted body of this comment.

**body\_text**

The plain text formatted body of this comment.

**commit\_id**

The SHA of current commit this comment was left on.

**created\_at**

A `datetime` instance representing the date and time this comment was created.

**diff\_hunk**

A string representation of the hunk of the diff where the comment was left.

**html\_url**

The URL to view this comment in the webbrowser.

**links**

A dictionary of relevant URLs usually returned in the `_links` attribute.

**original\_commit\_id**

The SHA of the original commit this comment was left on.

**original\_position**

The original position within the diff that this comment was left on.

**pull\_request\_url**

The URL to retrieve the pull request via the API.

**updated\_at**

A `datetime` instance representing the date and time this comment was updated.

**user**

A `ShortUser` instance representing the author of this comment.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete this comment.

**Returns** True if successful, False otherwise

**Return type** bool

**edit(body)**

Edit this comment.

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

**Returns** True if successful, False otherwise

**Return type** bool

**classmethod from\_dict(json\_dict, session)**

Return an instance of this class formed from `json_dict`.

**classmethod from\_json(json, session)**

Return an instance of this class formed from `json`.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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`

**reply** (*body*)

Reply to this review comment with a new review comment.

**Parameters** **body** (*str*) – The text of the comment.

**Returns** The created review comment.

**Return type** `ReviewComment`

## 12.1.11 Repository API Objects

This section of the documentation covers the representations of various objects related to the [Repositories API](#).

### Repository Objects

**class** `github3.repos.repo.Repository` (*json, session*)

This organizes the full representation of a single Repository.

The full representation of a Repository is not returned in collections but instead in individual requests, e.g., `repository()`.

This object has all the same attributes as `ShortRepository` as well as:

**allow\_merge\_commit**

---

**Note:** This attribute is not guaranteed to be present.

---

Whether the repository allows creating a merge commit when merging when a pull request.

#### **allow\_rebase\_merge**

---

**Note:** This attribute is not guaranteed to be present.

---

Whether the repository allows rebasing when merging a pull request.

#### **allow\_squash\_merge**

---

**Note:** This attribute is not guaranteed to be present.

---

Whether the repository allows squashing commits when merging a pull request.

#### **archived**

A boolean attribute that describes whether the current repository has been archived or not.

#### **clone\_url**

This is the URL that can be used to clone the repository via HTTPS, e.g., `https://github.com/sigmavirus24/github3.py.git`.

#### **created\_at**

A parsed `datetime` object representing the date the repository was created.

#### **default\_branch**

This is the default branch of the repository as configured by its administrator(s).

#### **forks\_count**

This is the number of forks of the repository.

#### **git\_url**

This is the URL that can be used to clone the repository via the Git protocol, e.g., `git://github.com/sigmavirus24/github3.py`.

#### **has\_downloads**

This is a boolean attribute that conveys whether or not the repository has downloads.

#### **has\_issues**

This is a boolean attribute that conveys whether or not the repository has issues.

#### **has\_pages**

This is a boolean attribute that conveys whether or not the repository has pages.

#### **has\_wiki**

This is a boolean attribute that conveys whether or not the repository has a wiki.

#### **homepage**

This is the administrator set homepage URL for the project. This may not be provided.

#### **language**

This is the language GitHub has detected for the repository.

#### **original\_license**

---

**Note:** When used with a Github Enterprise instance  $\leq 2.12.7$ , this attribute will not be returned. To handle these situations sensitively, the attribute will be set to `None`. Repositories may still have a license associated with them in these cases.

---

This is the `ShortLicense` returned as part of the repository. To retrieve the most recent license, see the `license()` method.

**mirror\_url**

The URL that GitHub is mirroring the repository from.

**network\_count**

The size of the repository's "network".

**open\_issues\_count**

The number of issues currently open on the repository.

**parent**

A representation of the parent repository as `ShortRepository`. If this Repository has no parent then this will be `None`.

**pushed\_at**

A parsed `datetime` object representing the date a push was last made to the repository.

**size**

The size of the repository.

**source**

A representation of the source repository as `ShortRepository`. If this Repository has no source then this will be `None`.

**ssh\_url**

This is the URL that can be used to clone the repository via the SSH protocol, e.g., `ssh@github.com:sigmavirus24/github3.py.git`.

**stargazers\_count**

The number of people who have starred this repository.

**subscribers\_count**

The number of people watching (or who have subscribed to notifications about) this repository.

**svn\_url**

This is the URL that can be used to clone the repository via SVN, e.g., `ssh@github.com:sigmavirus24/github3.py.git`.

**updated\_at**

A parsed `datetime` object representing the date a the repository was last updated by its administrator(s).

**watchers\_count**

The number of people watching this repository.

See also: <http://developer.github.com/v3/repos/>

**add\_collaborator** (*username*)

Add *username* as a collaborator to a repository.

**Parameters** *username* (str or `User`) – (required), username of the user

**Returns** True if successful, False otherwise

**Return type**

**archive** (*format*, *path=""*, *ref='master'*)

Get the tarball or zipball archive for this repo at ref.

See: <http://developer.github.com/v3/repos/contents/#get-archive-link>

**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** True if successful, False otherwise

**Return type** bool

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**asset** (*id*)

Return a single asset.

**Parameters** **id** (*int*) – (required), id of the asset

**Returns** the asset

**Return type** *Asset*

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

Iterate over all 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** generator of users

**Return type** *ShortUser*

**blob** (*sha*)

Get the blob indicated by *sha*.

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

**Returns** the git blob

**Return type** *Blob*

**branch** (*name*)

Get the branch *name* of this repository.

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

**Returns** the branch

**Return type** *Branch*

**branches** (*number=-1, protected=False, etag=None*)

Iterate over the branches in this repository.

**Parameters**

- **number** (*int*) – (optional), number of branches to return. Default: -1 returns all branches
- **protected** (*bool*) – (optional), True lists only protected branches. Default: False
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of branches

**Return type** *Branch*

**code\_frequency** (*number=-1, etag=None*)

Iterate over the code frequency per week.

New in version 0.7.

Returns a weekly aggregate of the number of additions and deletions pushed to this repository.

---

**Note:** All statistics methods may return a 202. On those occasions, you will not receive any objects. You should store your iterator and check the new `last_status` attribute. If it is a 202 you should wait before re-requesting.

---

**Parameters**

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

**Returns** generator of lists [`seconds_from_epoch`, `additions`, `deletions`]

**Return type** list

**collaborators** (*affiliation='all', number=-1, etag=None*)

Iterate over the collaborators of this repository.

**Parameters**

- **affiliation** (*str*) – (optional), affiliation of the collaborator to the repository. Default: “all” returns contributors with all affiliations



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

**Returns** generator of collaborators

**Return type** *Collaborator*

**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** generator of comments on commits

**Return type** *RepoComment*

**commit** (*sha*)

Get a single (repo) commit.

See `git_commit()` for the Git Data Commit.

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

**Returns** the commit

**Return type** *RepoCommit*

**commit\_activity** (*number=-1, etag=None*)

Iterate over last year of commit activity by week.

New in version 0.7.

See: <http://developer.github.com/v3/repos/statistics/>

---

**Note:** All statistics methods may return a 202. On those occasions, you will not receive any objects. You should store your iterator and check the new `last_status` attribute. If it is a 202 you should wait before re-requesting.

---

**Parameters**

- **number** (*int*) – (optional), number of weeks to return. Default -1 will return all of the weeks.
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of dictionaries

**Return type** dict

**commit\_comment** (*comment\_id*)

Get a single commit comment.

**Parameters** **comment\_id** (*int*) – (required), id of the comment used by GitHub

**Returns** the comment on the commit

**Return type** *RepoComment*

**commits** (*sha=None, path=None, author=None, number=-1, etag=None, since=None, until=None, per\_page=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
- **since** (*datetime* or *str*) – (optional), Only commits after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string.
- **until** (*datetime* or *str*) – (optional), Only commits before this date will be returned. This can be a *datetime* or an ISO8601 formatted date string.
- **per\_page** (*int*) – (optional), commits listing page size

**Returns** generator of commits

**Return type** *RepoCommit*

**compare\_commits** (*base, head*)

Compare two commits.

**Parameters**

- **base** (*str*) – (required), base for the comparison
- **head** (*str*) – (required), compare this against base

**Returns** the comparison of the commits

**Return type** *Comparison*

**contributor\_statistics** (*number=-1, etag=None*)

Iterate over the contributors list.

New in version 0.7.

See also: <http://developer.github.com/v3/repos/statistics/>

---

**Note:** All statistics methods may return a 202. On those occasions, you will not receive any objects. You should store your iterator and check the new `last_status` attribute. If it is a 202 you should wait before re-requesting.

---

**Parameters**

- **number** (*int*) – (optional), number of weeks to return. Default -1 will return all of the weeks.
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of contributor statistics for each contributor

**Return type** *ContributorStats*

**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** generator of contributor users

**Return type** *Contributor*

**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

**Returns** str (on Python 3, unicode on Python 2)

**create\_branch\_ref** (*name, sha=None*)

Create a branch git reference.

This is a shortcut for calling `github3.repos.repo.Repository.create_ref()`.

**Parameters**

- **branch** (*str*) – (required), the branch to create
- **sha** (*str*) – the commit to base the branch from

**Returns** a reference object representing the branch

**Return type** *Reference*

**create\_comment** (*body, sha, path=None, position=None, 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** the created comment

**Return type** *RepoComment*

**create\_commit** (*message, tree, parents, author=None, committer=None*)

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 omitted, GitHub will use the author parameters. Should be the same format as the author parameter.

**Returns** the created commit

**Return type** *Commit*

**create\_deployment** (*ref*, *required\_contexts=None*, *payload=""*, *auto\_merge=False*, *description=""*, *environment=None*)

Create a deployment.

#### Parameters

- **ref** (*str*) – (required), The ref to deploy. This can be a branch, tag, or sha.
- **required\_contexts** (*list*) – Optional array of status contexts verified against commit status checks. To bypass checking entirely pass an empty array. Default: []
- **payload** (*str*) – Optional JSON payload with extra information about the deployment. Default: ""
- **auto\_merge** (*bool*) – Optional parameter to merge the default branch into the requested deployment branch if necessary. Default: False
- **description** (*str*) – Optional short description. Default: ""
- **environment** (*str*) – Optional name for the target deployment environment (e.g., production, staging, qa). Default: “production”

**Returns** the created deployment

**Return type** *Deployment*

**create\_file** (*path*, *message*, *content*, *branch=None*, *committer=None*, *author=None*)

Create a file in this repository.

See also: <http://developer.github.com/v3/repos/contents/#create-a-file>

#### Parameters

- **path** (*str*) – (required), path of the file in the repository
- **message** (*str*) – (required), commit message
- **content** (*bytes*) – (required), the actual data in the file
- **branch** (*str*) – (optional), branch to create the commit on. Defaults to the default branch of the repository
- **committer** (*dict*) – (optional), if no information is given the authenticated user’s information will be used. You must specify both a name and email.
- **author** (*dict*) – (optional), if omitted this will be filled in with committer information. If passed, you must specify both a name and email.

**Returns** dictionary of contents and commit for created file

**Return type** *Contents, Commit*

**create\_fork** (*organization=None*)

Create a fork of this repository.

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

**Returns** the fork of this repository

**Return type** *Repository*

**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** the created hook

**Return type** *Hook*

**create\_issue** (*title, body=None, assignee=None, milestone=None, labels=None, assignees=None*)

Create 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** (*[str]*) – (optional), labels to apply to this issue
- **assignees** (*[str]*) – (optional), login of the users to assign the issue to

**Returns** the created issue

**Return type** *ShortIssue*

**create\_key** (*title, key, read\_only=False*)

Create a deploy key.

**Parameters**

- **title** (*str*) – (required), title of key
- **key** (*str*) – (required), key text
- **read\_only** (*bool*) – (optional), restrict key access to read-only, default is False

**Returns** the created key

**Return type** *Key*

**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, e.g., '#fafafa' or 'fafafa' (the latter is what is sent)

**Returns** the created label

**Return type** *Label*

**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** the created milestone

**Return type** *Milestone*

**create\_project** (*name, body=None*)

Create a project for this repository.

**Parameters**

- **name** (*str*) – (required), name of the project
- **body** (*str*) – (optional), body of the project

**Returns** the created project

**Return type** *Project*

**create\_pull** (*title, base, head, body=None*)

Create a pull request of head onto base branch in this repo.

**Parameters**

- **title** (*str*) – (required)
- **base** (*str*) – (required), e.g., 'master'
- **head** (*str*) – (required), e.g., 'username:branch'
- **body** (*str*) – (optional), markdown formatted description

**Returns** the created pull request

**Return type** *ShortPullRequest*

**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., 'master'
- **head** (*str*) – (required), e.g., 'username:branch'

**Returns** the created pull request

**Return type** *ShortPullRequest*

**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** the created ref

**Return type** *Reference*

**create\_release** (*tag\_name*, *target\_commitish=None*, *name=None*, *body=None*, *draft=False*, *prerelease=False*)

Create a release for this repository.

**Parameters**

- **tag\_name** (*str*) – (required), name to give to the tag
- **target\_commitish** (*str*) – (optional), vague concept of a target, either a SHA or a branch name.
- **name** (*str*) – (optional), name of the release
- **body** (*str*) – (optional), description of the release
- **draft** (*bool*) – (optional), whether this release is a draft or not
- **prerelease** (*bool*) – (optional), whether this is a prerelease or not

**Returns** the created release

**Return type** *Release*

**create\_status** (*sha*, *state*, *target\_url=None*, *description=None*, *context='default'*)

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
- **context** (*str*) – (optional), A string label to differentiate this status from the status of other systems

**Returns** the created status

**Return type** *Status*

**create\_tag** (*tag*, *message*, *sha*, *obj\_type*, *tagger*, *lightweight=False*)

Create a tag in this repository.

By default, this method creates an annotated tag. If you wish to create a lightweight tag instead, pass `lightweight=True`.

If you are creating an annotated tag, this method makes **2 calls** to the API:

1. Creates the tag object
2. Creates the reference for the tag

This behaviour is required by the GitHub API.

#### 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 creating a lightweight tag, this will return a *Reference*, otherwise it will return a *Tag*

**Return type** *Tag* or *Reference*

**create\_tree** (*tree*, *base\_tree=None*)

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** the created tree

**Return type** *Tree*

**delete** ()

Delete this repository.

**Returns** True if successful, False otherwise

**Return type** bool

**delete\_key** (*key\_id*)

Delete the key with the specified id from your deploy keys list.

**Returns** True if successful, False otherwise

**Return type** bool

**delete\_subscription** ()

Delete the user’s subscription to this repository.

**Returns** True if successful, False otherwise

**Return type** bool

**deployment** (*id*)

Retrieve the deployment identified by *id*.



**Parameters** `id` (*int*) – (required), id for deployments.

**Returns** the deployment

**Return type** *Deployment*

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

Iterate over deployments for this repository.

**Parameters**

- **number** (*int*) – (optional), number of deployments to return. Default: -1, returns all available deployments
- **etag** (*str*) – (optional), ETag from a previous request for all deployments

**Returns** generator of deployments

**Return type** *Deployment*

**directory\_contents** (*directory\_path, ref=None, return\_as=<class 'list'>*)

Get the contents of each file in `directory_path`.

If the path provided is actually a directory, you will receive a list back of the form:

```
[('filename.md', Contents(...)),
 ('github.py', Contents(...)),
 # ...
 ('fiz.py', Contents(...))]
```

You can either then transform it into a dictionary:

```
contents = dict(repo.directory_contents('path/to/dir/'))
```

Or you can use the `return_as` parameter to have it return a dictionary for you:

```
contents = repo.directory_contents('path/to/dir/', return_as=dict)
```

**Parameters**

- **path** (*str*) – (required), path to file, e.g. `github3/repos/repo.py`
- **ref** (*str*) – (optional), the string name of a commit/branch/tag. Default: `master`
- **return\_as** – (optional), how to return the directory's contents. Default: `list`

**Returns** list of tuples of the filename and the Contents returned

**Return type** [(*str*, *Contents*)]

**Raises** `github3.exceptions.UnprocessableResponseBody` – When the requested directory is not actually a directory

**edit** (*name, description=None, homepage=None, private=None, has\_issues=None, has\_wiki=None, has\_downloads=None, default\_branch=None, archived=None, allow\_merge\_commit=None, allow\_squash\_merge=None, allow\_rebase\_merge=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.
- **archived** (*bool*) – (optional), If not `None`, toggle the archived attribute on the repository to control whether it is archived or not.
- **allow\_rebase\_merge** (*bool*) – (optional), If not `None`, change whether the merge strategy that allows adding all commits from the head branch onto the base branch individually is enabled for this repository. API default: `None` - leave value unchanged.
- **allow\_squash\_merge** (*bool*) – (optional), If not `None`, change whether combining all commits from the head branch into a single commit in the base branch is allowed. API default: `None` - leave value unchanged.
- **allow\_merge\_commit** (*bool*) – (optional), If not `None`, change whether adding all commits from the head branch to the base branch with a merge commit is allowed. API default: `None` - leave value unchanged.

**Returns** True if successful, False otherwise

**Return type** bool

**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** generator of events

**Return type** *Event*

**file\_contents** (*path, ref=None*)

Get the contents of the file pointed to by `path`.

**Parameters**

- **path** (*str*) – (required), path to file, e.g. `github3/repos/repo.py`
- **ref** (*str*) – (optional), the string name of a commit/branch/tag. Default: `master`

**Returns** the contents of the file requested

**Return type** *Contents*

**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** generator of forks of this repository

**Return type** *ShortRepository*

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from *json*.

**git\_commit** (*sha*)

Get a single (git) commit.

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

**Returns** the single commit data from git

**Return type** *Commit*

**hook** (*hook\_id*)

Get a single hook.

**Parameters** **hook\_id** (*int*) – (required), id of the hook

**Returns** the hook

**Return type** *Hook*

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

Iterate over hooks registered on this repository.

**Parameters**

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

**Returns** generator of hooks

**Return type** *Hook*

**ignore** ()

Ignore notifications from this repository for the user.

New in version 1.0.

This replaces `Repository#set_subscription`.

**Returns** the new repository subscription

**Return type** :class:`~github3.notifications.RepositorySubscription`

**import\_issue** (*title, body, created\_at, assignee=None, milestone=None, closed=None, labels=None, comments=None*)

Import an issue into the repository.

See also: <https://gist.github.com/jonmagic/5282384165e0f86ef105>

#### Parameters

- **title** (*string*) – (required) Title of issue
- **body** (*string*) – (required) Body of issue
- **created\_at** (*datetime* or *str*) – (required) Creation timestamp
- **assignee** (*string*) – (optional) Username to assign issue to
- **milestone** (*int*) – (optional) Milestone ID
- **closed** (*boolean*) – (optional) Status of issue is Closed if True
- **labels** (*list*) – (optional) List of labels containing string names
- **comments** (*list*) – (optional) List of dictionaries which contain *created\_at* and *body* attributes

**Returns** the imported issue

**Return type** *ImportedIssue*

**imported\_issue** (*imported\_issue\_id*)

Retrieve imported issue specified by imported issue id.

**Parameters** **imported\_issue\_id** (*int*) – (required) id of imported issue

**Returns** the imported issue

**Return type** *ImportedIssue*

**imported\_issues** (*number=-1, since=None, etag=None*)

Retrieve the collection of imported issues via the API.

See also: <https://gist.github.com/jonmagic/5282384165e0f86ef105>

#### Parameters

- **number** (*int*) – (optional), number of imported issues to return. Default: -1 returns all branches
- **since** – (optional), Only imported issues after this date will be returned. This can be a *datetime* instance, ISO8601 formatted date string, or a string formatted like so: 2016-02-04 i.e. %Y-%m-%d
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of imported issues

**Return type** *ImportedIssue*

**is\_assignee** (*username*)

Check if the user can be assigned an issue on this repository.

**Parameters** **username** (*str* or *User*) – name of the user to check

**Returns** bool

**is\_collaborator** (*username*)

Check to see if *username* is a collaborator on this repository.

**Parameters** **username** (*str* or *User*) – (required), login for the user

**Returns** True if successful, False otherwise

**Return type** bool

**issue** (*number*)

Get the issue specified by *number*.

**Parameters** **number** (*int*) – (required), number of the issue on this repository

**Returns** the issue

**Return type** *Issue*

**issue\_events** (*number=-1, etag=None*)

Iterate 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 events on issues

**Return type** *IssueEvent*

**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.

Changed in version 0.9.0: The *state* parameter now accepts ‘all’ in addition to ‘open’ and ‘closed’.

**Parameters**

- **milestone** (*int*) – (optional), ‘none’, or ‘\*’
- **state** (*str*) – (optional), accepted values: (‘all’, ‘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** – (optional), accepted values: (‘created’, ‘updated’, ‘comments’, ‘created’)
- **direction** (*str*) – (optional), accepted values: (‘asc’, ‘desc’)
- **since** (*datetime* or *str*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, e.g., 2012-05-20T23:10:27Z
- **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** generator of issues

**Return type** *ShortIssue*

**key** (*id\_num*)

Get the specified deploy key.

**Parameters** **id\_num** (*int*) – (required), id of the key

**Returns** the deploy key

**Return type** *Key*

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

Iterate 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

**Return type** *Key*

**label** (*name*)

Get the label specified by *name*.

**Parameters** **name** (*str*) – (required), name of the label

**Returns** the label

**Return type** *Label*

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

Iterate 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

**Return type** *Label*

**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** generator of tuples

**Return type** tuple

**latest\_pages\_build** ()

Get the build information for the most recent Pages build.

**Returns** the information for the most recent build

**Return type** *PagesBuild*

**latest\_release** ()

Get the latest release.

Draft releases and prereleases are not returned by this endpoint.

**Returns** the release

**Return type** *Release*

**license** ()

Get the contents of a license for the repo.

**Returns** the license

**Return type** `RepositoryLicense`

**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** True if successful, False otherwise

**Return type** `bool`

**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** the commit resulting from the merge

**Return type** `RepoCommit`

**milestone** (*number*)

Get the milestone indicated by number.

**Parameters** **number** (*int*) – (required), unique id number of the milestone

**Returns** the milestone

**Return type** `Milestone`

**milestones** (*state=None, sort=None, direction=None, number=-1, etag=None*)

Iterate 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

**Return type** `Milestone`

**network\_events** (*number=-1, etag=None*)

Iterate 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

**Return type** *Event*

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**notifications** (*all=False, participating=False, since=None, number=-1, etag=None*)

Iterate 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** (*datetime* or *str*) – (optional), filters out any notifications updated before the given time. This can be a *datetime* or an *ISO8601* formatted date string, e.g., 2012-05-20T23:10:27Z
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of notification threads

**Return type** *Thread*

**pages** ()

Get information about this repository's pages site.

**Returns** information about this repository's GitHub pages site

**Return type** *PagesInfo*

**pages\_builds** (*number=-1, etag=None*)

Iterate over pages builds of this repository.

**Parameters**

- **number** (*int*) – (optional) the number of builds to return
- **etag** (*str*) – (optional), ETag value from a previous request

**Returns** generator of builds

**Return type** *PagesBuild*

**project** (*id, etag=None*)

Return the organization project with the given ID.

**Parameters** **id** (*int*) – (required), ID number of the project

**Returns** the project

**Return type** *Project*

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

Iterate over projects for 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 projects

**Return type** *Project*

**pull\_request** (*number*)

Get the pull request indicated by *number*.

**Parameters** **number** (*int*) – (required), number of the pull request.

**Returns** the pull request

**Return type** *PullRequest*

**pull\_requests** (*state=None, head=None, base=None, sort='created', direction='desc', number=-1, etag=None*)

List pull requests on repository.

Changed in version 0.9.0: The *state* parameter now accepts 'all' in addition to 'open' and 'closed'. The *sort* parameter was added. The *direction* parameter was added.

- **Parameters**

- **state** (*str*) – (optional), accepted values: ('all', 'open', 'closed')
- **head** (*str*) – (optional), filters pulls by head user and branch name in the format `user:ref-name`, e.g., `seveas:debian`
- **base** (*str*) – (optional), filter pulls by base branch name. Example: `develop`.
- **sort** (*str*) – (optional), Sort pull requests by `created`, `updated`, `popularity`, `long-running`. Default: 'created'
- **direction** (*str*) – (optional), Choose the direction to list pull requests. Accepted values: ('desc', 'asc'). Default: 'desc'
- **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 pull requests

**Return type** *ShortPullRequest*

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** *int*

**readme** ()

Get the README for this repository.

**Returns** this repository's readme

**Return type** *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** the reference

**Return type** *Reference*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**refs** (*subspace=”, number=-1, etag=None*)

Iterate 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

**Return type** *Reference*

**release** (*id*)

Get a single release.

**Parameters** **id** (*int*) – (required), id of release

**Returns** the release

**Return type** *Release*

**release\_from\_tag** (*tag\_name*)

Get a release by tag name.

release\_from\_tag() returns a release with specified tag while release() returns a release with specified release id

**Parameters** **tag\_name** (*str*) – (required) name of tag

**Returns** the release

**Return type** *Release*

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

Iterate over releases for this repository.

**Parameters**

- **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 releases

**Return type** *Release*

**remove\_collaborator** (*username*)

Remove collaborator *username* from the repository.

**Parameters** **username** (*str* or *User*) – (required), login name of the collaborator

**Returns** True if successful, False otherwise

**Return type** *bool*

**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

**Return type** *ShortUser*

**statuses** (*sha, number=-1, etag=None*)

Iterate over the statuses for a specific SHA.

**Warning:** Deprecated in v1.0. Also deprecated upstream <https://developer.github.com/v3/repos/statuses/>

**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 statuses

**Return type** *Status*

**subscribe** ()

Subscribe the user to this repository's notifications.

New in version 1.0.

This replaces `Repository#set_subscription`

**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** the new repository subscription

**Return type** *RepositorySubscription*

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

Iterate 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 users subscribed to this repository

**Return type** *ShortUser*

**subscription** ()

Return subscription for this Repository.

**Returns** the user's subscription to this repository

**Return type** *RepositorySubscription*

**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** the annotated tag

**Return type** *Tag*

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

Iterate 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 tags with GitHub repository specific information

**Return type** *RepoTag*

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

Iterate 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

**Return type** *Team*

**tree** (*sha*)

Get a tree.

**Parameters** **sha** (*str*) – (required), sha of the object for this tree

**Returns** the tree

**Return type** *Tree*

**weekly\_commit\_count** ()

Retrieve the total commit counts.

---

**Note:** All statistics methods may return a 202. If github3.py receives a 202 in this case, it will return an empty dictionary. You should give the API a moment to compose the data and then re-request it via this method.

---

..versionadded:: 0.7

The dictionary returned has two entries: `all` and `owner`. Each has a fifty-two element long list of commit counts. (Note: `all` includes the owner.) `d['all'][0]` will be the oldest week, `d['all'][51]` will be the most recent.

**Returns** the commit count as a dictionary

**Return type** dict

**class** github3.repos.repo.**ShortRepository** (*json, session*)

This represents a Repository object returned in collections.

GitHub's API returns different amounts of information about repositories based upon how that information is retrieved. This object exists to represent the full amount of information returned for a specific repository. For example, you would receive this class when calling `repository()`. To provide a clear distinction between the types of repositories, github3.py uses different classes with different sets of attributes.

This object only has the following attributes:

**url**

The GitHub API URL for this repository, e.g., `https://api.github.com/repos/sigmavirus24/github3.py`.

**archive\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `archive_urlt.variables` for the list of variables that can be passed to `archive_urlt.expand()`.

**assignees\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `assignees_urlt.variables` for the list of variables that can be passed to `assignees_urlt.expand()`.

**blobs\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `blobs_urlt.variables` for the list of variables that can be passed to `blobs_urlt.expand()`.

**branches\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check

`branches_urlt.variables` for the list of variables that can be passed to `branches_urlt.expand()`.

**collaborators\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `collaborators_urlt.variables` for the list of variables that can be passed to `collaborators_urlt.expand()`.

**comments\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `comments_urlt.variables` for the list of variables that can be passed to `comments_urlt.expand()`.

**commits\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `commits_urlt.variables` for the list of variables that can be passed to `commits_urlt.expand()`.

**compare\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `compare_urlt.variables` for the list of variables that can be passed to `compare_urlt.expand()`.

**contents\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `contents_urlt.variables` for the list of variables that can be passed to `contents_urlt.expand()`.

**contributors\_url**

The URL to retrieve this repository's list of contributors.

**deployments\_url**

The URL to retrieve this repository's list of deployments.

**description**

The administrator created description of the repository.

**downloads\_url**

The URL to retrieve this repository's list of downloads.

**events\_url**

The URL to retrieve this repository's list of events.

**fork**

Whether or not this repository is a fork of another.

**forks\_url**

The URL to retrieve this repository's list of forks.

**full\_name**

The full name of this repository, e.g., `sigmavirus24/github3.py`.

**git\_commits\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `git_commits_urlt.variables` for the list of variables that can be passed to `git_commits_urlt.expand()`.

**git\_refs\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `git_refs_urlt.variables` for the list of variables that can be passed to `git_refs_urlt.expand()`.

**git\_tags\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `git_tags_urlt.variables` for the list of variables that can be passed to `git_tags_urlt.expand()`.

**hooks\_url**

The URL to retrieve this repository's list of hooks.

**html\_url**

The HTML URL of this repository, e.g., `https://github.com/sigmavirus24/github3.py`.

**id**

The unique GitHub assigned numerical id of this repository.

**issue\_comment\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `issue_comment_urlt.variables` for the list of variables that can be passed to `issue_comment_urlt.expand()`.

**issue\_events\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `issue_events_urlt.variables` for the list of variables that can be passed to `issue_events_urlt.expand()`.

**issues\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `issues_urlt.variables` for the list of variables that can be passed to `issues_urlt.expand()`.

**keys\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `keys_urlt.variables` for the list of variables that can be passed to `keys_urlt.expand()`.

**labels\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `labels_urlt.variables` for the list of variables that can be passed to `labels_urlt.expand()`.

**languages\_url**

The URL to retrieve this repository's list of languages.

**merges\_url**

The URL to retrieve this repository's list of merges.

**milestones\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `milestones_urlt.variables` for the list of variables that can be passed to `milestones_urlt.expand()`.

**name**

The name of the repository, e.g., `github3.py`.

**notifications\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `notifications_urlt.variables` for the list of variables that can be passed to `notifications_urlt.expand()`.

**owner**

The owner of the repository, e.g., `sigmavirus24`.

**private**

Whether the repository is private or public.

**pulls\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `pulls_urlt.variables` for the list of variables that can be passed to `pulls_urlt.expand()`.

**releases\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `releases_urlt.variables` for the list of variables that can be passed to `releases_urlt.expand()`.

**stargazers\_url**

The URL to retrieve this repository's list of stargazers.

**statuses\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `statuses_urlt.variables` for the list of variables that can be passed to `statuses_urlt.expand()`.

**subscribers\_url**

The URL to retrieve this repository's list of subscribers.

**subscription\_url**

The URL to modify subscription to this repository.

**tags\_url**

The URL to retrieve this repository's list of tags.

**teams\_url**

The URL to retrieve this repository's list of teams.

**trees\_urlt**

The `URITemplate` object representing the URI template returned by GitHub's API. Check `trees_urlt.variables` for the list of variables that can be passed to `trees_urlt.expand()`.

New in version 1.0.0.

**add\_collaborator** (*username*)

Add *username* as a collaborator to a repository.

**Parameters** *username* (str or *User*) – (required), username of the user

**Returns** True if successful, False otherwise

**Return type**

**archive** (*format*, *path*=", *ref*='master')

Get the tarball or zipball archive for this repo at ref.

See: <http://developer.github.com/v3/repos/contents/#get-archive-link>

**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** True if successful, False otherwise

**Return type** bool



**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**asset(id)**

Return a single asset.

**Parameters** **id** (*int*) – (required), id of the asset

**Returns** the asset

**Return type** *Asset*

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

Iterate over all 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** generator of users

**Return type** *ShortUser*

**blob(sha)**

Get the blob indicated by sha.

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

**Returns** the git blob

**Return type** *Blob*

**branch(name)**

Get the branch name of this repository.

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

**Returns** the branch

**Return type** *Branch*

**branches** (*number=-1, protected=False, etag=None*)

Iterate over the branches in this repository.

**Parameters**

- **number** (*int*) – (optional), number of branches to return. Default: -1 returns all branches
- **protected** (*bool*) – (optional), True lists only protected branches. Default: False
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of branches

**Return type** *Branch*

**code\_frequency** (*number=-1, etag=None*)

Iterate over the code frequency per week.

New in version 0.7.

Returns a weekly aggregate of the number of additions and deletions pushed to this repository.

---

**Note:** All statistics methods may return a 202. On those occasions, you will not receive any objects. You should store your iterator and check the new `last_status` attribute. If it is a 202 you should wait before re-requesting.

---

**Parameters**

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

**Returns** generator of lists [*seconds\_from\_epoch, additions, deletions*]

**Return type** list

**collaborators** (*affiliation='all', number=-1, etag=None*)

Iterate over the collaborators of this repository.

**Parameters**

- **affiliation** (*str*) – (optional), affiliation of the collaborator to the repository. Default: “all” returns contributors with all affiliations
- **number** (*int*) – (optional), number of collaborators to return. Default: -1 returns all comments
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of collaborators

**Return type** *Collaborator*

**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** generator of comments on commits

**Return type** *RepoComment*

**commit** (*sha*)

Get a single (repo) commit.

See `git_commit()` for the Git Data Commit.

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

**Returns** the commit

**Return type** *RepoCommit*

**commit\_activity** (*number=-1, etag=None*)

Iterate over last year of commit activity by week.

New in version 0.7.

See: <http://developer.github.com/v3/repos/statistics/>

---

**Note:** All statistics methods may return a 202. On those occasions, you will not receive any objects. You should store your iterator and check the new `last_status` attribute. If it is a 202 you should wait before re-requesting.

---

#### Parameters

- **number** (*int*) – (optional), number of weeks to return. Default -1 will return all of the weeks.
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of dictionaries

**Return type** dict

**commit\_comment** (*comment\_id*)

Get a single commit comment.

**Parameters** **comment\_id** (*int*) – (required), id of the comment used by GitHub

**Returns** the comment on the commit

**Return type** *RepoComment*

**commits** (*sha=None, path=None, author=None, number=-1, etag=None, since=None, until=None, per\_page=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

- **since** (*datetime* or *str*) – (optional), Only commits after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string.
- **until** (*datetime* or *str*) – (optional), Only commits before this date will be returned. This can be a *datetime* or an ISO8601 formatted date string.
- **per\_page** (*int*) – (optional), commits listing page size

**Returns** generator of commits

**Return type** *RepoCommit*

**compare\_commits** (*base*, *head*)

Compare two commits.

**Parameters**

- **base** (*str*) – (required), base for the comparison
- **head** (*str*) – (required), compare this against base

**Returns** the comparison of the commits

**Return type** *Comparison*

**contributor\_statistics** (*number=-1*, *etag=None*)

Iterate over the contributors list.

New in version 0.7.

See also: <http://developer.github.com/v3/repos/statistics/>

---

**Note:** All statistics methods may return a 202. On those occasions, you will not receive any objects. You should store your iterator and check the new `last_status` attribute. If it is a 202 you should wait before re-requesting.

---

**Parameters**

- **number** (*int*) – (optional), number of weeks to return. Default -1 will return all of the weeks.
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of contributor statistics for each contributor

**Return type** *ContributorStats*

**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** generator of contributor users

**Return type** *Contributor*

**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

**Returns** str (on Python 3, unicode on Python 2)

**create\_branch\_ref** (*name, sha=None*)

Create a branch git reference.

This is a shortcut for calling `github3.repos.repo.Repository.create_ref()`.

**Parameters**

- **branch** (*str*) – (required), the branch to create
- **sha** (*str*) – the commit to base the branch from

**Returns** a reference object representing the branch

**Return type** *Reference*

**create\_comment** (*body, sha, path=None, position=None, 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** the created comment

**Return type** *RepoComment*

**create\_commit** (*message, tree, parents, author=None, committer=None*)

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 omitted, GitHub will use the author parameters. Should be the same format as the author parameter.

**Returns** the created commit

**Return type** *Commit*

**create\_deployment** (*ref*, *required\_contexts=None*, *payload=""*, *auto\_merge=False*, *description=""*, *environment=None*)

Create a deployment.

**Parameters**

- **ref** (*str*) – (required), The ref to deploy. This can be a branch, tag, or sha.
- **required\_contexts** (*list*) – Optional array of status contexts verified against commit status checks. To bypass checking entirely pass an empty array. Default: []
- **payload** (*str*) – Optional JSON payload with extra information about the deployment. Default: ""
- **auto\_merge** (*bool*) – Optional parameter to merge the default branch into the requested deployment branch if necessary. Default: False
- **description** (*str*) – Optional short description. Default: ""
- **environment** (*str*) – Optional name for the target deployment environment (e.g., production, staging, qa). Default: "production"

**Returns** the created deployment

**Return type** *Deployment*

**create\_file** (*path*, *message*, *content*, *branch=None*, *committer=None*, *author=None*)

Create a file in this repository.

See also: <http://developer.github.com/v3/repos/contents/#create-a-file>

**Parameters**

- **path** (*str*) – (required), path of the file in the repository
- **message** (*str*) – (required), commit message
- **content** (*bytes*) – (required), the actual data in the file
- **branch** (*str*) – (optional), branch to create the commit on. Defaults to the default branch of the repository
- **committer** (*dict*) – (optional), if no information is given the authenticated user's information will be used. You must specify both a name and email.
- **author** (*dict*) – (optional), if omitted this will be filled in with committer information. If passed, you must specify both a name and email.

**Returns** dictionary of contents and commit for created file

**Return type** *Contents, Commit*

**create\_fork** (*organization=None*)

Create a fork of this repository.

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

**Returns** the fork of this repository

**Return type** *Repository*

**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** the created hook

**Return type** *Hook*

**create\_issue** (*title, body=None, assignee=None, milestone=None, labels=None, assignees=None*)  
Create 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** (*[str]*) – (optional), labels to apply to this issue
- **assignees** (*[str]*) – (optional), login of the users to assign the issue to

**Returns** the created issue

**Return type** *ShortIssue*

**create\_key** (*title, key, read\_only=False*)  
Create a deploy key.

#### Parameters

- **title** (*str*) – (required), title of key
- **key** (*str*) – (required), key text
- **read\_only** (*bool*) – (optional), restrict key access to read-only, default is False

**Returns** the created key

**Return type** *Key*

**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, e.g., '#fafafa' or 'fafafa' (the latter is what is sent)

**Returns** the created label

**Return type** *Label*

**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** the created milestone

**Return type** *Milestone*

**create\_project** (*name, body=None*)

Create a project for this repository.

**Parameters**

- **name** (*str*) – (required), name of the project
- **body** (*str*) – (optional), body of the project

**Returns** the created project

**Return type** *Project*

**create\_pull** (*title, base, head, body=None*)

Create a pull request of head onto base branch in this repo.

**Parameters**

- **title** (*str*) – (required)
- **base** (*str*) – (required), e.g., 'master'
- **head** (*str*) – (required), e.g., 'username:branch'
- **body** (*str*) – (optional), markdown formatted description

**Returns** the created pull request

**Return type** *ShortPullRequest*

**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., 'master'
- **head** (*str*) – (required), e.g., 'username:branch'

**Returns** the created pull request

**Return type** *ShortPullRequest*

**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** the created ref



**Return type** *Reference*

**create\_release** (*tag\_name*, *target\_commitish=None*, *name=None*, *body=None*, *draft=False*, *prerelease=False*)

Create a release for this repository.

**Parameters**

- **tag\_name** (*str*) – (required), name to give to the tag
- **target\_commitish** (*str*) – (optional), vague concept of a target, either a SHA or a branch name.
- **name** (*str*) – (optional), name of the release
- **body** (*str*) – (optional), description of the release
- **draft** (*bool*) – (optional), whether this release is a draft or not
- **prerelease** (*bool*) – (optional), whether this is a prerelease or not

**Returns** the created release

**Return type** *Release*

**create\_status** (*sha*, *state*, *target\_url=None*, *description=None*, *context='default'*)

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
- **context** (*str*) – (optional), A string label to differentiate this status from the status of other systems

**Returns** the created status

**Return type** *Status*

**create\_tag** (*tag*, *message*, *sha*, *obj\_type*, *tagger*, *lightweight=False*)

Create a tag in this repository.

By default, this method creates an annotated tag. If you wish to create a lightweight tag instead, pass `lightweight=True`.

If you are creating an annotated tag, this method makes **2 calls** to the API:

1. Creates the tag object
2. Creates the reference for the tag

This behaviour is required by the GitHub API.

**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 creating a lightweight tag, this will return a *Reference*, otherwise it will return a *Tag*

**Return type** *Tag* or *Reference*

**create\_tree** (*tree*, *base\_tree=None*)

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** the created tree

**Return type** *Tree*

**delete** ()

Delete this repository.

**Returns** True if successful, False otherwise

**Return type** bool

**delete\_key** (*key\_id*)

Delete the key with the specified id from your deploy keys list.

**Returns** True if successful, False otherwise

**Return type** bool

**delete\_subscription** ()

Delete the user’s subscription to this repository.

**Returns** True if successful, False otherwise

**Return type** bool

**deployment** (*id*)

Retrieve the deployment identified by id.

**Parameters** **id** (*int*) – (required), id for deployments.

**Returns** the deployment

**Return type** *Deployment*

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

Iterate over deployments for this repository.

**Parameters**

- **number** (*int*) – (optional), number of deployments to return. Default: -1, returns all available deployments
- **etag** (*str*) – (optional), ETag from a previous request for all deployments

**Returns** generator of deployments

**Return type** *Deployment*

**directory\_contents** (*directory\_path*, *ref=None*, *return\_as=<class 'list'>*)

Get the contents of each file in *directory\_path*.

If the path provided is actually a directory, you will receive a list back of the form:

```
[('filename.md', Contents(...)),
 ('github.py', Contents(...)),
 # ...
 ('fiz.py', Contents(...))]
```

You can either then transform it into a dictionary:

```
contents = dict(repo.directory_contents('path/to/dir/'))
```

Or you can use the *return\_as* parameter to have it return a dictionary for you:

```
contents = repo.directory_contents('path/to/dir/', return_as=dict)
```

### Parameters

- **path** (*str*) – (required), path to file, e.g. github3/repos/repo.py
- **ref** (*str*) – (optional), the string name of a commit/branch/tag. Default: master
- **return\_as** – (optional), how to return the directory's contents. Default: *list*

**Returns** list of tuples of the filename and the Contents returned

**Return type** [(*str*, *Contents*)]

**Raises** `github3.exceptions.UnprocessableResponseBody` – When the requested directory is not actually a directory

**edit** (*name*, *description=None*, *homepage=None*, *private=None*, *has\_issues=None*, *has\_wiki=None*, *has\_downloads=None*, *default\_branch=None*, *archived=None*, *allow\_merge\_commit=None*, *allow\_squash\_merge=None*, *allow\_rebase\_merge=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.
- **archived** (*bool*) – (optional), If not `None`, toggle the archived attribute on the repository to control whether it is archived or not.
- **allow\_rebase\_merge** (*bool*) – (optional), If not `None`, change whether the merge strategy that allows adding all commits from the head branch onto the base branch individually is enabled for this repository. API default: `None` - leave value unchanged.
- **allow\_squash\_merge** (*bool*) – (optional), If not `None`, change whether combining all commits from the head branch into a single commit in the base branch is allowed. API default: `None` - leave value unchanged.
- **allow\_merge\_commit** (*bool*) – (optional), If not `None`, change whether adding all commits from the head branch to the base branch with a merge commit is allowed. API default: `None` - leave value unchanged.

**Returns** `True` if successful, `False` otherwise

**Return type** `bool`

**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** generator of events

**Return type** `Event`

**file\_contents** (*path, ref=None*)

Get the contents of the file pointed to by `path`.

**Parameters**

- **path** (*str*) – (required), path to file, e.g. `github3/repos/repo.py`
- **ref** (*str*) – (optional), the string name of a commit/branch/tag. Default: `master`

**Returns** the contents of the file requested

**Return type** `Contents`

**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** generator of forks of this repository

**Return type** *ShortRepository*

**classmethod** `from_dict` (*json\_dict*, *session*)

Return an instance of this class formed from *json\_dict*.

**classmethod** `from_json` (*json*, *session*)

Return an instance of this class formed from *json*.

**git\_commit** (*sha*)

Get a single (git) commit.

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

**Returns** the single commit data from git

**Return type** *Commit*

**hook** (*hook\_id*)

Get a single hook.

**Parameters** **hook\_id** (*int*) – (required), id of the hook

**Returns** the hook

**Return type** *Hook*

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

Iterate over hooks registered on this repository.

**Parameters**

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

**Returns** generator of hooks

**Return type** *Hook*

**ignore** ()

Ignore notifications from this repository for the user.

New in version 1.0.

This replaces `Repository#set_subscription`.

**Returns** the new repository subscription

**Return type** :class:`~github3.notifications.RepositorySubscription`

**import\_issue** (*title*, *body*, *created\_at*, *assignee=None*, *milestone=None*, *closed=None*, *labels=None*, *comments=None*)

Import an issue into the repository.

See also: <https://gist.github.com/jonmagic/5282384165e0f86ef105>

**Parameters**

- **title** (*string*) – (required) Title of issue
- **body** (*string*) – (required) Body of issue
- **created\_at** (*datetime* or *str*) – (required) Creation timestamp
- **assignee** (*string*) – (optional) Username to assign issue to
- **milestone** (*int*) – (optional) Milestone ID

- **closed** (*boolean*) – (optional) Status of issue is Closed if True
- **labels** (*list*) – (optional) List of labels containing string names
- **comments** (*list*) – (optional) List of dictionaries which contain `created_at` and `body` attributes

**Returns** the imported issue

**Return type** *ImportedIssue*

**imported\_issue** (*imported\_issue\_id*)

Retrieve imported issue specified by imported issue id.

**Parameters** **imported\_issue\_id** (*int*) – (required) id of imported issue

**Returns** the imported issue

**Return type** *ImportedIssue*

**imported\_issues** (*number=-1, since=None, etag=None*)

Retrieve the collection of imported issues via the API.

See also: <https://gist.github.com/jonmagic/5282384165e0f86ef105>

**Parameters**

- **number** (*int*) – (optional), number of imported issues to return. Default: -1 returns all branches
- **since** – (optional), Only imported issues after this date will be returned. This can be a `datetime` instance, ISO8601 formatted date string, or a string formatted like so: 2016-02-04 i.e. %Y-%m-%d
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of imported issues

**Return type** *ImportedIssue*

**is\_assignee** (*username*)

Check if the user can be assigned an issue on this repository.

**Parameters** **username** (*str* or *User*) – name of the user to check

**Returns** bool

**is\_collaborator** (*username*)

Check to see if `username` is a collaborator on this repository.

**Parameters** **username** (*str* or *User*) – (required), login for the user

**Returns** True if successful, False otherwise

**Return type** bool

**issue** (*number*)

Get the issue specified by `number`.

**Parameters** **number** (*int*) – (required), number of the issue on this repository

**Returns** the issue

**Return type** *Issue*

**issue\_events** (*number=-1, etag=None*)

Iterate 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 events on issues

**Return type** *IssueEvent*

**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.

Changed in version 0.9.0: The `state` parameter now accepts ‘all’ in addition to ‘open’ and ‘closed’.

**Parameters**

- **milestone** (*int*) – (optional), ‘none’, or ‘\*’
- **state** (*str*) – (optional), accepted values: (‘all’, ‘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** – (optional), accepted values: (‘created’, ‘updated’, ‘comments’, ‘created’)
- **direction** (*str*) – (optional), accepted values: (‘asc’, ‘desc’)
- **since** (*datetime* or *str*) – (optional), Only issues after this date will be returned. This can be a *datetime* or an ISO8601 formatted date string, e.g., 2012-05-20T23:10:27Z
- **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** generator of issues

**Return type** *ShortIssue*

**key** (*id\_num*)

Get the specified deploy key.

**Parameters** **id\_num** (*int*) – (required), id of the key

**Returns** the deploy key

**Return type** *Key*

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

Iterate 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

**Return type** *Key*

**label** (*name*)

Get the label specified by *name*.

**Parameters** **name** (*str*) – (required), name of the label

**Returns** the label

**Return type** *Label*

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

Iterate 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

**Return type** *Label*

**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** generator of tuples

**Return type** tuple

**latest\_pages\_build** ()

Get the build information for the most recent Pages build.

**Returns** the information for the most recent build

**Return type** *PagesBuild*

**latest\_release** ()

Get the latest release.

Draft releases and prereleases are not returned by this endpoint.

**Returns** the release

**Return type** *Release*

**license** ()

Get the contents of a license for the repo.

**Returns** the license

**Return type** *RepositoryLicense*

**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** True if successful, False otherwise



**Return type** bool

**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** the commit resulting from the merge

**Return type** *RepoCommit*

**milestone** (*number*)

Get the milestone indicated by number.

**Parameters** **number** (*int*) – (required), unique id number of the milestone

**Returns** the milestone

**Return type** *Milestone*

**milestones** (*state=None, sort=None, direction=None, number=-1, etag=None*)

Iterate 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

**Return type** *Milestone*

**network\_events** (*number=-1, etag=None*)

Iterate 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

**Return type** *Event*

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**notifications** (*all=False, participating=False, since=None, number=-1, etag=None*)

Iterate 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** (*datetime* or *str*) – (optional), filters out any notifications updated before the given time. This can be a *datetime* or an *ISO8601* formatted date string, e.g., 2012-05-20T23:10:27Z
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

**Returns** generator of notification threads

**Return type** *Thread*

**pages** ()

Get information about this repository's pages site.

**Returns** information about this repository's GitHub pages site

**Return type** *PagesInfo*

**pages\_builds** (*number=-1, etag=None*)

Iterate over pages builds of this repository.

**Parameters**

- **number** (*int*) – (optional) the number of builds to return
- **etag** (*str*) – (optional), ETag value from a previous request

**Returns** generator of builds

**Return type** *PagesBuild*

**project** (*id, etag=None*)

Return the organization project with the given ID.

**Parameters** **id** (*int*) – (required), ID number of the project

**Returns** the project

**Return type** *Project*

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

Iterate over projects for 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 projects

**Return type** *Project*

**pull\_request** (*number*)

Get the pull request indicated by *number*.

**Parameters** **number** (*int*) – (required), number of the pull request.

**Returns** the pull request

**Return type** *PullRequest*

**pull\_requests** (*state=None, head=None, base=None, sort='created', direction='desc', number=-1, etag=None*)

List pull requests on repository.

Changed in version 0.9.0: The `state` parameter now accepts 'all' in addition to 'open' and 'closed'. The `sort` parameter was added. The `direction` parameter was added.

- **Parameters**

- **state** (*str*) – (optional), accepted values: ('all', 'open', 'closed')
- **head** (*str*) – (optional), filters pulls by head user and branch name in the format `user:ref-name`, e.g., `seveas:debian`
- **base** (*str*) – (optional), filter pulls by base branch name. Example: `develop`.
- **sort** (*str*) – (optional), Sort pull requests by `created`, `updated`, `popularity`, `long-running`. Default: 'created'
- **direction** (*str*) – (optional), Choose the direction to list pull requests. Accepted values: ('desc', 'asc'). Default: 'desc'
- **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 pull requests

**Return type** *ShortPullRequest*

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**readme** ()

Get the README for this repository.

**Returns** this repository's readme

**Return type** *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** the reference

**Return type** *Reference*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**refs** (*subspace=""*, *number=-1*, *etag=None*)  
Iterate 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

**Return type** *Reference*

**release** (*id*)  
Get a single release.

**Parameters** **id** (*int*) – (required), id of release

**Returns** the release

**Return type** *Release*

**release\_from\_tag** (*tag\_name*)  
Get a release by tag name.

release\_from\_tag() returns a release with specified tag while release() returns a release with specified release id

**Parameters** **tag\_name** (*str*) – (required) name of tag

**Returns** the release

**Return type** *Release*

**releases** (*number=-1*, *etag=None*)  
Iterate over releases for this repository.

**Parameters**

- **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 releases

**Return type** *Release*

**remove\_collaborator** (*username*)

Remove collaborator *username* from the repository.

**Parameters** *username* (str or *User*) – (required), login name of the collaborator

**Returns** True if successful, False otherwise

**Return type** bool

**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

**Return type** *ShortUser*

**statuses** (*sha, number=-1, etag=None*)

Iterate over the statuses for a specific SHA.

**Warning:** Deprecated in v1.0. Also deprecated upstream <https://developer.github.com/v3/repos/statuses/>

**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 statuses

**Return type** *Status*

**subscribe** ()

Subscribe the user to this repository’s notifications.

New in version 1.0.

This replaces `Repository#set_subscription`

**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** the new repository subscription

**Return type** *RepositorySubscription*

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

Iterate 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 users subscribed to this repository

**Return type** *ShortUser*

**subscription()**

Return subscription for this Repository.

**Returns** the user's subscription to this repository

**Return type** *RepositorySubscription*

**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** the annotated tag

**Return type** *Tag*

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

Iterate 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 tags with GitHub repository specific information

**Return type** *RepoTag*

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

Iterate 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

**Return type** *Team*

**tree** (*sha*)

Get a tree.

**Parameters** **sha** (*str*) – (required), sha of the object for this tree

**Returns** the tree

**Return type** *Tree*

**weekly\_commit\_count ()**

Retrieve the total commit counts.

---

**Note:** All statistics methods may return a 202. If github3.py receives a 202 in this case, it will return an empty dictionary. You should give the API a moment to compose the data and then re-request it via this method.

---

..versionadded:: 0.7

The dictionary returned has two entries: `all` and `owner`. Each has a fifty-two element long list of commit counts. (Note: `all` includes the owner.) `d['all'][0]` will be the oldest week, `d['all'][51]` will be the most recent.

**Returns** the commit count as a dictionary

**Return type** dict

**class** github3.repos.repo.**StarredRepository** (*json, session*)

This object represents the data returned about a user's starred repos.

GitHub used to send back the `starred_at` attribute on Repositories but then changed the structure to a new object that separates that from the Repository representation. This consolidates the two.

Attributes:

**starred\_at**

A parsed `datetime` object representing the date the repository was starred.

**repository**

The `Repository` that was starred by the user.

See also: <https://developer.github.com/v3/activity/starring/#list-repositories-being-starred>

**as\_dict ()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json ()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from `json_dict`.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from `json`.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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`

**class** `github3.repos.contents.Contents` (*json, session*)

A representation of file contents returned via the API.

See also: <http://developer.github.com/v3/repos/contents/>

This object has the following attributes:

**content**

The body of the file. If this is present, it may be base64 encoded.

**encoding**

The encoding used on the *content* when returning the data from the API, e.g., `base64`. If *content* is not present this will not be present either.

**decoded**

---

**Note:** This is a computed attribute which isn't returned by the API.

---

Changed in version 0.5.2.

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')`.

**git\_url**

The URL for the Git API pertaining to this file.



**html\_url**

The URL to open this file in a browser.

**links**

A dictionary of links returned about the contents and related resources.

**name**

The name of the file.

**path**

The path to this file.

**sha**

The SHA1 of the contents of this file.

**size**

The size of file in bytes.

**submodule\_git\_url**

The URL of the git submodule (if this is a git submodule).

**target**

If the file is a symlink, this will be present and provides the type of file that the symlink points to.

**type**

Type of content, e.g., 'file', 'symlink', or 'submodule'.

**delete** (*message*, *branch=None*, *committer=None*, *author=None*)

Delete this file.

**Parameters**

- **message** (*str*) – (required), commit message to describe the removal
- **branch** (*str*) – (optional), branch where the file exists. Defaults to the default branch of the repository.
- **committer** (*dict*) – (optional), if no information is given the authenticated user's information will be used. You must specify both a name and email.
- **author** (*dict*) – (optional), if omitted this will be filled in with committer information. If passed, you must specify both a name and email.

**Returns** dictionary of new content and associated commit

**Return type** *Contents* and *Commit*

**update** (*message*, *content*, *branch=None*, *committer=None*, *author=None*)

Update this file.

**Parameters**

- **message** (*str*) – (required), commit message to describe the update
- **content** (*str*) – (required), content to update the file with
- **branch** (*str*) – (optional), branch where the file exists. Defaults to the default branch of the repository.
- **committer** (*dict*) – (optional), if no information is given the authenticated user's information will be used. You must specify both a name and email.
- **author** (*dict*) – (optional), if omitted this will be filled in with committer information. If passed, you must specify both a name and email.

**Returns** dictionary containing the updated contents object and the commit in which it was changed.

**Return type** dictionary of *Contents* and *Commit*

**class** `github3.repos.hook.Hook` (*json, session*)

The representation of a hook on a repository.

See also: <http://developer.github.com/v3/repos/hooks/>

This object has the following attributes:

**active**

A boolean attribute describing whether the hook is active or not.

**config**

A dictionary containing the configuration for this hook.

**created\_at**

A `datetime` object representing the date and time when this hook was created.

**events**

The list of events which trigger this hook.

**id**

The unique identifier for this hook.

**name**

The name provided to this hook.

**updated\_at**

A `datetime` object representing the date and time when this hook was updated.

**delete** ()

Delete this hook.

**Returns** True if successful, False otherwise

**Return type** bool

**edit** (*config={}*, *events=[]*, *add\_events=[]*, *rm\_events=[]*, *active=True*)

Edit this hook.

**Parameters**

- **config** (*dict*) – (optional), 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
- **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** True if successful, False otherwise

**Return type** bool

**ping** ()

Ping this hook.

**Returns** True if successful, False otherwise

**Return type** bool

**test** ()

Test this hook.

**Returns** True if successful, False otherwise

**Return type** bool

**class** github3.repos.issue\_import.**ImportedIssue** (*json, session*)

Represents an issue imported via the unofficial API.

See also: <https://gist.github.com/jonmagic/5282384165e0f86ef105>

This object has the following attributes:

**created\_at**

A *datetime* object representing the date and time when this imported issue was created.

**id**

The globally unique identifier for this imported issue.

**import\_issues\_url**

The URL used to import more issues via the API.

**repository\_url**

The URL used to retrieve the repository via the API.

**status**

The status of this imported issue.

**updated\_at**

A *datetime* object representing the date and time when this imported issue was last updated.

## Git-related Objects

**class** github3.repos.tag.**RepoTag** (*json, session*)

Representation of a tag made on a GitHub repository.

---

**Note:** This is distinct from *Tag*. This object has information specific to a tag on a *GitHub* repository. That includes URLs to the tarball and zipball files auto-generated by GitHub.

---

See also: <http://developer.github.com/v3/repos/#list-tags>

This object has the following attributes:

**commit**

Changed in version 1.0.0: This attribute used to be a two item dictionary.

A *MiniCommit* object representing the commit this tag references.

**name**

The name of this tag, e.g., v1.0.0.

**tarball\_url**

The URL for the tarball file generated by GitHub for this tag.

**zipball\_url**

The URL for the zipball file generated by GitHub for this tag.

## Branches

**class** `github3.repos.branch.Branch` (*json, session*)

The representation of a branch returned in a collection.

GitHub's API returns different amounts of information about repositories based upon how that information is retrieved. This object exists to represent the limited amount of information returned for a specific branch in a collection. For example, you would receive this class when calling `branches()`. To provide a clear distinction between the types of branches, github3.py uses different classes with different sets of attributes.

This object has the same attributes as a `ShortBranch` as well as the following:

### **links**

The dictionary of URLs returned by the API as `_links`.

### **protected**

A boolean attribute that describes whether this branch is protected or not.

### **original\_protection**

Changed in version 1.1.0: To support a richer branch protection API, this is the new name for the information formerly stored under the attribute `protection`.

A dictionary with details about the protection configuration of this branch.

### **protection\_url**

The URL to access and manage details about this branch's protection.

**class** `github3.repos.branch.ShortBranch` (*json, session*)

The representation of a branch returned in a collection.

GitHub's API returns different amounts of information about repositories based upon how that information is retrieved. This object exists to represent the limited amount of information returned for a specific branch in a collection. For example, you would receive this class when calling `branches()`. To provide a clear distinction between the types of branches, github3.py uses different classes with different sets of attributes.

This object has the following attributes:

### **commit**

A `MiniCommit` representation of the newest commit on this branch with the associated repository meta-data.

### **name**

The name of this branch.

**class** `github3.repos.branch.BranchProtection` (*json, session*)

The representation of a branch's protection.

### **See also:**

**Branch protection API documentation** [GitHub's documentation of branch protection](#)

This object has the following attributes:

### **enforce\_admins**

A `ProtectionEnforceAdmins` instance representing whether required status checks are required for admins.

### **restrictions**

A `ProtectionRestrictions` representing who can push to this branch. Team and user restrictions are only available for organization-owned repositories.

**required\_pull\_request\_reviews**

A *ProtectionRequiredPullRequestReviews* representing the protection provided by requiring pull request reviews.

**required\_status\_checks**

A *ProtectionRequiredStatusChecks* representing the protection provided by requiring status checks.

**class** github3.repos.branch.**ProtectionEnforceAdmins** (*json, session*)

The representation of a sub-portion of branch protection.

**See also:**

**Branch protection API documentation** GitHub's documentation of branch protection

This object has the following attributes:

**enabled**

A boolean attribute indicating whether the `enforce_admins` protection is enabled or disabled.

**class** github3.repos.branch.**ProtectionRestrictions** (*json, session*)

The representation of a sub-portion of branch protection.

**See also:**

**Branch protection API documentation** GitHub's documentation of branch protection

**Branch restriction documentation** GitHub's description of branch restriction

This object has the following attributes:

**original\_teams**

List of *ShortTeam* objects representing the teams allowed to push to the protected branch.

**original\_users**

List of *ShortUser* objects representing the users allowed to push to the protected branch.

**teams\_url**

The URL to retrieve the list of teams allowed to push to the protected branch.

**users\_url**

The URL to retrieve the list of users allowed to push to the protected branch.

**teams** (*number=-1*)

Retrieve an up-to-date listing of teams.

**Returns** An iterator of teams

**Return type** *ShortTeam*

**users** (*number=-1*)

Retrieve an up-to-date listing of users.

**Returns** An iterator of users

**Return type** *ShortUser*

**class** github3.repos.branch.**ProtectionRequiredPullRequestReviews** (*json, session*)

The representation of a sub-portion of branch protection.

**See also:**

**Branch protection API documentation** GitHub's documentation of branch protection

**class** `github3.repos.branch.ProtectionRequiredStatusChecks` (*json, session*)

The representation of a sub-portion of branch protection.

**See also:**

**Branch protection API documentation** GitHub's documentation of branch protection

**Required Status Checks documentation** GitHub's description of required status checks

**add\_contexts** (*contexts*)

Add contexts to the existing list of required contexts.

See: <https://developer.github.com/v3/repos/branches/#add-required-status-checks-contexts-of-protected-branch>

**Parameters** `contexts` (*list*) – The list of contexts to append to the existing list.

**Returns** The updated list of contexts.

**Return type** `list`

**contexts** ()

Retrieve the list of contexts required as status checks.

See: <https://developer.github.com/v3/repos/branches/#list-required-status-checks-contexts-of-protected-branch>

**Returns** A list of context names which are required status checks.

**Return type** `list`

**delete** ()

Remove required status checks from this branch.

See: <https://developer.github.com/v3/repos/branches/#remove-required-status-checks-of-protected-branch>

**Returns** True if successful, False otherwise

**Return type** `bool`

**remove\_contexts** (*contexts*)

Remove the specified contexts from the list of required contexts.

See: <https://developer.github.com/v3/repos/branches/#remove-required-status-checks-contexts-of-protected-branch>

**Parameters** `contexts` (*list*) – The context names to remove

**Returns** The updated list of contexts required as status checks.

**Return type** `list`

**replace\_contexts** (*contexts*)

Replace the existing contexts required as status checks.

See: <https://developer.github.com/v3/repos/branches/#replace-required-status-checks-contexts-of-protected-branch>

**Parameters** `contexts` (*list*) – The names of the contexts to be required as status checks

**Returns** The updated list of contexts required as status checks.

**Return type** `list`

**update** (*strict=None, contexts=None*)

Update required status checks for the branch.

This requires admin or owner permissions to the repository and branch protection to be enabled.

**See also:**

**API docs** Description of how to update the required status checks.

#### Parameters

- **strict** (*bool*) – Whether this should be strict protection or not.
- **contexts** (*list*) – A list of context names that should be required.

**Returns** A new instance of this class with the updated information

**Return type** *ProtectionRequiredStatusChecks*

## Commits

**class** `github3.repos.commit.MiniCommit` (*json, session*)  
A commit returned on a ShortBranch.

**class** `github3.repos.commit.ShortCommit` (*json, session*)  
Representation of an incomplete commit in a collection.

**class** `github3.repos.commit.RepoCommit` (*json, session*)  
Representation of a commit with repository and git data.

**class** `github3.repos.comparison.Comparison` (*json, session*)  
A representation of a comparison between two or more commit objects.

See also: <http://developer.github.com/v3/repos/commits/#compare-two-commits>

This object has the following attributes:

```
.. attribute:: ahead_by
```

The number of commits between the head and base commit.

#### **base\_commit**

A *ShortCommit* representing the base commit in this comparison.

#### **behind\_by**

The number of commits the head commit is behind the base.

#### **commits**

A list of *ShortCommit* objects representing the commits in the comparison.

#### **diff\_url**

The URL to retrieve the diff between the head and base commits.

#### **files**

A list of dictionaries describing each of the modified files in the comparison.

#### **html\_url**

The URL to view the comparison in a browser.

#### **patch\_url**

The URL to retrieve the patch-formatted diff of this comparison.

#### **permalink\_url**

The permanent URL to retrieve this comparison.

#### **status**

Whether the head commit is ahead or behind of base.

**total\_commits**

The total number of commits difference.

**diff()**

Retrieve the diff for this comparison.

**Returns** the diff as a bytes object

**Return type** bytes

**patch()**

Retrieve the patch formatted diff for this commit.

**Returns** the patch as a bytes object

**Return type** bytes

## Release Objects

**class** github3.repos.release.**Asset** (*json, session*)

Representation of an asset in a release.

**See also:**

**List Assets, [assets\(\)](#)** Documentation around listing assets of a release

**Upload Assets, [upload\\_asset\(\)](#)** Documentation around uploading assets to a release

**Get a Single Asset, [asset\(\)](#)** Documentation around retrieving an asset

**Edit an Asset, [edit\(\)](#)** Documentation around editing an asset

**Delete an Asset, [delete\(\)](#)** Documentation around deleting an asset

This object has the following attributes:

**browser\_download\_url**

The user-friendly URL to download this asset via a browser.

**content\_type**

The Content-Type provided by the uploader when this asset was created.

**created\_at**

A `datetime` object representing the date and time when this asset was created.

**download\_count**

The number of times this asset has been downloaded.

**download\_url**

The URL to retrieve this uploaded asset via the API, e.g., tarball, zipball, etc.

**id**

The unique identifier of this asset.

**label**

The short description of this asset.

**name**

The name provided to this asset.

**size**

The file size of this asset.



**state**

The state of this asset, e.g., 'uploaded'.

**updated\_at**

A `datetime` object representing the date and time when this asset was most recently updated.

**delete()**

Delete this asset if the user has push access.

**Returns** True if successful; False if not successful

**Return type** bool

**download(path=)**

Download the data for this asset.

**Parameters** `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

**Returns** name of the file, if successful otherwise `None`

**Return type** `str`

**edit(name, label=None)**

Edit this asset.

**Parameters**

- **name** (`str`) – (required), The file name of the asset
- **label** (`str`) – (optional), An alternate description of the asset

**Returns** True if successful, False otherwise

**Return type** bool

**class** `github3.repos.release.Release` (`json`, `session`)

Representation of a GitHub release.

It holds the information GitHub returns about a release from a `Repository`.

Please see GitHub's [Releases Documentation](#) for more information.

This object has the following attributes:

**original\_assets**

A list of `Asset` objects representing the assets uploaded for this release.

**assets\_url**

The URL to retrieve the assets from the API.

**author**

A `ShortUser` representing the creator of this release.

**body**

The description of this release as written by the release creator.

**created\_at**

A `datetime` object representing the date and time when this release was created.

**draft**

A boolean attribute describing whether this release is a draft.

**html\_url**

The URL to view this release in a browser.

**id**

The unique identifier of this release.

**name**

The name given to this release by the *author*.

**prerelease**

A boolean attribute indicating whether the release is a pre-release.

**published\_at**

A *datetime* object representing the date and time when this release was published.

**tag\_name**

The name of the tag associated with this release.

**tarball\_url**

The URL to retrieve a GitHub generated tarball for this release from the API.

**target\_commitish**

The reference (usually a commit) that is targeted by this release.

**upload\_url**

A *URITemplate* object that expands to form the URL to upload assets to.

**zipball\_url**

The URL to retrieve a GitHub generated zipball for this release from the API.

**archive** (*format*, *path=""*)

Get the tarball or zipball archive for this release.

**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

**Returns** True if successful, False otherwise

**Return type** bool

**asset** (*asset\_id*)

Retrieve the asset from this release with *asset\_id*.

**Parameters** **asset\_id** (*int*) – ID of the Asset to retrieve

**Returns** the specified asset, if it exists

**Return type** *Asset*

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

Iterate over the assets available for this release.

**Parameters**

- **number** (*int*) – (optional), Number of assets to return
- **etag** (*str*) – (optional), last ETag header sent

**Returns** generator of asset objects

**Return type** *Asset*

**delete()**

Delete this release.

Only users with push access to the repository can delete a release.

**Returns** True if successful; False if not successful

**Return type** bool

**edit** (*tag\_name=None, target\_commitish=None, name=None, body=None, draft=None, prerelease=None*)

Edit this release.

Only users with push access to the repository can edit a release.

If the edit is successful, this object will update itself.

#### Parameters

- **tag\_name** (*str*) – (optional), Name of the tag to use
- **target\_commitish** (*str*) – (optional), The “commitish” value that determines where the Git tag is created from. Defaults to the repository’s default branch.
- **name** (*str*) – (optional), Name of the release
- **body** (*str*) – (optional), Description of the release
- **draft** (*boolean*) – (optional), True => Release is a draft
- **prerelease** (*boolean*) – (optional), True => Release is a prerelease

**Returns** True if successful; False if not successful

**Return type** bool

**upload\_asset** (*content\_type, name, asset, label=None*)

Upload an asset to this release.

---

**Note:** All parameters are required.

---

#### Parameters

- **content\_type** (*str*) – The content type of the asset. Wikipedia has a list of common media types
- **name** (*str*) – The name of the file
- **asset** – The file or bytes object to upload.
- **label** – (optional), An alternate short description of the asset.

**Returns** the created asset

**Return type** *Asset*

## Pages Objects

**class** github3.repos.pages.**PagesInfo** (*json, session*)

Representation of the information about a GitHub pages website.

**cname**

The cname in use for the pages site, if one is set.

**custom\_404**

A boolean attribute indicating whether the user configured a custom 404 page for this site.

**status**

The current status of the pages site, e.g., `built`.

**class** `github3.repos.pages.PagesBuild` (*json, session*)

Representation of a single build of a GitHub pages website.

**commit**

The SHA of the commit that triggered this build.

**created\_at**

A `datetime` object representing the date and time when this build was created.

**duration**

The time it spent processing this build.

**error**

If this build errored, a dictionary containing the error message and details about the error.

**pusher**

A `ShortUser` representing the user who pushed the commit that triggered this build.

**status**

The current status of the build, e.g., `building`.

**updated\_at**

A `datetime` object representing the date and time when this build was last updated.

## Comment Objects

More information about these classes can be found in the official documentation about [comments](#).

**class** `github3.repos.comment.ShortComment` (*json, session*)

The representation of an abridged comment on an object in a repo.

This object has the following attributes:

**author\_association**

The affiliation the author of this comment has with the repository.

**body**

The Markdown formatted text of this comment.

**commit\_id**

The SHA1 associated with this comment.

**created\_at**

A `datetime` object representing the date and time when this comment was created.

**html\_url**

The URL to view this comment in a browser.

**id**

The unique identifier of this comment.

**line**

The line number where the comment is located.

**path**

The path to the file where this comment was made.

**position**

The position in the diff where the comment was left.

**updated\_at**

A `datetime` object representing the date and time when this comment was most recently updated.

**user**

A `ShortUser` representing the author of this comment.

**class** `github3.repos.comment.RepoComment` (*json, session*)

The representation of the full comment on an object in a repository.

This object has the same attributes as a `ShortComment` as well as the following:

**body\_html**

The HTML formatted text of this comment.

**body\_text**

The plain-text formatted text of this comment.

## Deployment and Status Objects

**class** `github3.repos.deployment.Deployment` (*json, session*)

Representation of a deployment of a repository at a point in time.

See also: <https://developer.github.com/v3/repos/deployments/>

This object has the following attributes:

**created\_at**

A `datetime` representing the date and time when this deployment was created.

**creator**

A `ShortUser` representing the user who created this deployment.

**description**

The description of this deployment as provided by the `creator`.

**environment**

The environment targeted for this deployment, e.g., 'production', 'staging'.

**id**

The unique identifier of this deployment.

**payload**

The JSON payload string sent as part to trigger this deployment.

**ref**

The reference used to create this deployment, e.g., 'deploy-20140526'.

**sha**

The SHA1 of the branch on GitHub when it was deployed.

**statuses\_url**

The URL to retrieve the statuses of this deployment from the API.

**updated\_at**

A `datetime` object representing the date and time when this deployment was most recently updated.

**create\_status** (*state, target\_url=None, description=None*)

Create a new deployment status for this deployment.

### Parameters

- **state** (*str*) – (required), The state of the status. Can be one of `pending`, `success`, `error`, or `failure`.
- **target\_url** (*str*) – The target URL to associate with this status. This URL should contain output to keep the user updated while the task is running or serve as historical information for what happened in the deployment. Default: `''`.
- **description** (*str*) – A short description of the status. Default: `''`.

**Returns** the incomplete deployment status

**Return type** *DeploymentStatus*

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

Iterate over the deployment statuses for this deployment.

**Parameters**

- **number** (*int*) – (optional), the number of statuses to return. Default: `-1`, returns all statuses.
- **etag** (*str*) – (optional), the ETag header value from the last time you iterated over the statuses.

**Returns** generator of the statuses of this deployment

**Return type** *DeploymentStatus*

**class** `github3.repos.deployment.DeploymentStatus` (*json, session*)

Representation of the status of a deployment of a repository.

See also <https://developer.github.com/v3/repos/deployments/#get-a-single-deployment-status>

This object has the following attributes:

**created\_at**

A *datetime* representing the date and time when this deployment status was created.

**creator**

A *ShortUser* representing the user who created this deployment status.

**deployment\_url**

The URL to retrieve the information about the deployment from the API.

**description**

The description of this status as provided by the *creator*.

**id**

The unique identifier of this deployment.

**state**

The state of the deployment, e.g., `'success'`.

**target\_url**

The URL to associate with this status. This should link to the output of the deployment.

**class** `github3.repos.status.ShortStatus` (*json, session*)

Representation of a short status on a repository.

New in version 1.0.0.

This is the representation found in a *CombinedStatus* object.

See also: <http://developer.github.com/v3/repos/statuses/>

This object has the following attributes:

**context**

This is a string that explains the context of this status object. For example, 'continuous-integration/travis-ci/pr'.

**created\_at**

A `datetime` object representing the date and time when this status was created.

**creator**

A `ShortUser` representing the user who created this status.

**description**

A short description of the status.

**id**

The unique identifier of this status object.

**state**

The state of this status, e.g., 'success', 'pending', 'failure'.

**target\_url**

The URL to retrieve more information about this status.

**updated\_at**

A `datetime` object representing the date and time when this status was most recently updated.

**class** `github3.repos.status.CombinedStatus` (*json, session*)

A representation of the combined statuses in a repository.

See also: <http://developer.github.com/v3/repos/statuses/>

This object has the following attributes:

**commit\_url**

The URL of the commit this combined status is present on.

**repository**

A `ShortRepository` representing the repository on which this combined status exists.

**sha**

The SHA1 of the commit this status exists on.

**state**

The state of the combined status, e.g., 'success', 'pending', 'failure'.

**statuses**

The list of `ShortStatus` objects representing the individual statuses that is combined in this object.

**total\_count**

The total number of sub-statuses.

**class** `github3.repos.status.Status` (*json, session*)

Representation of a full status on a repository.

See also: <http://developer.github.com/v3/repos/statuses/>

This object has the same attributes as a `ShortStatus` as well as the following attributes:

**creator**

A `ShortUser` representing the user who created this status.

## Contributor Statistics Objects

**class** `github3.repos.stats.ContributorStats` (*json, session*)

Representation of a user's contributor statistics to a repository.

See also <http://developer.github.com/v3/repos/statistics/>

This object has the following attributes:

**author**

A *ShortUser* representing the contributor whose stats this object represents.

**total**

The total number of commits authored by *author*.

**weeks**

A list of dictionaries containing weekly statistical data.

**alternate\_weeks**

---

**Note:** *github3* generates this data for a more humane interface to the data in *weeks*.

---

A list of dictionaries that provide an easier to remember set of keys as well as a `datetime` object representing the start of the week. The dictionary looks vaguely like:

```
{
    'start of week': datetime(2013, 5, 5, 5, 0, tzinfo=tzutc())
    'additions': 100,
    'deletions': 150,
    'commits': 5,
}
```

## 12.1.12 Search Results

These classes are meant to expose the entirety of an item returned as a search result by GitHub's Search API.

**class** `github3.search.CodeSearchResult` (*json, session*)

A representation of a code search result from the API.

This object has the following attributes:

**git\_url**

The URL to retrieve the blob via Git

**html\_url**

The URL to view the blob found in a browser.

**name**

The name of the file where the search result was found.

**path**

The path in the repository to the file containing the result.

**repository**

A *ShortRepository* representing the repository in which the result was found.

**score**

The confidence score assigned to the result.



**sha**

The SHA1 of the blob in which the code can be found.

**text\_matches**

A list of the text matches in the blob that generated this result.

---

**Note:** To receive these, you must pass `text_match=True` to `search_code()`.

---

**class** `github3.search.IssueSearchResult` (*json, session*)

A representation of a search result containing an issue.

This object has the following attributes:

**issue**

A *ShortIssue* representing the issue found in this search result.

**score**

The confidence score of this search result.

**text\_matches**

A list of matches in the issue for this search result.

---

**Note:** To receive these, you must pass `text_match=True` to `search_issues()`.

---

**class** `github3.search.RepositorySearchResult` (*json, session*)

A representation of a search result containing a repository.

This object has the following attributes:

```
.. attribute:: repository
```

A *ShortRepository* representing the repository found by the search.

**score**

The confidence score of this search result.

**text\_matches**

A list of the text matches in the repository that generated this result.

---

**Note:** To receive these, you must pass `text_match=True` to `search_code()`.

---

**class** `github3.search.UserSearchResult` (*json, session*)

Representation of a search result for a user.

This object has the following attributes:

**score**

The confidence score of this result.

**text\_matches**

If present, a list of text strings that match the search string.

**user**

A *ShortUser* representing the user found in this search result.

### 12.1.13 Custom Iterator Structures

Many of the methods in `github3.py` that return iterators of another object are actually returning one of the iterators below. These iterators effectively allow users to ignore GitHub's API pagination of large sets of data. In all senses, they behave like a normal Python iterator. Their difference is that they have extra logic around making API requests and coercing the JSON into predefined objects.

**class** `github3.structs.GitHubIterator` (*count*, *url*, *cls*, *session*, *params=None*, *etag=None*,  
*headers=None*)

The `GitHubIterator` class powers all of the `iter_*` methods.

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**cls = None**

Class for constructing an item to return

**count = None**

Number of items left in the iterator

**etag = None**

The ETag Header value returned by GitHub

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from `json_dict`.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from `json`.

**headers = None**

Headers generated for the GET request

**last\_response = None**

The last response seen

**last\_status = None**

Last status code received

**last\_url = None**

Last URL that was requested

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**original = None**

Original number of items requested

**params = None**

Parameters of the query string

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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`

**url = None**

URL the class used to make it's first GET

**class** `github3.structs.SearchIterator` (*count, url, cls, session, params=None, etag=None, headers=None*)

This is a special-cased class for returning iterable search results.

It inherits from `GitHubIterator`. All members and methods documented here are unique to instances of this class. For other members and methods, check its parent class.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** `dict`

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from *json*.

**items = None**

Items array returned in the last request

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**total\_count = None**

Total count returned by GitHub

## 12.1.14 Users and their Associated Objects

This section of the documentation covers the representations of various objects related to the [Users API](#).

## User Objects

**class** `github3.users.ShortUser` (*json, session*)

Object for the shortened representation of a User.

GitHub’s API returns different amounts of information about users based upon how that information is retrieved. Often times, when iterating over several users, GitHub will return less information. To provide a clear distinction between the types of users, github3.py uses different classes with different sets of attributes.

New in version 1.0.0.

**avatar\_url**

The URL of the avatar (possibly from Gravatar)

**events\_urlt**

A URITemplate object from `uritemplate` that can be used to generate an events URL

**followers\_url**

A string representing the resource to retrieve a User’s followers

**following\_urlt**

A URITemplate object from `uritemplate` that can be used to generate the URL to check if this user is following `other_user`

**gists\_urlt**

A URITemplate object from `uritemplate` that can be used to generate the URL to retrieve a Gist by its id

**gravatar\_id**

The identifier for the user’s gravatar

**html\_url**

The URL of the user’s publicly visible profile. For example, `https://github.com/sigmavirus24`

**id**

The unique ID of the account

**login**

The username of the user, e.g., `sigmavirus24`

**organizations\_url**

A string representing the resource to retrieve the organizations to which a user belongs

**received\_events\_url**

A string representing the resource to retrieve the events a user received

**repos\_url**

A string representing the resource to list a user’s repositories

**site\_admin**

A boolean attribute indicating whether the user is a member of GitHub’s staff

**starred\_urlt**

A URITemplate object from `uritemplate` that can be used to generate a URL to retrieve whether the user has starred a repository.

**subscriptions\_url**

A string representing the resource to list a user’s subscriptions

**type**

A string representing the type of User account this. In all cases should be “User”

**url**

A string of this exact resource retrievable from GitHub's API

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete the user.

Per GitHub API documentation, it is often preferable to suspend the user.

---

**Note:** This is only available for admins of a GitHub Enterprise instance.

---

**Returns** bool – True if successful, False otherwise

**demote()**

Demote a site administrator to simple user.

You can demote any user account except your own.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**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** generator of *Events*

**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*

**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*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**impersonate** (*scopes=None*)

Obtain an impersonation token for the user.

The retrieved token will allow impersonation of the user. This is only available for admins of a GitHub Enterprise instance.

**Parameters** **scopes** (*list*) – (optional), areas you want this token to apply to, i.e., ‘gist’, ‘user’

**Returns** *Authorization*

**is\_assignee\_on** (*username, repository*)

Check if this user can be assigned to issues on *username/repository*.

**Parameters**

- **username** (*str*) – owner’s username of the repository
- **repository** (*str*) – name of the repository

**Returns** True if the use can be assigned, False otherwise

**Return type** *bool*

**is\_following** (*username*)

Check if this user is following *username*.

**Parameters** **username** (*str*) – (required)

**Returns** *bool*

**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*

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**organization\_events** (*org*, *number=-1*, *etag=None*)

Iterate over events from the user's organization dashboard.

---

**Note:** 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** generator of *Events*

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

Iterate over organizations the user is member of.

#### Parameters

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

**Returns** generator of *ShortOrganizations*

**promote** ()

Promote a user to site administrator.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**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*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**rename** (*login*)

Rename the user.

---

**Note:** This is only available for administrators of a GitHub Enterprise instance.

---

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

**Returns** bool

**revoke\_impersonation** ()

Revoke all impersonation tokens for the current user.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**starred\_repositories** (*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 *StarredRepository*

**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`

**suspend** ()

Suspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**unsuspend** ()

Unsuspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**class** `github3.users.User` (*json, session*)

Object for the full representation of a User.

GitHub’s API returns different amounts of information about users based upon how that information is retrieved. This object exists to represent the full amount of information returned for a specific user. For example, you would receive this class when calling `user()`. To provide a clear distinction between the types of users, `github3.py` uses different classes with different sets of attributes.

This object no longer contains information about the currently authenticated user (e.g., `me()`).

Changed in version 1.0.0.

This object contains all of the attributes available on `ShortUser` as well as the following:

**bio**

The markdown formatted User’s biography

**blog**

The URL of the user’s blog

**company**

The name or GitHub handle of the user’s company

**created\_at**

A parsed `datetime` object representing the date the user was created

**email**

The email address the user has on their public profile page

**followers\_count**

The number of followers of this user

**following\_count**

The number of users this user follows

**hireable**

Whether or not the user has opted into GitHub jobs advertising

**location**

The location specified by the user on their public profile

**name**

The name specified by their user on their public profile

**public\_gists\_count**

The number of public gists owned by this user

**updated\_at**

A parsed `datetime` object representing the date the user was last updated

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete the user.

Per GitHub API documentation, it is often preferable to suspend the user.

---

**Note:** This is only available for admins of a GitHub Enterprise instance.

---

**Returns** bool – True if successful, False otherwise

**demote()**

Demote a site administrator to simple user.

You can demote any user account except your own.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**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** generator of *Events*

**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*

**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*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**impersonate** (*scopes=None*)

Obtain an impersonation token for the user.

The retrieved token will allow impersonation of the user. This is only available for admins of a GitHub Enterprise instance.

**Parameters** **scopes** (*list*) – (optional), areas you want this token to apply to, i.e., ‘gist’, ‘user’

**Returns** *Authorization*

**is\_assignee\_on** (*username, repository*)

Check if this user can be assigned to issues on *username/repository*.

**Parameters**

- **username** (*str*) – owner’s username of the repository
- **repository** (*str*) – name of the repository

**Returns** True if the use can be assigned, False otherwise

**Return type** *bool*

**is\_following** (*username*)

Check if this user is following *username*.

**Parameters** **username** (*str*) – (required)

**Returns** bool

**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*

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**organization\_events** (*org, number=-1, etag=None*)

Iterate over events from the user's organization dashboard.

---

**Note:** 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** generator of *Events*

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

Iterate over organizations the user is member of.

**Parameters**

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

**Returns** generator of *ShortOrganizations*

**promote** ()

Promote a user to site administrator.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**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*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**rename** (*login*)

Rename the user.

---

**Note:** This is only available for administrators of a GitHub Enterprise instance.

---

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

**Returns** bool

**revoke\_impersonation** ()

Revoke all impersonation tokens for the current user.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**starred\_repositories** (*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 *StarredRepository*

**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*

**suspend** ()

Suspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**unsuspend** ()

Unsuspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**class** github3.users.**AuthenticatedUser** (*json, session*)

Object to represent the currently authenticated user.

This is returned by *me* (). It contains the extra information that is not returned for other users such as the currently authenticated user’s plan and private email information.

New in version 1.0.0.

Changed in version 1.0.0: The *total\_private\_gists* attribute is no longer returned by GitHub’s API and so is removed.

This object has all of the same attribute as the *ShortUser* and *User* objects as well as:

**disk\_usage**

The amount of repository space that has been used by you, the user

**owned\_private\_repos\_count**

The number of private repositories owned by you, the user

**plan**

---

**Note:** When used with a Github Enterprise instance  $\leq 2.12.7$ , this attribute will not be returned. To handle these situations sensitively, the attribute will be set to `None`. Repositories may still have a license associated with them in these cases.

---

The name of the plan that you, the user, have purchased

**as\_dict ()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json ()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete ()**

Delete the user.

Per GitHub API documentation, it is often preferable to suspend the user.

---

**Note:** This is only available for admins of a GitHub Enterprise instance.

---

**Returns** bool – True if successful, False otherwise

**demote ()**

Demote a site administrator to simple user.

You can demote any user account except your own.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**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** generator of *Events*

**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*

**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*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**impersonate** (*scopes=None*)

Obtain an impersonation token for the user.

The retrieved token will allow impersonation of the user. This is only available for admins of a GitHub Enterprise instance.

**Parameters** **scopes** (*list*) – (optional), areas you want this token to apply to, i.e., ‘gist’, ‘user’

**Returns** *Authorization*

**is\_assignee\_on** (*username, repository*)

Check if this user can be assigned to issues on *username/repository*.

**Parameters**

- **username** (*str*) – owner’s username of the repository
- **repository** (*str*) – name of the repository

**Returns** True if the use can be assigned, False otherwise

**Return type** *bool*

**is\_following** (*username*)

Check if this user is following *username*.

**Parameters** **username** (*str*) – (required)

**Returns** *bool*

**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*

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**organization\_events** (*org*, *number=-1*, *etag=None*)

Iterate over events from the user's organization dashboard.

---

**Note:** 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** generator of *Events*

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

Iterate over organizations the user is member of.

**Parameters**

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

**Returns** generator of *ShortOrganizations*

**promote()**

Promote a user to site administrator.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**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*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**rename** (*login*)

Rename the user.

---

**Note:** This is only available for administrators of a GitHub Enterprise instance.

---

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

**Returns** bool

**revoke\_impersonation** ()

Revoke all impersonation tokens for the current user.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**starred\_repositories** (*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 *StarredRepository*

**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*

**suspend** ()

Suspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**unsuspend** ()

Unsuspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**class** github3.users.**Collaborator** (*json, session*)

Object for the representation of a collaborator.

New in version 1.1.0.

When retrieving a repository’s contributors, GitHub returns the same information as a *ShortUser* with an additional attribute:

**permissions**

Admin, push, and pull permissions of a collaborator

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object’s attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete** ()

Delete the user.

Per GitHub API documentation, it is often preferable to suspend the user.

---

**Note:** This is only available for admins of a GitHub Enterprise instance.

---

**Returns** bool – True if successful, False otherwise

**demote** ()

Demote a site administrator to simple user.

You can demote any user account except your own.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**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** generator of *Events*

**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*

**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*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod** `from_json` (*json*, *session*)

Return an instance of this class formed from `json`.

**impersonate** (*scopes=None*)

Obtain an impersonation token for the user.

The retrieved token will allow impersonation of the user. This is only available for admins of a GitHub Enterprise instance.

**Parameters** `scopes` (*list*) – (optional), areas you want this token to apply to, i.e., ‘gist’, ‘user’

**Returns** `Authorization`

**is\_assignee\_on** (*username*, *repository*)

Check if this user can be assigned to issues on `username/repository`.

**Parameters**

- **username** (*str*) – owner’s username of the repository
- **repository** (*str*) – name of the repository

**Returns** True if the use can be assigned, False otherwise

**Return type** `bool`

**is\_following** (*username*)

Check if this user is following `username`.

**Parameters** `username` (*str*) – (required)

**Returns** `bool`

**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`

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**organization\_events** (*org*, *number=-1*, *etag=None*)

Iterate over events from the user’s organization dashboard.

---

**Note:** 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** generator of *Events*

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

Iterate over organizations the user is member of.

**Parameters**

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

**Returns** generator of *ShortOrganizations*

**promote** ()

Promote a user to site administrator.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**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*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**rename** (*login*)

Rename the user.

---

**Note:** This is only available for administrators of a GitHub Enterprise instance.

---

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

**Returns** bool

**revoke\_impersonation** ()

Revoke all impersonation tokens for the current user.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**starred\_repositories** (*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 *StarredRepository*

**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*

**suspend** ()

Suspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise



**unsuspend()**

Unsuspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**class github3.users.Contributor** (*json, session*)

Object for the specialized representation of a contributor.

New in version 1.0.0.

Changed in version 1.1.0: This class now refreshes to a *User*.

The attribute `contributions` was renamed to `contributions_count`, the documentation already declared it as `contributions_count`, it was the implementation now reflects this as well.

When retrieving a repository's contributors, GitHub returns the same information as a *ShortUser* with an additional attribute:

**contributions\_count**

The number of contributions a contributor has made to the repository

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete the user.

Per GitHub API documentation, it is often preferable to suspend the user.

---

**Note:** This is only available for admins of a GitHub Enterprise instance.

---

**Returns** bool – True if successful, False otherwise

**demote()**

Demote a site administrator to simple user.

You can demote any user account except your own.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**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** generator of *Events*

**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*

**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*

**classmethod from\_dict** (*json\_dict, session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json, session*)

Return an instance of this class formed from *json*.

**impersonate** (*scopes=None*)

Obtain an impersonation token for the user.

The retrieved token will allow impersonation of the user. This is only available for admins of a GitHub Enterprise instance.

**Parameters** **scopes** (*list*) – (optional), areas you want this token to apply to, i.e., ‘gist’, ‘user’

**Returns** *Authorization*

**is\_assignee\_on** (*username, repository*)

Check if this user can be assigned to issues on *username/repository*.

**Parameters**

- **username** (*str*) – owner’s username of the repository
- **repository** (*str*) – name of the repository

**Returns** True if the use can be assigned, False otherwise

**Return type** bool

**is\_following** (*username*)

Check if this user is following *username*.

**Parameters** **username** (*str*) – (required)

**Returns** bool

**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*

**new\_session** ()

Generate a new session.

**Returns** A brand new session

**Return type** *GitHubSession*

**organization\_events** (*org, number=-1, etag=None*)

Iterate over events from the user's organization dashboard.

---

**Note:** 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** generator of *Events*

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

Iterate over organizations the user is member of.

**Parameters**

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

**Returns** generator of *ShortOrganizations*

**promote** ()

Promote a user to site administrator.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**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*

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**rename** (*login*)

Rename the user.

---

**Note:** This is only available for administrators of a GitHub Enterprise instance.

---

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

**Returns** bool

**revoke\_impersonation** ()

Revoke all impersonation tokens for the current user.

This is only available for admins of a GitHub Enterprise instance.

**Returns** bool – True if successful, False otherwise

**starred\_repositories** (*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 *StarredRepository*

**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*

**suspend** ()

Suspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

**unsuspend** ()

Unsuspend the user.

This is only available for admins of a GitHub Enterprise instance.

This API is disabled if you use LDAP, check the GitHub API docs for more information.

**Returns** bool – True if successful, False otherwise

## AuthenticatedUser Peripherals

**class** github3.users.**Key** (*json, session*)

The object representing a user's SSH key.

Please see GitHub's [Key Documentation](#) for more information.

Changed in version 1.0.0: Removed `title` attribute

**key**

A string containing the actual text of the SSH Key

**id**

GitHub's unique ID for this key

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**delete()**

Delete this key.

**classmethod from\_dict(json\_dict, session)**

Return an instance of this class formed from json\_dict.

**classmethod from\_json(json, session)**

Return an instance of this class formed from json.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh(conditional=False)**

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**update** (*title*, *key*)

Update this key.

**Warning:** As of 20 June 2014, the API considers keys to be immutable. This will soon begin to return MethodNotAllowed errors.

**Parameters**

- **title** (*str*) – (required), title of the key
- **key** (*str*) – (required), text of the key file

**Returns** bool

**class** github3.users.**Plan** (*json*, *session*)

The *Plan* object.

Please see GitHub’s [Authenticated User](#) documentation for more details.

**collaborators\_count**

Changed in version 1.0.0.

The number of collaborators allowed on this plan

**name**

The name of the plan on GitHub

**private\_repos\_count**

Changed in version 1.0.0.

The number of allowed private repositories

**space**

The amount of space allotted by this plan

**as\_dict** ()

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object’s attributes serialized to a dictionary

**Return type** dict

**as\_json** ()

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object’s attributes as a JSON string

**Return type** str

**classmethod** `from_dict` (*json\_dict*, *session*)  
Return an instance of this class formed from `json_dict`.

**classmethod** `from_json` (*json*, *session*)  
Return an instance of this class formed from `json`.

**is\_free** ()  
Check if this is a free plan.

**Returns** bool

**new\_session** ()  
Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**  
Number of requests before GitHub imposes a ratelimit.

**Returns** int

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

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

**class** `github3.users.Email` (*json*, *session*)  
The object used to represent an `AuthenticatedUser`'s email.

Please see GitHub's [Emails documentation](#) for more information.

The attributes represented on this object include:

**email**  
The email address as a string

**verified**  
A boolean value representing whether the address has been verified or not

**primary**  
A boolean value representing whether the address is the primary address for the user or not

**as\_dict** ()  
Return the attributes for this object as a dictionary.

This is equivalent to calling:



```
json.loads(obj.as_json())
```

**Returns** this object’s attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object’s attributes as a JSON string

**Return type** str

**classmethod from\_dict** (*json\_dict*, *session*)

Return an instance of this class formed from *json\_dict*.

**classmethod from\_json** (*json*, *session*)

Return an instance of this class formed from *json*.

**new\_session()**

Generate a new session.

**Returns** A brand new session

**Return type** GitHubSession

**ratelimit\_remaining**

Number of requests before GitHub imposes a ratelimit.

**Returns** int

**refresh** (*conditional=False*)

Re-retrieve the information for this object.

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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

## 12.1.15 Internals

### Decorators

This part of the documentation covers the decorators module which contains all of the decorators used in github3.py.

**Warning:** These decorators are only to be used internally in development of this library.

### Decorator Functions

`github3.decorators.requires_auth(x)`

`github3.decorators.requires_basic_auth(x)`

`github3.decorators.requires_app_credentials(func)`

Require `client_id` and `client_secret` to be associated.

This is used to note and enforce which methods require a `client_id` and `client_secret` to be used.

### 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.

**Warning:** These classes are only to be used internally in development of this library.

**class** `github3.models.GitHubCore(json, session)`

The base object for all objects that require a session.

The `GitHubCore` object provides some basic attributes and methods to other sub-classes that are very useful to have.

**as\_dict()**

Return the attributes for this object as a dictionary.

This is equivalent to calling:

```
json.loads(obj.as_json())
```

**Returns** this object's attributes serialized to a dictionary

**Return type** dict

**as\_json()**

Return the json data for this object.

This is equivalent to calling:

```
json.dumps(obj.as_dict())
```

**Returns** this object's attributes as a JSON string

**Return type** str

**classmethod from\_dict** (*json\_dict, session*)  
Return an instance of this class formed from `json_dict`.

**classmethod from\_json** (*json, session*)  
Return an instance of this class formed from `json`.

**new\_session** ()  
Generate a new session.

**Returns** A brand new session

**Return type** `GitHubSession`

**ratelimit\_remaining**  
Number of requests before GitHub imposes a ratelimit.

**Returns** `int`

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

The reasoning for the return value is the following example:

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

Which is really an anti-pattern.

Changed in version 0.5.

**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`



### 13.1 Release Notes and History

All of the release notes that have been recorded for `github3.py` are organized here with the newest releases first.

#### 13.1.1 1.x Release Series

##### 1.1.0: 2018-04-09

This is a small release with some enhancements.

##### Features Added

- `Repository` `collaborators` now returns a `users.Collaborator` object, instead of a `users.ShortUser` object. This is to support collaborator affiliations. A refresh call of this object (and `users.Contributor`) will result in a full `users.User` object.
- The call to iterate collaborators of a repository (`Repository#collaborators`) can now take an `affiliation` filter with options of `outside`, `direct`, and `all`. The default is `all`, which preserves the previous behavior of this method.

##### Bugs Fixed

- Parse certain attributes on `IssueEvent` into objects (again, this was a regression in 1.0)
- Handle older GitHub Enterprise responses for authenticated user objects
- Handle large file pull request responses not including a `patch` attribute

### 1.0.2: 2018-03-28

#### Bugs Fixed

- Handle 304s from users passing etag
- Generate `_api` attribute for `ShortBranch` objects
- Require auth for creating gists
- Ensure only desired auth scheme is used with requests
- Handle older GitHub Enterprise responses for repository objects

### 1.0.1: 2018-03-14

#### Bugs Fixed

- Fix missing `python-dateutil` requirement when installing from a wheel.

### 1.0.0: 2018-03-13

1.0.0 is a huge release. It includes a great deal of changes to `github3.py`. It is suggested you read the following release notes carefully.

Unfortunately, it's plausible that some things have slipped through the cracks in these release notes.

#### Breaking Changes

- Methods that iterate over collections return a separate class than methods that retrieve a single instance. These objects have separate representations when retrieving the data from GitHub's API. They include:
  - Team now can be represented by `ShortTeam` or `Team`
  - Organization now can be represented by `ShortOrganization` or `Organization`
  - Issue now can be represented by `ShortIssue` or `Issue`
  - PullRequest now can be represented by `ShortPullRequest` or `PullRequest`
  - Commit now can be represented by `ShortCommit`, or `Commit`
  - Gist now can be represented by `ShortGist`, `GistFork`, or `Gist`
  - GistFile now can be represented by `ShortGistFile` or `GistFile`
  - Repository objects:
    - \* Branch now can be represented by `ShortBranch` or `Branch`
    - \* RepoComment now can be represented by `ShortComment` or `ShortRepoComment`
    - \* Repository now can be represented by `ShortRepository` or `Repository`
    - \* RepoCommit now can be represented by `MiniCommit`, `ShortCommit`, or `RepoCommit`
    - \* Status now can be represented by `ShortStatus` or `Status`
  - User now can be represented by `ShortUser`, `Contributor`, `User`, or `AuthenticatedUser`
  - License now can be represented by `ShortLicense` or `License`

- Refreshing a short representation of an object will result in a new object of a new class returned. For example:

```
import github3
users = [(u, u.refresh()) for u in github3.all_users(10)]
for short_user, user in users:
    assert isinstance(short_user, github3.users.ShortUser)
    assert isinstance(user, github3.users.User)
```

- Remove `Thread.comment`, `Thread.thread`, `Thread.urls` attributes.
- Remove `Thread#is_unread` method. Use the `Thread.unread` attribute instead.
- Subscription has been split into two objects: `ThreadSubscription` and `RepositorySubscription` with the same methods.
- Remove `is_ignored` method from our Subscription objects. Use the `ignored` attribute instead.
- Remove `is_subscribed` method from our Subscription objects. Use the `subscribed` attribute instead.
- Move `Users#add_email_addresses` to `GitHub#add_email_addresses`.
- Move `Users#delete_email_addresses` to `GitHub#delete_email_addresses`.
- Remove `Users#add_email_address` and `Users#delete_email_address`.
- Remove `Repository#update_label`.
- When you download a release asset, instead of returning `True` or `False`, it will return the name of the file in which it saved the asset.
- The download method on `github3.pulls.PullFile` instances has been removed.
- The `contents` method on `github3.pulls.PullFile` instances now return instances of `github3.repos.contents.Contents`.
- Replace `Repository#comments_on_commit` with `RepoCommit#comments`.
- `Organization#add_member` has been changed. The second parameter has been changed to `team_id` and now expects an integer.
- `Organization#add_repository` has been changed. The second parameter has been changed to `team_id` and now expects an integer.
- All methods and functions starting with `iter_` have been renamed.

Old name	New name
<code>github3.iter_all_repos</code>	<code>github3.all_repositories</code>
<code>github3.iter_all_users</code>	<code>github3.all_users</code>
<code>github3.iter_events</code>	<code>github3.all_events</code>
<code>github3.iter_followers</code>	<code>github3.followers_of</code>
<code>github3.iter_following</code>	<code>github3.followed_by</code>
<code>github3.iter_repo_issues</code>	<code>github3.issues_on</code>
<code>github3.iter_orgs</code>	<code>github3.organizations_with</code>
<code>github3.iter_user_repos</code>	<code>github3.repositories_by</code>
<code>github3.iter_starred</code>	<code>github3.starred_by</code>
<code>github3.iter_subscriptions</code>	<code>github3.subscriptions_for</code>
<code>Deployment#iter_statuses</code>	<code>Deployment#statuses</code>
<code>Gist#iter_comments</code>	<code>Gist#comments</code>
<code>Gist#iter_commits</code>	<code>Gist#commits</code>
<code>Gist#iter_files</code>	<code>Gist#files</code>

Continued on next page

Table 1 – continued from previous page

Old name	New name
Gist#iter_forks	Gist#forks
GitHub#iter_all_repos	GitHub#all_repositories
GitHub#iter_all_users	GitHub#all_users
GitHub#iter_authorizations	GitHub#authorizations
GitHub#iter_emails	GitHub#emails
GitHub#iter_events	GitHub#events
GitHub#iter_followers	GitHub#{followers, followers_of}
GitHub#iter_following	GitHub#{following, followed_by}
GitHub#iter_gists	GitHub#{gists, gists_by, public_gists}
GitHub#iter_notifications	GitHub#notifications
GitHub#iter_org_issues	GitHub#organization_issues
GitHub#iter_issues	GitHub#issues
GitHub#iter_user_issues	GitHub#user_issues
GitHub#iter_repo_issues	GitHub#issues_on
GitHub#iter_keys	GitHub#keys
GitHub#iter_orgs	GitHub#{organizations, organizations_with}
GitHub#iter_repos	GitHub#repositories
GitHub#iter_user_repos	GitHub#repositories_by
GitHub#iter_user_teams	GitHub#user_teams
Issue#iter_comments	Issue#comments
Issue#iter_events	Issue#events
Issue#iter_labels	Issue#labels
Milestone#iter_labels	Milestone#labels
Organization#iter_members	Organization#members
Organization#iter_public_members	Organization#public_members
Organization#iter_repos	Organization#repositories
Organization#iter_teams	Organization#teams
PullRequest#iter_comments	PullRequest#review_comments
PullRequest#iter_commits	PullRequest#commits
PullRequest#iter_files	PullRequest#files
PullRequest#iter_issue_comments	PullRequest#issue_comments
Team#iter_members	Team#members
Team#iter_repos	Team#repositories
Repository#iter_assignees	Repository#assignees
Repository#iter_branches	Repository#branches
Repository#iter_code_frequency	Repository#code_frequency
Repository#iter_collaborators	Repository#collaborators
Repository#iter_comments	Repository#comments
Repository#iter_comments_on_commit	RepoCommit#comments
Repository#iter_commit_activity	Repository#commit_activity
Repository#iter_commits	Repository#commits
Repository#iter_contributor_statistics	Repository#contributor_statistics
Repository#iter_contributors	Repository#contributors
Repository#iter_forks	Repository#forks
Repository#iter_hooks	Repository#hooks
Repository#iter_issues	Repository#issues
Repository#iter_issue_events	Repository#issue_events
Repository#iter_keys	Repository#keys
Repository#iter_labels	Repository#labels

Continued on next page



Table 1 – continued from previous page

Old name	New name
Repository#iter_languages	Repository#languages
Repository#iter_milestones	Repository#milestones
Repository#iter_network_events	Repository#network_events
Repository#iter_notifications	Repository#notifications
Repository#iter_pages_builds	Repository#pages_builds
Repository#iter_pulls	Repository#pull_requests
Repository#iter_refs	Repository#refs
Repository#iter_releases	Repository#releases
Repository#iter_stargazers	Repository#stargazers
Repository#iter_subscribers	Repository#subscribers
Repository#iter_statuses	Repository#statuses
Repository#iter_tags	Repository#tags
Repository#iter_teams	Repository#teams
Repository#iter_teams	Repository#teams
User#iter_events	User#events
User#iter_followers	User#followers
User#iter_following	User#following
User#iter_keys	User#keys
User#iter_org_events	User#organization_events
User#iter_received_events	User#received_events
User#iter_orgs	User#organizations
User#iter_starred	User#starred_repositories
User#iter_subscriptions	User#subscriptions

- `github3.login` has been simplified and split into two functions:
  - `github3.login` serves the majority use case and only provides an authenticated GitHub object.
  - `github3.enterprise_login` allows GitHub Enterprise users to log into their service.
- `GitHub#iter_followers` was split into two functions:
  - `GitHub#followers_of` which iterates over all of the followers of a user whose username you provide
  - `GitHub#followers` which iterates over all of the followers of the authenticated user
- `GitHub#iter_following` was split into two functions:
  - `GitHub#followed_by` which iterates over all of the users followed by the username you provide
  - `GitHub#following` which iterates over all of the users followed by the authenticated user
- `GitHub#iter_gists` was split into three functions:
  - `GitHub#public_gists` which iterates over all of the public gists on GitHub
  - `GitHub#gists_for` which iterates over all the public gists of a specific user
  - `GitHub#gists` which iterates over the authenticated users gists
- `GitHub#iter_orgs` was split into two functions:
  - `GitHub#organizations` which iterates over the authenticated user’s organization memberships
  - `GitHub#organizations_with` which iterates over the given user’s organization memberships
- `GitHub#iter_subscriptions` was split into two functions:
  - `GitHub#subscriptions_for` which iterates over an arbitrary user’s subscriptions

- `GitHub#subscriptions` which iterates over the authenticated user's subscriptions
- `GitHub#iter_starred` was split into two functions:
  - `GitHub#starred_by` which iterates over an arbitrary user's stars
  - `GitHub#starred` which iterates over the authenticated user's stars
- `GitHub#user` was split into two functions:
  - `GitHub#user` which retrieves an arbitrary user's information
  - `GitHub#me` which retrieves the authenticated user's information
- `GitHub#update_user` has been renamed to `GitHub#update_me` and only uses 1 API call now. It was renamed to reflect the addition of `GitHub#me`.
- The legacy watching API has been removed:
  - `GitHub#subscribe`
  - `GitHub#unsubscribe`
  - `GitHub#is_subscribed`
- `GitHub#create_repo` was renamed to `GitHub#create_repository`
- `GitHub#delete_key` was removed. To delete a key retrieve it with `GitHub#key` and then call `Key#delete`.
- `Repository#set_subscription` was split into two simpler functions
  - `Repository#subscribe` subscribes the authenticated user to the repository's notifications
  - `Repository#ignore` ignores notifications from the repository for the authenticated user
- `Repository#contents` was split into two simpler functions
  - `Repository#file_contents` returns the contents of a file object
  - `Repository#directory_contents` returns the contents of files in a directory.
- `Organization#add_repo` and `Team#add_repo` have been renamed to `Organization#add_repository` and `Team#add_repository` respectively.
- `Organization#create_repo` has been renamed to `Organization#create_repository`. It no longer accepts `has_downloads`. It now accepts `license_template`.
- `Organization#remove_repo` has been renamed to `Organization#remove_repository`. It now accepts `team_id` instead of `team`.
- `github3.ratelimit_remaining` was removed
- `GitHub` instances can no longer be used as context managers
- The pull request API has changed.
  - The `links` attribute now contains the raw `_links` attribute from the API.
  - The `merge_commit_sha` attribute has been removed since it was deprecated in the GitHub API.
  - To present a more consistent universal API, certain attributes have been renamed.

Old name	New attribute name
PullFile.additions	additions_count
PullFile.deletions	deletions_count
PullFile.changes	changes_count
PullRequest.additions	additions_count
PullRequest.comments	comments_count
PullRequest.commits	commits_count
PullRequest.deletions	deletions_count
PullRequest.review_comments	review_comments_count

- The Gist API has changed.
  - The `forks` and `files` attributes that used to keep count of the number of forks and files have been **removed**.
  - The `comments` attribute which provided the number of comments on a gist, has been **renamed** to `comments_count`.
  - The `is_public` method has been removed since it just returned the `Gist.public` attribute.
- Most instances of `login` as a parameter have been changed to `username` for clarity and consistency. This affects the following methods:
  - `github3.authorize`
  - `github3.repositories_by`
  - `github3.user`
  - `GitHub`
  - `GitHub#authorize`
  - `GitHub#follow`
  - `GitHub#is_following`
  - `GitHub#is_starred`
  - `GitHub#issue`
  - `GitHub#followers_of`
  - `GitHub#followed_by`
  - `GitHub#gists_by`
  - `GitHub#issues_on`
  - `GitHub#organizations_with`
  - `GitHub#starred_by`
  - `GitHub#subscriptions_for`
  - `GitHub#user`
  - `GitHubEnterprise`
  - `Issue#assign`
  - `Organization#add_member`
  - `Organization#is_member`
  - `Organization#is_public_member`

- Organization#remove\_member
- Repository#add\_collaborator
- Repository#is\_assignee
- Repository#is\_collaborator
- Repository#remove\_collaborator
- Team#add\_member
- Team#is\_member
- User#is\_assignee\_on
- User#is\_following

- Repository.stargazers is now Repository.stargazers\_count (conforming with the attribute name returned by the API).
- The Issue API has changed in order to provide a more consistent attribute API. Issue.comments is now Issue.comments\_count and returns the number of comments on an issue.
- The Issue.labels attribute has also been renamed. It is now available from Issue.original\_labels. This will provide the user with the list of Label objects that was returned by the API. To retrieve an updated list of labels, the user can now use Issue#labels, e.g.

```
i = github3.issue('sigmavirus24', 'github3.py', 30)
labels = list(i.labels())
```

- The Organization and User APIs have changed to become more consistent with the rest of the library and GitHub API. The following attribute names have been changed

Old name	New attribute name
Organization.followers	followers_count
Organization.following	following_count
Organization.public_repos	public_repos_count
User.followers	followers_count
User.following	following_count
User.public_repos	public_repos_count

- The Release.assets attribute has been renamed to Release.original\_assets. To retrieve up-to-date assets, use the Release#assets method.
- The Authorization API has changed. The update method has been split into three methods: add\_scopes, remove\_scopes, replace\_scopes. This highlights the fact that Authorization#update used to require more than one request.
- Event#is\_public has been removed. Simply check the event's public attribute instead.
- Repository#delete\_file and Repository#update\_file have been removed. Simply delete or update a file using the Contents API.
- Content#delete now returns a dictionary that matches the JSON returned by the API. It contains the Contents and the Commit associated with the deletion.
- Content#update now returns a dictionary that matches the JSON returned by the API. It contains the Contents and the Commit associated with the deletion.
- Issue.pull\_request has been renamed to Issue.pull\_request\_urls

## New Features

- Most objects now have a `session` attribute. This is a subclass of a `Session` object from `requests`. This can now be used in conjunction with a third-party caching mechanism. The suggested caching library is `cachecontrol`.
- All object's `url` attribute are now available.
- You can now retrieve a repository by its id with `GitHub#repository_with_id`.
- You can call the `pull_request` method on an `Issue` now to retrieve the associated pull request:

```
import github3

i = github3.issue('sigmavirus24', 'github3.py', 301)
pr = i.pull_request()
```

- Add support for the Issue locking API currently in Preview Mode
- Add `Organization#all_events`.
- Add `Tag.tagger_as_User` which attempts to return the tagger as as `User`.
- Add `Repo.statuses` and a corresponding `repo.status.CombinedStatus` to
- Support filtering organization members by whether they have 2FA enabled.
- Support filtering organization and team members by role.
- Add `GitHub#all_organizations`.
- Add `PullRequest#create_comment`.
- Add `Repository#release_by_tag_name` to retrieve a `Release` from a `Repository` by its associated tag name.
- Add `Repository#latest_release` to retrieve the latest `Release` for a `Repository`.
- Add `GitHub#license` to retrieve a `github3.license.License` by the license name.
- Add `GitHub#licenses` to iterate over all the licenses returned by GitHub's Licenses API.
- Add protection information to `github3.repos.branch.Branch`.
- Add `Branch#protect` and `Branch#unprotect` to support updating a `Branch`'s protection status.
- Vastly improved GitHub Enterprise support:
  - Add `User#rename` to rename a user in a GitHub Enterprise installation.
  - Add `GitHub#create_user` to create a user.
  - Add `User#impersonate` to create an impersonation token by an admin for a particular user.
  - Add `User#revoke_impersonation` to revoke all impersonation tokens for a user.
  - Add `User#promote` to promote a particular user to a site administrator.
  - Add `User#demote` to demote a site administrator to a simple user.
  - Add `User#suspend` to suspend a user's account.
  - Add `User#unsuspend` to reinstate a user's account.
- Add `original_content` attribute to a `GistFile`
- Add `GistFile#content` to retrieve the contents of a file in a gist from the API.

- Add support for the alpha `bulk issue import` API
- You can now download a file in a pull request to a file on disk.
- You can retrieve the contents of the file in a pull request as bytes.
- Add `id` attribute to `github3.repos.milestone.Milestone`.
- Add support for `sort`, `direction`, and `since` parameters to the `comments` method on `github3.issues.Issue`.
- Add `branch` argument to `update` and `delete` methods on `github3.repos.contents.Contents`.
- Add `permissions` attribute to `github3.repos.repo.Repository` object to retrieve the permissions for a specific repository.
- Allow a deployment to be retrieved by its `id`.
- Add the `delete` method to the `github3.repos.release.Asset` class.

### Bugs Fixed

- Fix the dependencies and requirements. In 1.0.0a3 we moved to using the `setup.cfg` file to define optional dependencies for wheels. By doing so we accidentally left out our actual hard dependencies.
- The `context` parameter to `Repository#create_status` now properly defaults to "default".
- Fix `AttributeError` when `IssueEvent` has `assignee`.
- Correctly set the `message` attribute on `RepoCommit` instances.
- Include `browser_download_url` on `Asset` instances.
- (Packaging related) Fix `setup.py` to use proper values for certain parameters.
- Fix `ValueError` for `Repository#create_file`.
- Pull request files can now be downloaded even when the repository is private.
- Fix exception when merging a pull request with an empty commit message.
- Add missing Issue events.
- Coerce review comment positions to integers.

### Deprecations and Other Changes

- Deprecate `Organization#events` in favor of `Organization#public_events`.
- Fix test failures on windows caused by unclosed file handles. `get` a combined view of commit statuses for a given ref.
- The `refresh` method will eventually stop updating the instance in place and instead only return new instances of objects.

## 13.1.2 0.x Release Series

### 0.9.3: 2014-11-04

- Backport of `PullRequest#create_review_comment` by Adrian Moisey

- Backport of `PullRequest#review_comments` by Adrian Moisey
- Backport of a fix that allows authenticated users to download Release Assets. Original bug reported by Eugene Fidelin in issue #288.
- Documentation typo fix by Marc Abramowitz

### 0.9.2: 2014-10-05

- Updates for new team management API changes
  - Add `Team#invite`, `Team#membership_for`, and `Team#revoke_membership`
  - Deprecate `Team#add_member`, `Team#remove_member`, and `Organization#add_member`.
  - Update payload handler for `TeamAddEvent`.

### 0.9.1: 2014-08-10

- Correct Repository attribute `fork_count` should be `forks_count`

### 0.9.0: 2014-05-04

- Add Deployments API
- Add Pages API
- Add support so applications can revoke a `single authorization` or `all authorizations` created by the application
- Add the ability for users to ping hooks
- Allow users to list a `Repository's collaborators`
- Allow users to create an empty blob on a Repository
- Update how users can list issues and pull requests. See: <http://developer.github.com/changes/2014-02-28-issue-and-pull-query-enhancements/> This includes breaking changes to `Repository#iter_pulls`.
- Update methods to handle the `pagination changes`.
- Fix typo `stargazers_url`
- Add `assets` attribute to Release object.
- Fix wrong argument to `Organization#create_team` (`permissions` versus `permission`)
- Fix Issue Search Result's representation and initialization
- Fix Repository Search Result's initialization
- Allow users to pass a two-factor authentication callback to `GitHub#authorize`.

### 0.8.2: 2014-02-11

- Fix bug in `GitHub#search_users` (and `github3.search_users`). Thanks @abesto
- Expose the stargazers count for repositories. Thanks @seveas

### 0.8.1: 2014-01-26

- Add documentation for using Two Factor Authentication
- Fix oversight where `github3.login` could not be used for 2FA

### 0.8.0: 2014-01-03

- **Breaking Change** Remove legacy search API  
I realize this should have been scheduled for 1.0 but I was a bit eager to remove this.
- Use Betamax to start recording integration tests
- Add support for Releases API
- Add support for Feeds API
- Add support for Two-Factor Authentication via the API
- Add support for New Search API
  - Add `github3.search_code`, `github3.search_issues`, `github3.search_repositories`, `github3.search_users`
  - Add `GitHub#search_code`, `GitHub#search_issues`, `GitHub#search_repositories`, `GitHub#search_users`
- Switch to requests  $\geq 2.0$
- Totally remove all references to the Downloads API
- Fix bug in `Repository#update_file` where `branch` was not being sent to the API. Thanks @tpetr!
- Add `GitHub#rate_limit` to return all of the information from the `/rate_limit` endpoint.
- Catch missing attributes `-diff_hunk`, `original_commit_id` – on `ReviewComment`.
- Add support for the Emojis endpoint
- Note deprecation of a few object attributes
- Add support for the `ReleaseEvent`
- Add `GitHub#iter_user_teams` to return all of the teams the authenticated user belongs to

### 0.7.1: 2013-09-30

- Add dependency on `uritemplate.py` to add URITemplates to different classes. See the documentation for attributes which are templates.
- Fixed issue trying to parse `html_url` on Pull Requests courtesy of @rogerhu.
- Remove `expecter` as a test dependency courtesy of @esacteksab.
- Fixed issue #141 trying to find an Event that doesn't exist.

### 0.7.0: 2013-05-19

- Fix `Issue.close`, `Issue.reopen`, and `Issue.assign`. (Issue #106)
- Add `check_authorization` to the `GitHub` class to cover the new part of the API.



- Add `create_file`, `update_file`, `delete_file`, `iter_contributor_statistics`, `iter_commit_activity`, `iter_code_frequency` and `weekly_commit_count` to the `Repository` object.
- Add `update` and `delete` methods to the `Contents` object.
- Add `is_following` to the `User` object.
- Add `head`, `base` parameters to `Repository.iter_pulls`.
- The signature of `Hook.edit` has changed since that endpoint has changed as well. See: [github/developer.github.com@b95f291a47954154a6a8cd7c2296cdda9b610164](https://github.com/developer.github.com@b95f291a47954154a6a8cd7c2296cdda9b610164)
- `github3.GitHub` can now be used as a context manager, e.g.,

```
with github.GitHub() as gh:
    u = gh.user('sigmavirus24')
```

### 0.6.1: 2013-04-06

- Add equality for labels courtesy of Alejandro Gomez (@alejandrogomez)

### 0.6.0: 2013-04-05

- 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 a bit. (Split up `repos.py`, `issues.py`, `gists.py` and a few others into sub-modules for my sanity.)
- 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.
- Make `github3.octocat` and `github3.github.GitHub.octocat` both support sending messages to make the Octocat say things. (Think cowsay)
- Remove vendored dependency of PySO8601.
- Split `GitHub.iter_repos` into `GitHub.iter_user_repos` and `GitHub.iter_repos`. As a consequence `github3.iter_repos` is now `github3.iter_user_repos`
- `IssueComment.update` was corrected to match GitHub's documentation
- `github3.login` now accepts an optional `url` parameter for users of the GitHubEnterprise API, courtesy of Kristian Glass (@doismellburning)
- Several classes now allow their instances to be compared with `==` and `!=`. In most cases this will check the unique id provided by GitHub. In others, it will check SHAs and any other guaranteed immutable and unique attribute. The class doc-strings all have information about this and details about how equivalence is determined.

### 0.5.3: 2013-03-19

- Add missing optional parameter to `Repository.contents`. Thanks @tpetr

### 0.5.2: 2013-03-02

- Stop trying to decode the byte strings returned by `b64decode`. Fixes #72

### 0.5.1: 2013-02-21

- Hot fix an issue when a user doesn't have a real name set

### 0.5.0: 2013-02-16

- 100% (mock) test coverage
- Add support for the [announced meta](#) endpoint.
- Add support for conditional refreshing, e.g.,

```
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:

```
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.issue.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.

#### 0.4.0: 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

#### 0.3.0: 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 `github3.notifications.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()`
- `github3.orgs.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:

```
import github3
u = github3.user('sigmavirus24')
r = github3.repository(u, 'github3.py')
```

And

```
import github3

repo = github3.repository('sigmavirus24', 'github3.py')

template = """Some kind of template where you mention this repository
{0}"""

print(template.format(repo))
# Some kind of template where you mention this repository
# sigmavirus24/github3.py
```

Current list of objects with this feature:

- `github3.users.User` (uses the login name)
- `github3.users.Key` (uses the key text)
- `github3.users.Repository` (uses the login/name pair)
- `github3.users.RepoTag` (uses the tag name)
- `github3.users.Contents` (uses the decoded content)
- 60% test coverage with mock
- Upgrade to requests 1.0.x

**0.2.0: 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

**0.1.0: 2012-11-13**

- Add API for GitHub Enterprise customers.

**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)

**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

**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



All development happens on [GitHub](#). Please remember to add yourself to the list of contributors in `AUTHORS.rst`, especially if you're going to be working on the list below.

### 14.1 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.

Otherwise, feel free to example the list of issues where we would like [help](#) and feel free to take one.

### 14.2 Running the Unittests

The tests are generally run using `tox`. `Tox` can be installed like so

```
pip install tox
```

We test against PyPy and the following versions of Python:

- 2.7
- 3.4
- 3.5
- 3.6

If you simply run `tox` it will run tests against all of these versions of python and run `flake8` against the codebase as well. If you want to run against one specific version, you can do

```
tox -e py36
```

And if you want to run tests against a specific file, you can do

```
tox -e py36 -- tests/unit/test_github.py
```

To run the tests, `tox` uses `py.test` so you can pass any options or parameters to `py.test` after specifying `--`. For example, you can get more verbose output by doing

```
tox -e py36 -- -vv
```

### 14.2.1 Writing Tests for github3.py

#### Unit Tests

In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application.

—Unit Testing on Wikipedia

In `github3.py` we use unit tests to make assertions about how the library behaves without making a request to the internet. For example, one assertion we might write would check if custom information is sent along in a request to GitHub.

An existing test like this can be found in `tests/unit/test_repos_release.py`:

```
def test_delete(self):
    self.instance.delete()
    self.session.delete.assert_called_once_with(
        self.example_data['url'],
        headers={'Accept': 'application/vnd.github.manifold-preview'}
    )
```

In this test, we check that the library passes on important headers to the API to ensure the request will work properly. `self.instance` is created for us and is an instance of the `Release` class. The test then calls `delete` to make a request to the API. `self.session` is a mock object which fakes out a normal session. It does not allow the request through but allows us to verify how `github3.py` makes a request. We can see that `github3.py` called `delete` on the session. We assert that it was only called once and that the only parameters sent were a URL and the custom headers that we are concerned with.

#### Mocks

Above we talked about mock objects. What are they?

In object-oriented programming, mock objects are simulated objects that mimic the behavior of real objects in controlled ways. A programmer typically creates a mock object to test the behavior of some other object, in much the same way that a car designer uses a crash test dummy to simulate the dynamic behavior of a human in vehicle impacts.

—Mock Object on Wikipedia



We use mocks in `github3.py` to prevent the library from talking directly with GitHub. The mocks we use intercept requests the library makes so we can verify the parameters we use. In the example above, we were able to check that certain parameters were the only ones sent to a session method because we mocked out the session.

You may have noticed in the example above that we did not have to set up the mock object. There is a convenient helper written in `tests/unit/helper.py` to do this for you.

### Example - Testing the Release Object

Here's a full example of how we test the `Release` object in `tests/unit/test_repos_release.py`.

Our first step is to import the `UnitHelper` class from `tests/unit/helper.py` and the `Release` object from `github3/repos/release.py`.

```
from .helper import UnitHelper
from github3.repos.release import Release
```

Then we construct our test class and indicate which class we will be testing (or describing).

```
class TestRelease(UnitHelper):
    described_class = Release
```

We can then use the [GitHub API documentation about Releases](#) to retrieve example release data. We then can use that as example data for our test like so:

```
class TestRelease(UnitHelper):
    described_class = Release
    example_data = {
        "url": releases_url("/1"),
        "html_url": "https://github.com/octocat/Hello-World/releases/v1.0.0",
        "assets_url": releases_url("/1/assets"),
        "upload_url": releases_url("/1/assets{?name}"),
        "id": 1,
        "tag_name": "v1.0.0",
        "target_commitish": "master",
        "name": "v1.0.0",
        "body": "Description of the release",
        "draft": False,
        "prerelease": False,
        "created_at": "2013-02-27T19:35:32Z",
        "published_at": "2013-02-27T19:35:32Z"
    }
```

The above code now will handle making clean and brand new instances of the `Release` object with the example data and a faked out session. We can now construct our first test.

```
def test_delete(self):
    self.instance.delete()
    self.session.delete.assert_called_once_with(
        self.example_data['url'],
        headers={'Accept': 'application/vnd.github.manifold-preview'}
    )
```

### Integration Tests

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items.

—Integration tests on Wikipedia

In `github3.py` we use integration tests to ensure that when we make what should be a valid request to GitHub, it is in fact valid. For example, if we were testing how `github3.py` requests a user's information, we would expect a request for a real user's data to be valid. If the test fails we know either what the library is doing is wrong or the data requested does not exist.

An existing test that demonstrates integration testing can be found in `tests/integration/test_repos_release.py`:

```
def test_iter_assets(self):
    """Test the ability to iterate over the assets of a release."""
    cassette_name = self.cassette_name('iter_assets')
    with self.recorder.use_cassette(cassette_name):
        repository = self.gh.repository('sigmavirus24', 'github3.py')
        release = repository.release(76677)
        for asset in release.iter_assets():
            assert isinstance(asset, github3.repos.release.Asset)
            assert asset is not None
```

In this test we use `self.recorder` to record our interaction with GitHub. We then proceed to make the request to GitHub that will exercise the code we wish to test. First we request a `Repository` object from GitHub and then using that we request a `Release` object. After receiving that release, we exercise the code that lists the assets of a `Release`. We verify that each asset is an instance of the `Asset` class and that at the end the `asset` variable is not `None`. If `asset` was `None`, that would indicate that GitHub did not return any data and it did not exercise the code we are trying to test.

### Betamax

`Betamax` is the library that we use to create the recorder above. It sets up the session object to intercept every request and corresponding response and save them to what it calls `cassettes`. After you record the interaction it never has to speak to the internet again for that request.

In `github3.py` there is a helper class (much like `UnitHelper`) in `tests/integration/helper.py` which sets everything up for us.

### Example - Testing the Release Object

Here's an example of how we write an integration test for `github3.py`. The example can be found in `tests/integration/test_repos_release.py`.

Our first steps are the necessary imports.

```
import github3

from .helper import IntegrationHelper
```

Then we start writing our test right away.

```

class TestRelease(IntegrationHelper):
    def test_delete(self):
        """Test the ability to delete a release."""
        self.token_login()
        cassette_name = self.cassette_name('delete')
        with self.recorder.use_cassette(cassette_name):
            repository = self.gh.repository('github3py', 'github3.py')
            release = repository.create_release(
                '0.8.0.pre', 'develop', '0.8.0 fake release',
                'To be deleted'
            )
            assert release is not None
            assert release.delete() is True

```

Every test has access to `self.gh` which is an instance of `GitHub`. `IntegrationHelper` provides a lot of methods that allow you to focus on what we are testing instead of setting up for the test. The first of those methods we see in use is `self.token_login` which handles authenticating with a token. It's sister method is `self.basic_login` which handles authentication with basic credentials. Both of these methods will set up the authentication for you on `self.gh`.

The next convenience method we see is `self.cassette_name`. It constructs a cassette name for you based on the test class name and the string you provide it.

Every test also has access to `self.recorder`. This is the Betamax recorder that has been set up for you to record your interactions. The recorder is started when you write

```

with self.recorder.use_cassette(cassette_name):
    # ...

```

Everything that talks to GitHub should be written inside of the context created by the context manager there. No requests to GitHub should be made outside of that context.

In that context, we then retrieve a repository and create a release for it. We want to be sure that we will be deleting something that exists so we assert that what we received back from GitHub is not `None`. Finally we call `delete` and assert that it returns `True`.

When you write your new test and record a new cassette, be sure to add the new cassette file to the repository, like so:

```
git add tests/cassettes/Release_delete.json
```

## Recording Cassettes that Require Authentication/Authorization

If you need to write a test that requires an Authorization (i.e., OAuth token) or Authentication (i.e., username and password), all you need to do is set environment variables when running `py.test`, e.g.,

```

GH_AUTH="abc123" py.test
GH_USER="sigmavirus24" GH_PASSWORD="super-secure-password-plz-kthxbai" py.test

```

If you are concerned that your credentials will be saved, you need not worry. Betamax sanitizes information like that before saving the cassette. It never does hurt to double check though.



## CHAPTER 15

---

### Contact

---

- Twitter: [@sigmavirus24](#)
- Private email: [graffatcolmingov \[at\] gmail](mailto:graffatcolmingov@gmail.com)



**g**

`github3`, [29](#)

`github3.api`, [29](#)





## A

- active (github3.orgs.Membership attribute), 104
- active (github3.repos.hook.Hook attribute), 190
- actor (github3.events.Event attribute), 42
- actor (github3.issues.event.IssueEvent attribute), 86
- add\_collaborator() (github3.repos.repo.Repository method), 138
- add\_collaborator() (github3.repos.repo.ShortRepository method), 164
- add\_contexts() (github3.repos.branch.ProtectionRequiredStatusChecks method), 194
- add\_email\_addresses() (github3.github.GitHub method), 53
- add\_labels() (github3.issues.issue.Issue method), 80
- add\_labels() (github3.issues.issue.ShortIssue method), 76
- add\_member() (github3.orgs.Organization method), 114
- add\_member() (github3.orgs.ShortOrganization method), 107
- add\_member() (github3.orgs.ShortTeam method), 98
- add\_member() (github3.orgs.Team method), 101
- add\_repository() (github3.orgs.Organization method), 114
- add\_repository() (github3.orgs.ShortOrganization method), 107
- add\_repository() (github3.orgs.ShortTeam method), 98
- add\_repository() (github3.orgs.Team method), 101
- add\_scopes() (github3.auths.Authorization method), 40
- additions (github3.gists.history.GistHistory attribute), 49
- additions\_count (github3.pulls.PullFile attribute), 132
- additions\_count (github3.pulls.PullRequest attribute), 126
- admin\_stats() (github3.github.GitHubEnterprise method), 74
- all\_events() (github3.github.GitHub method), 53
- all\_events() (github3.orgs.Organization method), 114
- all\_events() (github3.orgs.ShortOrganization method), 107
- all\_events() (in module github3), 32
- all\_organizations() (github3.github.GitHub method), 53
- all\_repositories() (github3.github.GitHub method), 53
- all\_repositories() (in module github3), 31
- all\_users() (github3.github.GitHub method), 54
- all\_users() (in module github3), 32
- allow\_merge\_commit (github3.repos.repo.Repository attribute), 136
- allow\_rebase\_merge (github3.repos.repo.Repository attribute), 137
- allow\_squash\_merge (github3.repos.repo.Repository attribute), 137
- alternate\_weeks (github3.repos.stats.ContributorStats attribute), 204
- api() (github3.github.GitHubStatus method), 74
- app (github3.auths.Authorization attribute), 39
- archive() (github3.repos.release.Release method), 198
- archive() (github3.repos.repo.Repository method), 138
- archive() (github3.repos.repo.ShortRepository method), 164
- archive\_url (github3.repos.repo.ShortRepository attribute), 161
- archived (github3.repos.repo.Repository attribute), 137
- as\_dict() (github3.auths.Authorization method), 40
- as\_dict() (github3.events.Event method), 42
- as\_dict() (github3.github.GitHub method), 54
- as\_dict() (github3.issues.comment.IssueComment method), 85
- as\_dict() (github3.issues.event.IssueEvent method), 86
- as\_dict() (github3.issues.issue.Issue method), 80
- as\_dict() (github3.issues.issue.ShortIssue method), 76
- as\_dict() (github3.issues.label.Label method), 88
- as\_dict() (github3.issues.milestone.Milestone method), 90
- as\_dict() (github3.models.GitHubCore method), 238
- as\_dict() (github3.notifications.RepositorySubscription method), 96
- as\_dict() (github3.notifications.Thread method), 92
- as\_dict() (github3.notifications.ThreadSubscription method), 94
- as\_dict() (github3.orgs.Membership method), 105
- as\_dict() (github3.orgs.Organization method), 114

- as\_dict() (github3.orgs.ShortOrganization method), 107
- as\_dict() (github3.orgs.ShortTeam method), 98
- as\_dict() (github3.orgs.Team method), 101
- as\_dict() (github3.pulls.PullDestination method), 131
- as\_dict() (github3.pulls.PullFile method), 133
- as\_dict() (github3.pulls.PullRequest method), 126
- as\_dict() (github3.pulls.ReviewComment method), 135
- as\_dict() (github3.pulls.ShortPullRequest method), 121
- as\_dict() (github3.repos.repo.Repository method), 139
- as\_dict() (github3.repos.repo.ShortRepository method), 164
- as\_dict() (github3.repos.repo.StarredRepository method), 187
- as\_dict() (github3.structs.GitHubIterator method), 206
- as\_dict() (github3.structs.SearchIterator method), 207
- as\_dict() (github3.users.AuthenticatedUser method), 220
- as\_dict() (github3.users.Collaborator method), 224
- as\_dict() (github3.users.Contributor method), 229
- as\_dict() (github3.users.Email method), 236
- as\_dict() (github3.users.Key method), 233
- as\_dict() (github3.users.Plan method), 235
- as\_dict() (github3.users.ShortUser method), 210
- as\_dict() (github3.users.User method), 215
- as\_json() (github3.auths.Authorization method), 40
- as\_json() (github3.events.Event method), 43
- as\_json() (github3.github.GitHub method), 54
- as\_json() (github3.issues.comment.IssueComment method), 85
- as\_json() (github3.issues.event.IssueEvent method), 87
- as\_json() (github3.issues.issue.Issue method), 80
- as\_json() (github3.issues.issue.ShortIssue method), 76
- as\_json() (github3.issues.label.Label method), 88
- as\_json() (github3.issues.milestone.Milestone method), 90
- as\_json() (github3.models.GitHubCore method), 238
- as\_json() (github3.notifications.RepositorySubscription method), 96
- as\_json() (github3.notifications.Thread method), 92
- as\_json() (github3.notifications.ThreadSubscription method), 94
- as\_json() (github3.orgs.Membership method), 105
- as\_json() (github3.orgs.Organization method), 115
- as\_json() (github3.orgs.ShortOrganization method), 108
- as\_json() (github3.orgs.ShortTeam method), 98
- as\_json() (github3.orgs.Team method), 102
- as\_json() (github3.pulls.PullDestination method), 131
- as\_json() (github3.pulls.PullFile method), 133
- as\_json() (github3.pulls.PullRequest method), 127
- as\_json() (github3.pulls.ReviewComment method), 135
- as\_json() (github3.pulls.ShortPullRequest method), 122
- as\_json() (github3.repos.repo.Repository method), 139
- as\_json() (github3.repos.repo.ShortRepository method), 165
- as\_json() (github3.repos.repo.StarredRepository method), 187
- as\_json() (github3.structs.GitHubIterator method), 206
- as\_json() (github3.structs.SearchIterator method), 207
- as\_json() (github3.users.AuthenticatedUser method), 220
- as\_json() (github3.users.Collaborator method), 224
- as\_json() (github3.users.Contributor method), 229
- as\_json() (github3.users.Email method), 237
- as\_json() (github3.users.Key method), 234
- as\_json() (github3.users.Plan method), 235
- as\_json() (github3.users.ShortUser method), 210
- as\_json() (github3.users.User method), 215
- Asset (class in github3.repos.release), 196
- asset() (github3.repos.release.Release method), 198
- asset() (github3.repos.repo.Repository method), 139
- asset() (github3.repos.repo.ShortRepository method), 165
- assets() (github3.repos.release.Release method), 198
- assets\_url (github3.repos.release.Release attribute), 197
- assign() (github3.issues.issue.Issue method), 81
- assign() (github3.issues.issue.ShortIssue method), 77
- assignee (github3.issues.issue.ShortIssue attribute), 75
- assignee (github3.pulls.ShortPullRequest attribute), 120
- assignees (github3.issues.issue.ShortIssue attribute), 75
- assignees (github3.pulls.ShortPullRequest attribute), 120
- assignees() (github3.repos.repo.Repository method), 139
- assignees() (github3.repos.repo.ShortRepository method), 165
- assignees\_url (github3.repos.repo.ShortRepository attribute), 161
- AuthenticatedUser (class in github3.users), 219
- author (github3.git.ShortCommit attribute), 50
- author (github3.repos.release.Release attribute), 197
- author (github3.repos.stats.ContributorStats attribute), 204
- author\_association (github3.events.EventIssueComment attribute), 46
- author\_association (github3.events.EventReviewComment attribute), 45
- author\_association (github3.issues.comment.IssueComment attribute), 84
- author\_association (github3.pulls.ReviewComment attribute), 134
- author\_association (github3.repos.comment.ShortComment attribute), 200
- Authorization (class in github3.auths), 39
- authorization() (github3.github.GitHub method), 54
- authorizations() (github3.github.GitHub method), 54
- authorize() (github3.github.GitHub method), 55
- authorize() (in module github3), 30
- avatar\_url (github3.events.EventOrganization attribute), 44
- avatar\_url (github3.events.EventUser attribute), 44
- avatar\_url (github3.orgs.ShortOrganization attribute), 106
- avatar\_url (github3.users.ShortUser attribute), 209

**B**

Base (class in github3.pulls), 132  
 base (github3.pulls.ShortPullRequest attribute), 120  
 base\_commit (github3.repos.comparison.Comparison attribute), 195  
 behind\_by (github3.repos.comparison.Comparison attribute), 195  
 bio (github3.users.User attribute), 214  
 Blob (class in github3.git), 49  
 blob() (github3.repos.repo.Repository method), 139  
 blob() (github3.repos.repo.ShortRepository method), 165  
 blob\_url (github3.pulls.PullFile attribute), 132  
 blobs\_url (github3.repos.repo.ShortRepository attribute), 161  
 blog (github3.orgs.Organization attribute), 113  
 blog (github3.users.User attribute), 214  
 body (github3.events.EventIssueComment attribute), 46  
 body (github3.events.EventReviewComment attribute), 45  
 body (github3.issues.comment.IssueComment attribute), 84  
 body (github3.issues.issue.ShortIssue attribute), 75  
 body (github3.pulls.ReviewComment attribute), 134  
 body (github3.pulls.ShortPullRequest attribute), 120  
 body (github3.repos.comment.ShortComment attribute), 200  
 body (github3.repos.release.Release attribute), 197  
 body\_html (github3.issues.comment.IssueComment attribute), 84  
 body\_html (github3.issues.issue.Issue attribute), 80  
 body\_html (github3.pulls.ReviewComment attribute), 134  
 body\_html (github3.pulls.ShortPullRequest attribute), 120  
 body\_html (github3.repos.comment.RepoComment attribute), 201  
 body\_text (github3.issues.comment.IssueComment attribute), 84  
 body\_text (github3.issues.issue.Issue attribute), 80  
 body\_text (github3.pulls.ReviewComment attribute), 134  
 body\_text (github3.pulls.ShortPullRequest attribute), 120  
 body\_text (github3.repos.comment.RepoComment attribute), 201  
 Branch (class in github3.repos.branch), 192  
 branch() (github3.repos.repo.Repository method), 140  
 branch() (github3.repos.repo.ShortRepository method), 165  
 branches() (github3.repos.repo.Repository method), 140  
 branches() (github3.repos.repo.ShortRepository method), 165  
 branches\_url (github3.repos.repo.ShortRepository attribute), 161  
 BranchProtection (class in github3.repos.branch), 192

browser\_download\_url (github3.repos.release.Asset attribute), 196

**C**

change\_status (github3.gists.history.GistHistory attribute), 49  
 changes\_count (github3.pulls.PullFile attribute), 132  
 check\_authorization() (github3.github.GitHub method), 55  
 clone\_url (github3.repos.repo.Repository attribute), 137  
 close() (github3.issues.issue.Issue method), 81  
 close() (github3.issues.issue.ShortIssue method), 77  
 close() (github3.pulls.PullRequest method), 127  
 close() (github3.pulls.ShortPullRequest method), 122  
 closed\_at (github3.issues.issue.ShortIssue attribute), 75  
 closed\_at (github3.pulls.ShortPullRequest attribute), 120  
 closed\_by (github3.issues.issue.Issue attribute), 80  
 closed\_issues\_count (github3.issues.milestone.Milestone attribute), 89  
 cls (github3.structs.GitHubIterator attribute), 206  
 cname (github3.repos.pages.PagesInfo attribute), 199  
 code\_frequency() (github3.repos.repo.Repository method), 140  
 code\_frequency() (github3.repos.repo.ShortRepository method), 166  
 CodeSearchResult (class in github3.search), 204  
 Collaborator (class in github3.users), 224  
 collaborators() (github3.repos.repo.Repository method), 140  
 collaborators() (github3.repos.repo.ShortRepository method), 166  
 collaborators\_count (github3.users.Plan attribute), 235  
 collaborators\_url (github3.repos.repo.ShortRepository attribute), 162  
 CombinedStatus (class in github3.repos.status), 203  
 comment() (github3.issues.issue.Issue method), 81  
 comment() (github3.issues.issue.ShortIssue method), 77  
 comments() (github3.issues.issue.Issue method), 81  
 comments() (github3.issues.issue.ShortIssue method), 77  
 comments() (github3.repos.repo.Repository method), 141  
 comments() (github3.repos.repo.ShortRepository method), 166  
 comments\_count (github3.gists.gist.ShortGist attribute), 47  
 comments\_count (github3.issues.issue.ShortIssue attribute), 75  
 comments\_count (github3.pulls.PullRequest attribute), 126  
 comments\_url (github3.gists.gist.ShortGist attribute), 47  
 comments\_url (github3.issues.issue.ShortIssue attribute), 75  
 comments\_url (github3.pulls.ShortPullRequest attribute), 120

- comments\_url (github3.repos.repo.ShortRepository attribute), 162
- Commit (class in github3.git), 50
- commit (github3.repos.branch.ShortBranch attribute), 192
- commit (github3.repos.pages.PagesBuild attribute), 200
- commit (github3.repos.tag.RepoTag attribute), 191
- commit() (github3.repos.repo.Repository method), 141
- commit() (github3.repos.repo.ShortRepository method), 167
- commit\_activity() (github3.repos.repo.Repository method), 141
- commit\_activity() (github3.repos.repo.ShortRepository method), 167
- commit\_comment() (github3.repos.repo.Repository method), 141
- commit\_comment() (github3.repos.repo.ShortRepository method), 167
- commit\_id (github3.events.EventReviewComment attribute), 45
- commit\_id (github3.issues.event.IssueEvent attribute), 86
- commit\_id (github3.pulls.ReviewComment attribute), 134
- commit\_id (github3.repos.comment.ShortComment attribute), 200
- commit\_url (github3.issues.event.IssueEvent attribute), 86
- commit\_url (github3.repos.status.CombinedStatus attribute), 203
- commits (github3.repos.comparison.Comparison attribute), 195
- commits() (github3.pulls.PullRequest method), 127
- commits() (github3.pulls.ShortPullRequest method), 122
- commits() (github3.repos.repo.Repository method), 141
- commits() (github3.repos.repo.ShortRepository method), 167
- commits\_count (github3.pulls.PullRequest attribute), 126
- commits\_url (github3.gists.gist.Gist attribute), 48
- commits\_url (github3.pulls.ShortPullRequest attribute), 121
- commits\_url (github3.repos.repo.ShortRepository attribute), 162
- committed\_at (github3.gists.history.GistHistory attribute), 49
- committer (github3.git.ShortCommit attribute), 50
- CommitTree (class in github3.git), 51
- company (github3.orgs.Organization attribute), 113
- company (github3.users.User attribute), 214
- compare\_commits() (github3.repos.repo.Repository method), 142
- compare\_commits() (github3.repos.repo.ShortRepository method), 168
- compare\_url (github3.repos.repo.ShortRepository attribute), 162
- Comparison (class in github3.repos.comparison), 195
- conceal\_member() (github3.orgs.Organization method), 115
- conceal\_member() (github3.orgs.ShortOrganization method), 108
- config (github3.repos.hook.Hook attribute), 190
- content (github3.git.Blob attribute), 50
- content (github3.repos.contents.Contents attribute), 188
- content\_type (github3.repos.release.Asset attribute), 196
- Contents (class in github3.repos.contents), 188
- contents() (github3.pulls.PullFile method), 133
- contents\_url (github3.pulls.PullFile attribute), 132
- contents\_url (github3.repos.repo.ShortRepository attribute), 162
- context (github3.repos.status.ShortStatus attribute), 202
- contexts() (github3.repos.branch.ProtectionRequiredStatusChecks method), 194
- contributions\_count (github3.users.Contributor attribute), 229
- Contributor (class in github3.users), 229
- contributor\_statistics() (github3.repos.repo.Repository method), 142
- contributor\_statistics() (github3.repos.repo.ShortRepository method), 168
- contributors() (github3.repos.repo.Repository method), 142
- contributors() (github3.repos.repo.ShortRepository method), 168
- contributors\_url (github3.repos.repo.ShortRepository attribute), 162
- ContributorStats (class in github3.repos.stats), 204
- count (github3.structs.GitHubIterator attribute), 206
- create\_blob() (github3.repos.repo.Repository method), 143
- create\_blob() (github3.repos.repo.ShortRepository method), 168
- create\_branch\_ref() (github3.repos.repo.Repository method), 143
- create\_branch\_ref() (github3.repos.repo.ShortRepository method), 169
- create\_comment() (github3.issues.issue.Issue method), 81
- create\_comment() (github3.issues.issue.ShortIssue method), 77
- create\_comment() (github3.pulls.PullRequest method), 127
- create\_comment() (github3.pulls.ShortPullRequest method), 122
- create\_comment() (github3.repos.repo.Repository method), 143
- create\_comment() (github3.repos.repo.ShortRepository method), 169
- create\_commit() (github3.repos.repo.Repository method), 143

- [create\\_commit\(\)](#) (github3.repos.repo.ShortRepository method), 169  
[create\\_deployment\(\)](#) (github3.repos.repo.Repository method), 144  
[create\\_deployment\(\)](#) (github3.repos.repo.ShortRepository method), 170  
[create\\_file\(\)](#) (github3.repos.repo.Repository method), 144  
[create\\_file\(\)](#) (github3.repos.repo.ShortRepository method), 170  
[create\\_fork\(\)](#) (github3.repos.repo.Repository method), 145  
[create\\_fork\(\)](#) (github3.repos.repo.ShortRepository method), 170  
[create\\_gist\(\)](#) (github3.github.GitHub method), 55  
[create\\_gist\(\)](#) (in module github3), 30  
[create\\_hook\(\)](#) (github3.repos.repo.Repository method), 145  
[create\\_hook\(\)](#) (github3.repos.repo.ShortRepository method), 170  
[create\\_issue\(\)](#) (github3.github.GitHub method), 55  
[create\\_issue\(\)](#) (github3.repos.repo.Repository method), 145  
[create\\_issue\(\)](#) (github3.repos.repo.ShortRepository method), 171  
[create\\_key\(\)](#) (github3.github.GitHub method), 56  
[create\\_key\(\)](#) (github3.repos.repo.Repository method), 145  
[create\\_key\(\)](#) (github3.repos.repo.ShortRepository method), 171  
[create\\_label\(\)](#) (github3.repos.repo.Repository method), 145  
[create\\_label\(\)](#) (github3.repos.repo.ShortRepository method), 171  
[create\\_milestone\(\)](#) (github3.repos.repo.Repository method), 146  
[create\\_milestone\(\)](#) (github3.repos.repo.ShortRepository method), 171  
[create\\_project\(\)](#) (github3.orgs.Organization method), 115  
[create\\_project\(\)](#) (github3.orgs.ShortOrganization method), 108  
[create\\_project\(\)](#) (github3.repos.repo.Repository method), 146  
[create\\_project\(\)](#) (github3.repos.repo.ShortRepository method), 172  
[create\\_pull\(\)](#) (github3.repos.repo.Repository method), 146  
[create\\_pull\(\)](#) (github3.repos.repo.ShortRepository method), 172  
[create\\_pull\\_from\\_issue\(\)](#) (github3.repos.repo.Repository method), 146  
[create\\_pull\\_from\\_issue\(\)](#) (github3.repos.repo.ShortRepository method), 172  
[create\\_ref\(\)](#) (github3.repos.repo.Repository method), 147  
[create\\_ref\(\)](#) (github3.repos.repo.ShortRepository method), 172  
[create\\_release\(\)](#) (github3.repos.repo.Repository method), 147  
[create\\_release\(\)](#) (github3.repos.repo.ShortRepository method), 173  
[create\\_repository\(\)](#) (github3.github.GitHub method), 56  
[create\\_repository\(\)](#) (github3.orgs.Organization method), 115  
[create\\_repository\(\)](#) (github3.orgs.ShortOrganization method), 108  
[create\\_review\(\)](#) (github3.pulls.PullRequest method), 127  
[create\\_review\(\)](#) (github3.pulls.ShortPullRequest method), 122  
[create\\_review\\_comment\(\)](#) (github3.pulls.PullRequest method), 128  
[create\\_review\\_comment\(\)](#) (github3.pulls.ShortPullRequest method), 123  
[create\\_status\(\)](#) (github3.repos.deployment.Deployment method), 201  
[create\\_status\(\)](#) (github3.repos.repo.Repository method), 147  
[create\\_status\(\)](#) (github3.repos.repo.ShortRepository method), 173  
[create\\_tag\(\)](#) (github3.repos.repo.Repository method), 147  
[create\\_tag\(\)](#) (github3.repos.repo.ShortRepository method), 173  
[create\\_team\(\)](#) (github3.orgs.Organization method), 116  
[create\\_team\(\)](#) (github3.orgs.ShortOrganization method), 109  
[create\\_tree\(\)](#) (github3.repos.repo.Repository method), 148  
[create\\_tree\(\)](#) (github3.repos.repo.ShortRepository method), 174  
[create\\_user\(\)](#) (github3.github.GitHubEnterprise method), 74  
[created\\_at](#) (github3.auths.Authorization attribute), 39  
[created\\_at](#) (github3.events.Event attribute), 42  
[created\\_at](#) (github3.events.EventIssueComment attribute), 46  
[created\\_at](#) (github3.events.EventReviewComment attribute), 45  
[created\\_at](#) (github3.gists.gist.GistFork attribute), 47  
[created\\_at](#) (github3.gists.gist.ShortGist attribute), 47  
[created\\_at](#) (github3.issues.comment.IssueComment attribute), 84  
[created\\_at](#) (github3.issues.event.IssueEvent attribute), 86  
[created\\_at](#) (github3.issues.issue.ShortIssue attribute), 75  
[created\\_at](#) (github3.issues.milestone.Milestone attribute), 89  
[created\\_at](#) (github3.notifications.RepositorySubscription attribute), 96  
[created\\_at](#) (github3.notifications.ThreadSubscription attribute), 96



- tribute), 94
  - created\_at (github3.orgs.Organization attribute), 113
  - created\_at (github3.orgs.Team attribute), 101
  - created\_at (github3.pulls.ReviewComment attribute), 134
  - created\_at (github3.pulls.ShortPullRequest attribute), 121
  - created\_at (github3.repos.comment.ShortComment attribute), 200
  - created\_at (github3.repos.deployment.Deployment attribute), 201
  - created\_at (github3.repos.deployment.DeploymentStatus attribute), 202
  - created\_at (github3.repos.hook.Hook attribute), 190
  - created\_at (github3.repos.issue\_import.ImportedIssue attribute), 191
  - created\_at (github3.repos.pages.PagesBuild attribute), 200
  - created\_at (github3.repos.release.Asset attribute), 196
  - created\_at (github3.repos.release.Release attribute), 197
  - created\_at (github3.repos.repo.Repository attribute), 137
  - created\_at (github3.repos.status.ShortStatus attribute), 203
  - created\_at (github3.users.User attribute), 214
  - creator (github3.issues.milestone.Milestone attribute), 89
  - creator (github3.repos.deployment.Deployment attribute), 201
  - creator (github3.repos.deployment.DeploymentStatus attribute), 202
  - creator (github3.repos.status.ShortStatus attribute), 203
  - creator (github3.repos.status.Status attribute), 203
  - custom\_404 (github3.repos.pages.PagesInfo attribute), 199
- ## D
- decoded (github3.repos.contents.Contents attribute), 188
  - default\_branch (github3.repos.repo.Repository attribute), 137
  - delete() (github3.auths.Authorization method), 40
  - delete() (github3.issues.comment.IssueComment method), 85
  - delete() (github3.issues.label.Label method), 88
  - delete() (github3.issues.milestone.Milestone method), 90
  - delete() (github3.notifications.RepositorySubscription method), 96
  - delete() (github3.notifications.ThreadSubscription method), 94
  - delete() (github3.orgs.ShortTeam method), 98
  - delete() (github3.orgs.Team method), 102
  - delete() (github3.pulls.ReviewComment method), 135
  - delete() (github3.repos.branch.ProtectionRequiredStatusChecks method), 194
  - delete() (github3.repos.contents.Contents method), 189
  - delete() (github3.repos.hook.Hook method), 190
  - delete() (github3.repos.release.Asset method), 197
  - delete() (github3.repos.release.Release method), 198
  - delete() (github3.repos.repo.Repository method), 148
  - delete() (github3.repos.repo.ShortRepository method), 174
  - delete() (github3.users.AuthenticatedUser method), 220
  - delete() (github3.users.Collaborator method), 225
  - delete() (github3.users.Contributor method), 229
  - delete() (github3.users.Key method), 234
  - delete() (github3.users.ShortUser method), 210
  - delete() (github3.users.User method), 215
  - delete\_email\_addresses() (github3.github.GitHub method), 57
  - delete\_key() (github3.repos.repo.Repository method), 148
  - delete\_key() (github3.repos.repo.ShortRepository method), 174
  - delete\_subscription() (github3.notifications.Thread method), 93
  - delete\_subscription() (github3.repos.repo.Repository method), 148
  - delete\_subscription() (github3.repos.repo.ShortRepository method), 174
  - deletions (github3.gists.history.GistHistory attribute), 49
  - deletions\_count (github3.pulls.PullFile attribute), 132
  - deletions\_count (github3.pulls.PullRequest attribute), 126
  - demote() (github3.users.AuthenticatedUser method), 220
  - demote() (github3.users.Collaborator method), 225
  - demote() (github3.users.Contributor method), 229
  - demote() (github3.users.ShortUser method), 210
  - demote() (github3.users.User method), 215
  - Deployment (class in github3.repos.deployment), 201
  - deployment() (github3.repos.repo.Repository method), 148
  - deployment() (github3.repos.repo.ShortRepository method), 174
  - deployment\_url (github3.repos.deployment.DeploymentStatus attribute), 202
  - deployments() (github3.repos.repo.Repository method), 149
  - deployments() (github3.repos.repo.ShortRepository method), 174
  - deployments\_url (github3.repos.repo.ShortRepository attribute), 162
  - DeploymentStatus (class in github3.repos.deployment), 202
  - description (github3.gists.gist.ShortGist attribute), 47
  - description (github3.issues.milestone.Milestone attribute), 90
  - description (github3.orgs.ShortOrganization attribute), 106
  - description (github3.repos.deployment.Deployment attribute), 201
  - description (github3.repos.deployment.DeploymentStatus attribute), 202
  - description (github3.repos.repo.ShortRepository at-

- tribute), 162
  - description (github3.repos.status.ShortStatus attribute), 203
  - diff() (github3.pulls.PullRequest method), 128
  - diff() (github3.pulls.ShortPullRequest method), 123
  - diff() (github3.repos.comparison.Comparison method), 196
  - diff\_hunk (github3.events.EventReviewComment attribute), 45
  - diff\_hunk (github3.pulls.ReviewComment attribute), 134
  - diff\_url (github3.pulls.ShortPullRequest attribute), 121
  - diff\_url (github3.repos.comparison.Comparison attribute), 195
  - directory\_contents() (github3.repos.repo.Repository method), 149
  - directory\_contents() (github3.repos.repo.ShortRepository method), 175
  - disk\_usage (github3.users.AuthenticatedUser attribute), 219
  - display\_login (github3.events.EventUser attribute), 44
  - download() (github3.repos.release.Asset method), 197
  - download\_count (github3.repos.release.Asset attribute), 196
  - download\_url (github3.repos.release.Asset attribute), 196
  - downloads\_url (github3.repos.repo.ShortRepository attribute), 162
  - draft (github3.repos.release.Release attribute), 197
  - due\_on (github3.issues.milestone.Milestone attribute), 90
  - duration (github3.repos.pages.PagesBuild attribute), 200
- ## E
- edit() (github3.issues.comment.IssueComment method), 85
  - edit() (github3.issues.issue.Issue method), 81
  - edit() (github3.issues.issue.ShortIssue method), 77
  - edit() (github3.orgs.Membership method), 105
  - edit() (github3.orgs.Organization method), 116
  - edit() (github3.orgs.ShortOrganization method), 109
  - edit() (github3.orgs.ShortTeam method), 98
  - edit() (github3.orgs.Team method), 102
  - edit() (github3.pulls.ReviewComment method), 135
  - edit() (github3.repos.hook.Hook method), 190
  - edit() (github3.repos.release.Asset method), 197
  - edit() (github3.repos.release.Release method), 199
  - edit() (github3.repos.repo.Repository method), 149
  - edit() (github3.repos.repo.ShortRepository method), 175
  - Email (class in github3.users), 236
  - email (github3.orgs.Organization attribute), 113
  - email (github3.users.Email attribute), 236
  - email (github3.users.User attribute), 214
  - emails() (github3.github.GitHub method), 57
  - emojis() (github3.github.GitHub method), 57
  - enabled (github3.repos.branch.ProtectionEnforceAdmins attribute), 193
  - encoding (github3.git.Blob attribute), 50
  - encoding (github3.repos.contents.Contents attribute), 188
  - enforce\_admins (github3.repos.branch.BranchProtection attribute), 192
  - environment (github3.repos.deployment.Deployment attribute), 201
  - error (github3.repos.pages.PagesBuild attribute), 200
  - etag (github3.structs.GitHubIterator attribute), 206
  - Event (class in github3.events), 42
  - event (github3.issues.event.IssueEvent attribute), 86
  - EventIssue (class in github3.events), 46
  - EventIssueComment (class in github3.events), 46
  - EventOrganization (class in github3.events), 44
  - EventPullRequest (class in github3.events), 45
  - EventReviewComment (class in github3.events), 45
  - events (github3.repos.hook.Hook attribute), 190
  - events() (github3.issues.issue.Issue method), 82
  - events() (github3.issues.issue.ShortIssue method), 78
  - events() (github3.orgs.Organization method), 117
  - events() (github3.orgs.ShortOrganization method), 110
  - events() (github3.repos.repo.Repository method), 150
  - events() (github3.repos.repo.ShortRepository method), 176
  - events() (github3.users.AuthenticatedUser method), 220
  - events() (github3.users.Collaborator method), 225
  - events() (github3.users.Contributor method), 230
  - events() (github3.users.ShortUser method), 210
  - events() (github3.users.User method), 215
  - events\_url (github3.issues.issue.ShortIssue attribute), 75
  - events\_url (github3.orgs.ShortOrganization attribute), 106
  - events\_url (github3.repos.repo.ShortRepository attribute), 162
  - events\_urlt (github3.users.ShortUser attribute), 209
  - EventUser (class in github3.events), 44
- ## F
- feeds() (github3.github.GitHub method), 57
  - file\_contents() (github3.repos.repo.Repository method), 150
  - file\_contents() (github3.repos.repo.ShortRepository method), 176
  - filename (github3.gists.file.ShortGistFile attribute), 48
  - filename (github3.pulls.PullFile attribute), 132
  - files (github3.gists.gist.ShortGist attribute), 47
  - files (github3.repos.comparison.Comparison attribute), 195
  - files() (github3.pulls.PullRequest method), 128
  - files() (github3.pulls.ShortPullRequest method), 123
  - fingerprint (github3.auths.Authorization attribute), 39
  - follow() (github3.github.GitHub method), 57
  - followed\_by() (github3.github.GitHub method), 57
  - followed\_by() (in module github3), 32
  - followers() (github3.github.GitHub method), 58

[followers\(\)](#) (github3.users.AuthenticatedUser method), 221  
[followers\(\)](#) (github3.users.Collaborator method), 225  
[followers\(\)](#) (github3.users.Contributor method), 230  
[followers\(\)](#) (github3.users.ShortUser method), 210  
[followers\(\)](#) (github3.users.User method), 216  
[followers\\_count](#) (github3.orgs.Organization attribute), 113  
[followers\\_count](#) (github3.users.User attribute), 214  
[followers\\_of\(\)](#) (github3.github.GitHub method), 58  
[followers\\_of\(\)](#) (in module github3), 32  
[followers\\_url](#) (github3.users.ShortUser attribute), 209  
[following\(\)](#) (github3.github.GitHub method), 58  
[following\(\)](#) (github3.users.AuthenticatedUser method), 221  
[following\(\)](#) (github3.users.Collaborator method), 225  
[following\(\)](#) (github3.users.Contributor method), 230  
[following\(\)](#) (github3.users.ShortUser method), 211  
[following\(\)](#) (github3.users.User method), 216  
[following\\_count](#) (github3.orgs.Organization attribute), 113  
[following\\_count](#) (github3.users.User attribute), 214  
[following\\_url](#) (github3.users.ShortUser attribute), 209  
[fork](#) (github3.repos.repo.ShortRepository attribute), 162  
[forks\(\)](#) (github3.repos.repo.Repository method), 151  
[forks\(\)](#) (github3.repos.repo.ShortRepository method), 176  
[forks\\_count](#) (github3.repos.repo.Repository attribute), 137  
[forks\\_url](#) (github3.gists.gist.Gist attribute), 48  
[forks\\_url](#) (github3.repos.repo.ShortRepository attribute), 162  
[from\\_dict\(\)](#) (github3.auths.Authorization class method), 40  
[from\\_dict\(\)](#) (github3.events.Event class method), 43  
[from\\_dict\(\)](#) (github3.github.GitHub class method), 58  
[from\\_dict\(\)](#) (github3.issues.comment.IssueComment class method), 85  
[from\\_dict\(\)](#) (github3.issues.event.IssueEvent class method), 87  
[from\\_dict\(\)](#) (github3.issues.issue.Issue class method), 82  
[from\\_dict\(\)](#) (github3.issues.issue.ShortIssue class method), 78  
[from\\_dict\(\)](#) (github3.issues.label.Label class method), 88  
[from\\_dict\(\)](#) (github3.issues.milestone.Milestone class method), 90  
[from\\_dict\(\)](#) (github3.models.GitHubCore class method), 238  
[from\\_dict\(\)](#) (github3.notifications.RepositorySubscription class method), 96  
[from\\_dict\(\)](#) (github3.notifications.Thread class method), 93  
[from\\_dict\(\)](#) (github3.notifications.ThreadSubscription class method), 95  
[from\\_dict\(\)](#) (github3.orgs.Membership class method), 105  
[from\\_dict\(\)](#) (github3.orgs.Organization class method), 117  
[from\\_dict\(\)](#) (github3.orgs.ShortOrganization class method), 110  
[from\\_dict\(\)](#) (github3.orgs.ShortTeam class method), 99  
[from\\_dict\(\)](#) (github3.orgs.Team class method), 102  
[from\\_dict\(\)](#) (github3.pulls.PullDestination class method), 131  
[from\\_dict\(\)](#) (github3.pulls.PullFile class method), 133  
[from\\_dict\(\)](#) (github3.pulls.PullRequest class method), 128  
[from\\_dict\(\)](#) (github3.pulls.ReviewComment class method), 135  
[from\\_dict\(\)](#) (github3.pulls.ShortPullRequest class method), 123  
[from\\_dict\(\)](#) (github3.repos.repo.Repository class method), 151  
[from\\_dict\(\)](#) (github3.repos.repo.ShortRepository class method), 177  
[from\\_dict\(\)](#) (github3.repos.repo.StarredRepository class method), 187  
[from\\_dict\(\)](#) (github3.structs.GitHubIterator class method), 206  
[from\\_dict\(\)](#) (github3.structs.SearchIterator class method), 208  
[from\\_dict\(\)](#) (github3.users.AuthenticatedUser class method), 221  
[from\\_dict\(\)](#) (github3.users.Collaborator class method), 225  
[from\\_dict\(\)](#) (github3.users.Contributor class method), 230  
[from\\_dict\(\)](#) (github3.users.Email class method), 237  
[from\\_dict\(\)](#) (github3.users.Key class method), 234  
[from\\_dict\(\)](#) (github3.users.Plan class method), 235  
[from\\_dict\(\)](#) (github3.users.ShortUser class method), 211  
[from\\_dict\(\)](#) (github3.users.User class method), 216  
[from\\_json\(\)](#) (github3.auths.Authorization class method), 40  
[from\\_json\(\)](#) (github3.events.Event class method), 43  
[from\\_json\(\)](#) (github3.github.GitHub class method), 59  
[from\\_json\(\)](#) (github3.issues.comment.IssueComment class method), 85  
[from\\_json\(\)](#) (github3.issues.event.IssueEvent class method), 87  
[from\\_json\(\)](#) (github3.issues.issue.Issue class method), 82  
[from\\_json\(\)](#) (github3.issues.issue.ShortIssue class method), 78  
[from\\_json\(\)](#) (github3.issues.label.Label class method), 89  
[from\\_json\(\)](#) (github3.issues.milestone.Milestone class method), 90  
[from\\_json\(\)](#) (github3.models.GitHubCore class method), 239



- [from\\_json\(\) \(github3.notifications.RepositorySubscription class method\), 96](#)  
[from\\_json\(\) \(github3.notifications.Thread class method\), 93](#)  
[from\\_json\(\) \(github3.notifications.ThreadSubscription class method\), 95](#)  
[from\\_json\(\) \(github3.orgs.Membership class method\), 105](#)  
[from\\_json\(\) \(github3.orgs.Organization class method\), 117](#)  
[from\\_json\(\) \(github3.orgs.ShortOrganization class method\), 110](#)  
[from\\_json\(\) \(github3.orgs.ShortTeam class method\), 99](#)  
[from\\_json\(\) \(github3.orgs.Team class method\), 102](#)  
[from\\_json\(\) \(github3.pulls.PullDestination class method\), 131](#)  
[from\\_json\(\) \(github3.pulls.PullFile class method\), 133](#)  
[from\\_json\(\) \(github3.pulls.PullRequest class method\), 128](#)  
[from\\_json\(\) \(github3.pulls.ReviewComment class method\), 135](#)  
[from\\_json\(\) \(github3.pulls.ShortPullRequest class method\), 123](#)  
[from\\_json\(\) \(github3.repos.repo.Repository class method\), 151](#)  
[from\\_json\(\) \(github3.repos.repo.ShortRepository class method\), 177](#)  
[from\\_json\(\) \(github3.repos.repo.StarredRepository class method\), 187](#)  
[from\\_json\(\) \(github3.structs.GitHubIterator class method\), 206](#)  
[from\\_json\(\) \(github3.structs.SearchIterator class method\), 208](#)  
[from\\_json\(\) \(github3.users.AuthenticatedUser class method\), 221](#)  
[from\\_json\(\) \(github3.users.Collaborator class method\), 225](#)  
[from\\_json\(\) \(github3.users.Contributor class method\), 230](#)  
[from\\_json\(\) \(github3.users.Email class method\), 237](#)  
[from\\_json\(\) \(github3.users.Key class method\), 234](#)  
[from\\_json\(\) \(github3.users.Plan class method\), 236](#)  
[from\\_json\(\) \(github3.users.ShortUser class method\), 211](#)  
[from\\_json\(\) \(github3.users.User class method\), 216](#)  
[full\\_name \(github3.repos.repo.ShortRepository attribute\), 162](#)
- ## G
- [Gist \(class in github3.gists.gist\), 48](#)  
[gist\(\) \(github3.github.GitHub method\), 59](#)  
[gist\(\) \(in module github3\), 30](#)  
[GistFile \(class in github3.gists.file\), 48](#)  
[GistFork \(class in github3.gists.gist\), 47](#)  
[GistHistory \(class in github3.gists.history\), 49](#)  
[gists\(\) \(github3.github.GitHub method\), 59](#)  
[gists\\_by\(\) \(github3.github.GitHub method\), 59](#)  
[gists\\_by\(\) \(in module github3\), 32](#)  
[gists\\_urlt \(github3.users.ShortUser attribute\), 209](#)  
[git\\_commit\(\) \(github3.repos.repo.Repository method\), 151](#)  
[git\\_commit\(\) \(github3.repos.repo.ShortRepository method\), 177](#)  
[git\\_commits\\_urlt \(github3.repos.repo.ShortRepository attribute\), 162](#)  
[git\\_pull\\_url \(github3.gists.gist.ShortGist attribute\), 47](#)  
[git\\_push\\_url \(github3.gists.gist.ShortGist attribute\), 47](#)  
[git\\_refs\\_urlt \(github3.repos.repo.ShortRepository attribute\), 162](#)  
[git\\_tags\\_urlt \(github3.repos.repo.ShortRepository attribute\), 162](#)  
[git\\_url \(github3.repos.contents.Contents attribute\), 188](#)  
[git\\_url \(github3.repos.repo.Repository attribute\), 137](#)  
[git\\_url \(github3.search.CodeSearchResult attribute\), 204](#)  
[GitHub \(class in github3.github\), 52](#)  
[github3 \(module\), 29](#)  
[github3.api \(module\), 29](#)  
[GitHubCore \(class in github3.models\), 238](#)  
[GitHubEnterprise \(class in github3.github\), 74](#)  
[GitHubIterator \(class in github3.structs\), 206](#)  
[GitHubStatus \(class in github3.github\), 74](#)  
[gitignore\\_template\(\) \(github3.github.GitHub method\), 59](#)  
[gitignore\\_template\(\) \(in module github3\), 31](#)  
[gitignore\\_templates\(\) \(github3.github.GitHub method\), 59](#)  
[gitignore\\_templates\(\) \(in module github3\), 31](#)  
[GitObject \(class in github3.git\), 51](#)  
[gravatar\\_id \(github3.events.EventOrganization attribute\), 44](#)  
[gravatar\\_id \(github3.events.EventUser attribute\), 44](#)  
[gravatar\\_id \(github3.users.ShortUser attribute\), 209](#)
- ## H
- [has\\_downloads \(github3.repos.repo.Repository attribute\), 137](#)  
[has\\_issues \(github3.repos.repo.Repository attribute\), 137](#)  
[has\\_pages \(github3.repos.repo.Repository attribute\), 137](#)  
[has\\_repository\(\) \(github3.orgs.ShortTeam method\), 99](#)  
[has\\_repository\(\) \(github3.orgs.Team method\), 102](#)  
[has\\_wiki \(github3.repos.repo.Repository attribute\), 137](#)  
[Hash \(class in github3.git\), 51](#)  
[hashed\\_token \(github3.auths.Authorization attribute\), 39](#)  
[Head \(class in github3.pulls\), 132](#)  
[head \(github3.pulls.ShortPullRequest attribute\), 121](#)  
[headers \(github3.structs.GitHubIterator attribute\), 206](#)  
[hireable \(github3.users.User attribute\), 214](#)  
[history \(github3.gists.gist.Gist attribute\), 48](#)  
[homepage \(github3.repos.repo.Repository attribute\), 137](#)  
[Hook \(class in github3.repos.hook\), 190](#)

- hook() (github3.repos.repo.Repository method), 151
- hook() (github3.repos.repo.ShortRepository method), 177
- hooks() (github3.repos.repo.Repository method), 151
- hooks() (github3.repos.repo.ShortRepository method), 177
- hooks\_url (github3.orgs.ShortOrganization attribute), 106
- hooks\_url (github3.repos.repo.ShortRepository attribute), 163
- html\_url (github3.events.EventIssueComment attribute), 46
- html\_url (github3.events.EventReviewComment attribute), 45
- html\_url (github3.gists.gist.ShortGist attribute), 47
- html\_url (github3.issues.comment.IssueComment attribute), 84
- html\_url (github3.issues.issue.ShortIssue attribute), 75
- html\_url (github3.orgs.Organization attribute), 113
- html\_url (github3.pulls.ReviewComment attribute), 134
- html\_url (github3.pulls.ShortPullRequest attribute), 121
- html\_url (github3.repos.comment.ShortComment attribute), 200
- html\_url (github3.repos.comparison.Comparison attribute), 195
- html\_url (github3.repos.contents.Contents attribute), 188
- html\_url (github3.repos.release.Release attribute), 197
- html\_url (github3.repos.repo.ShortRepository attribute), 163
- html\_url (github3.search.CodeSearchResult attribute), 204
- html\_url (github3.users.ShortUser attribute), 209
- |
- id (github3.auths.Authorization attribute), 39
- id (github3.events.Event attribute), 42
- id (github3.events.EventIssueComment attribute), 46
- id (github3.events.EventOrganization attribute), 44
- id (github3.events.EventPullRequest attribute), 45
- id (github3.events.EventReviewComment attribute), 45
- id (github3.events.EventUser attribute), 44
- id (github3.gists.gist.GistFork attribute), 47
- id (github3.gists.gist.ShortGist attribute), 47
- id (github3.issues.comment.IssueComment attribute), 84
- id (github3.issues.event.IssueEvent attribute), 86
- id (github3.issues.issue.ShortIssue attribute), 75
- id (github3.issues.milestone.Milestone attribute), 90
- id (github3.notifications.Thread attribute), 92
- id (github3.orgs.ShortOrganization attribute), 106
- id (github3.orgs.ShortTeam attribute), 97
- id (github3.pulls.ReviewComment attribute), 134
- id (github3.pulls.ShortPullRequest attribute), 121
- id (github3.repos.comment.ShortComment attribute), 200
- id (github3.repos.deployment.Deployment attribute), 201
- id (github3.repos.deployment.DeploymentStatus attribute), 202
- id (github3.repos.hook.Hook attribute), 190
- id (github3.repos.issue\_import.ImportedIssue attribute), 191
- id (github3.repos.release.Asset attribute), 196
- id (github3.repos.release.Release attribute), 197
- id (github3.repos.repo.ShortRepository attribute), 163
- id (github3.repos.status.ShortStatus attribute), 203
- id (github3.users.Key attribute), 233
- id (github3.users.ShortUser attribute), 209
- ignore() (github3.repos.repo.Repository method), 151
- ignore() (github3.repos.repo.ShortRepository method), 177
- ignored (github3.notifications.RepositorySubscription attribute), 96
- ignored (github3.notifications.ThreadSubscription attribute), 94
- impersonate() (github3.users.AuthenticatedUser method), 221
- impersonate() (github3.users.Collaborator method), 226
- impersonate() (github3.users.Contributor method), 230
- impersonate() (github3.users.ShortUser method), 211
- impersonate() (github3.users.User method), 216
- import\_issue() (github3.repos.repo.Repository method), 151
- import\_issue() (github3.repos.repo.ShortRepository method), 177
- import\_issues\_url (github3.repos.issue\_import.ImportedIssue attribute), 191
- imported\_issue() (github3.repos.repo.Repository method), 152
- imported\_issue() (github3.repos.repo.ShortRepository method), 178
- imported\_issues() (github3.repos.repo.Repository method), 152
- imported\_issues() (github3.repos.repo.ShortRepository method), 178
- ImportedIssue (class in github3.repos.issue\_import), 191
- invite() (github3.orgs.ShortTeam method), 99
- invite() (github3.orgs.Team method), 102
- is\_assignee() (github3.repos.repo.Repository method), 152
- is\_assignee() (github3.repos.repo.ShortRepository method), 178
- is\_assignee\_on() (github3.users.AuthenticatedUser method), 221
- is\_assignee\_on() (github3.users.Collaborator method), 226
- is\_assignee\_on() (github3.users.Contributor method), 230
- is\_assignee\_on() (github3.users.ShortUser method), 211
- is\_assignee\_on() (github3.users.User method), 216
- is\_closed() (github3.issues.issue.Issue method), 82
- is\_closed() (github3.issues.issue.ShortIssue method), 78
- is\_collaborator() (github3.repos.repo.Repository method), 152

- [is\\_collaborator\(\)](#) (github3.repos.repo.ShortRepository method), 178  
[is\\_following\(\)](#) (github3.github.GitHub method), 59  
[is\\_following\(\)](#) (github3.users.AuthenticatedUser method), 221  
[is\\_following\(\)](#) (github3.users.Collaborator method), 226  
[is\\_following\(\)](#) (github3.users.Contributor method), 231  
[is\\_following\(\)](#) (github3.users.ShortUser method), 211  
[is\\_following\(\)](#) (github3.users.User method), 216  
[is\\_free\(\)](#) (github3.users.Plan method), 236  
[is\\_member\(\)](#) (github3.orgs.Organization method), 117  
[is\\_member\(\)](#) (github3.orgs.ShortOrganization method), 110  
[is\\_member\(\)](#) (github3.orgs.ShortTeam method), 99  
[is\\_member\(\)](#) (github3.orgs.Team method), 102  
[is\\_merged\(\)](#) (github3.pulls.PullRequest method), 128  
[is\\_merged\(\)](#) (github3.pulls.ShortPullRequest method), 123  
[is\\_public\\_member\(\)](#) (github3.orgs.Organization method), 117  
[is\\_public\\_member\(\)](#) (github3.orgs.ShortOrganization method), 110  
[is\\_starred\(\)](#) (github3.github.GitHub method), 60  
[Issue](#) (class in github3.issues.issue), 80  
[issue](#) (github3.issues.event.RepositoryIssueEvent attribute), 88  
[issue](#) (github3.search.IssueSearchResult attribute), 205  
[issue\(\)](#) (github3.github.GitHub method), 60  
[issue\(\)](#) (github3.pulls.PullRequest method), 128  
[issue\(\)](#) (github3.pulls.ShortPullRequest method), 124  
[issue\(\)](#) (github3.repos.repo.Repository method), 153  
[issue\(\)](#) (github3.repos.repo.ShortRepository method), 178  
[issue\(\)](#) (in module github3), 31  
[issue\\_comment\\_urlt](#) (github3.repos.repo.ShortRepository attribute), 163  
[issue\\_comments\(\)](#) (github3.pulls.PullRequest method), 129  
[issue\\_comments\(\)](#) (github3.pulls.ShortPullRequest method), 124  
[issue\\_events\(\)](#) (github3.repos.repo.Repository method), 153  
[issue\\_events\(\)](#) (github3.repos.repo.ShortRepository method), 178  
[issue\\_events\\_urlt](#) (github3.repos.repo.ShortRepository attribute), 163  
[issue\\_url](#) (github3.events.EventIssueComment attribute), 46  
[issue\\_url](#) (github3.issues.comment.IssueComment attribute), 84  
[issue\\_url](#) (github3.pulls.ShortPullRequest attribute), 121  
[IssueComment](#) (class in github3.issues.comment), 84  
[IssueEvent](#) (class in github3.issues.event), 86  
[issues\(\)](#) (github3.github.GitHub method), 60  
[issues\(\)](#) (github3.repos.repo.Repository method), 153  
[issues\(\)](#) (github3.repos.repo.ShortRepository method), 179  
[issues\\_on\(\)](#) (github3.github.GitHub method), 60  
[issues\\_on\(\)](#) (in module github3), 31  
[issues\\_url](#) (github3.orgs.ShortOrganization attribute), 106  
[issues\\_urlt](#) (github3.repos.repo.ShortRepository attribute), 163  
[IssueSearchResult](#) (class in github3.search), 205  
[items](#) (github3.structs.SearchIterator attribute), 208
- ## K
- [Key](#) (class in github3.users), 233  
[key](#) (github3.users.Key attribute), 233  
[key\(\)](#) (github3.github.GitHub method), 61  
[key\(\)](#) (github3.repos.repo.Repository method), 153  
[key\(\)](#) (github3.repos.repo.ShortRepository method), 179  
[keys\(\)](#) (github3.github.GitHub method), 61  
[keys\(\)](#) (github3.repos.repo.Repository method), 154  
[keys\(\)](#) (github3.repos.repo.ShortRepository method), 179  
[keys\(\)](#) (github3.users.AuthenticatedUser method), 221  
[keys\(\)](#) (github3.users.Collaborator method), 226  
[keys\(\)](#) (github3.users.Contributor method), 231  
[keys\(\)](#) (github3.users.ShortUser method), 211  
[keys\(\)](#) (github3.users.User method), 217  
[keys\\_urlt](#) (github3.repos.repo.ShortRepository attribute), 163
- ## L
- [Label](#) (class in github3.issues.label), 88  
[label](#) (github3.pulls.PullDestination attribute), 131  
[label](#) (github3.repos.release.Asset attribute), 196  
[label\(\)](#) (github3.repos.repo.Repository method), 154  
[label\(\)](#) (github3.repos.repo.ShortRepository method), 179  
[labels\(\)](#) (github3.issues.issue.Issue method), 82  
[labels\(\)](#) (github3.issues.issue.ShortIssue method), 78  
[labels\(\)](#) (github3.issues.milestone.Milestone method), 90  
[labels\(\)](#) (github3.repos.repo.Repository method), 154  
[labels\(\)](#) (github3.repos.repo.ShortRepository method), 180  
[labels\\_urlt](#) (github3.issues.issue.ShortIssue attribute), 76  
[labels\\_urlt](#) (github3.repos.repo.ShortRepository attribute), 163  
[language](#) (github3.gists.file.ShortGistFile attribute), 49  
[language](#) (github3.repos.repo.Repository attribute), 137  
[languages\(\)](#) (github3.repos.repo.Repository method), 154  
[languages\(\)](#) (github3.repos.repo.ShortRepository method), 180  
[languages\\_url](#) (github3.repos.repo.ShortRepository attribute), 163  
[last\\_message\(\)](#) (github3.github.GitHubStatus method), 75  
[last\\_read\\_at](#) (github3.notifications.Thread attribute), 92  
[last\\_response](#) (github3.structs.GitHubIterator attribute), 206  
[last\\_status](#) (github3.structs.GitHubIterator attribute), 206

- last\_url (github3.structs.GitHubIterator attribute), 206
  - latest\_pages\_build() (github3.repos.repo.Repository method), 154
  - latest\_pages\_build() (github3.repos.repo.ShortRepository method), 180
  - latest\_release() (github3.repos.repo.Repository method), 154
  - latest\_release() (github3.repos.repo.ShortRepository method), 180
  - license() (github3.github.GitHub method), 61
  - license() (github3.repos.repo.Repository method), 154
  - license() (github3.repos.repo.ShortRepository method), 180
  - licenses() (github3.github.GitHub method), 61
  - line (github3.repos.comment.ShortComment attribute), 200
  - links (github3.events.EventReviewComment attribute), 45
  - links (github3.pulls.ReviewComment attribute), 134
  - links (github3.pulls.ShortPullRequest attribute), 121
  - links (github3.repos.branch.Branch attribute), 192
  - links (github3.repos.contents.Contents attribute), 189
  - list\_types() (github3.events.Event static method), 43
  - location (github3.orgs.Organization attribute), 113
  - location (github3.users.User attribute), 215
  - lock() (github3.issues.issue.Issue method), 82
  - lock() (github3.issues.issue.ShortIssue method), 78
  - locked (github3.events.EventPullRequest attribute), 45
  - locked (github3.issues.issue.ShortIssue attribute), 76
  - login (github3.events.EventOrganization attribute), 44
  - login (github3.events.EventUser attribute), 44
  - login (github3.orgs.ShortOrganization attribute), 106
  - login (github3.users.ShortUser attribute), 209
  - login() (github3.github.GitHub method), 61
  - login() (in module github3), 29
- ## M
- mark() (github3.notifications.Thread method), 93
  - mark\_notifications() (github3.repos.repo.Repository method), 155
  - mark\_notifications() (github3.repos.repo.ShortRepository method), 180
  - markdown() (github3.github.GitHub method), 62
  - markdown() (in module github3), 34
  - me() (github3.github.GitHub method), 62
  - members() (github3.orgs.Organization method), 117
  - members() (github3.orgs.ShortOrganization method), 110
  - members() (github3.orgs.ShortTeam method), 99
  - members() (github3.orgs.Team method), 103
  - members\_count (github3.orgs.ShortTeam attribute), 97
  - members\_count (github3.orgs.Team attribute), 101
  - members\_url (github3.orgs.ShortOrganization attribute), 106
  - members\_urlt (github3.orgs.ShortTeam attribute), 97
  - Membership (class in github3.orgs), 104
  - membership\_for() (github3.orgs.ShortTeam method), 99
  - membership\_for() (github3.orgs.Team method), 103
  - membership\_in() (github3.github.GitHub method), 62
  - merge() (github3.pulls.PullRequest method), 129
  - merge() (github3.pulls.ShortPullRequest method), 124
  - merge() (github3.repos.repo.Repository method), 155
  - merge() (github3.repos.repo.ShortRepository method), 181
  - merge\_commit\_sha (github3.pulls.ShortPullRequest attribute), 121
  - mergeable (github3.pulls.PullRequest attribute), 126
  - mergeable\_state (github3.pulls.PullRequest attribute), 126
  - merged (github3.pulls.PullRequest attribute), 126
  - merged\_at (github3.pulls.ShortPullRequest attribute), 121
  - merged\_by (github3.pulls.PullRequest attribute), 126
  - merges\_url (github3.repos.repo.ShortRepository attribute), 163
  - message (github3.git.ShortCommit attribute), 50
  - message (github3.git.Tag attribute), 52
  - messages() (github3.github.GitHubStatus method), 75
  - meta() (github3.github.GitHub method), 62
  - Milestone (class in github3.issues.milestone), 89
  - milestone (github3.issues.issue.ShortIssue attribute), 76
  - milestone() (github3.repos.repo.Repository method), 155
  - milestone() (github3.repos.repo.ShortRepository method), 181
  - milestones() (github3.repos.repo.Repository method), 155
  - milestones() (github3.repos.repo.ShortRepository method), 181
  - milestones\_urlt (github3.repos.repo.ShortRepository attribute), 163
  - MiniCommit (class in github3.repos.commit), 195
  - mirror\_url (github3.repos.repo.Repository attribute), 138
  - mode (github3.git.Hash attribute), 51
- ## N
- name (github3.issues.label.Label attribute), 88
  - name (github3.orgs.Organization attribute), 113
  - name (github3.orgs.ShortTeam attribute), 97
  - name (github3.repos.branch.ShortBranch attribute), 192
  - name (github3.repos.contents.Contents attribute), 189
  - name (github3.repos.hook.Hook attribute), 190
  - name (github3.repos.release.Asset attribute), 196
  - name (github3.repos.release.Release attribute), 198
  - name (github3.repos.repo.ShortRepository attribute), 163
  - name (github3.repos.tag.RepoTag attribute), 191
  - name (github3.search.CodeSearchResult attribute), 204
  - name (github3.users.Plan attribute), 235
  - name (github3.users.User attribute), 215



- network\_count (github3.repos.repo.Repository attribute), 138
- network\_events() (github3.repos.repo.Repository method), 155
- network\_events() (github3.repos.repo.ShortRepository method), 181
- new\_session() (github3.auths.Authorization method), 40
- new\_session() (github3.events.Event method), 43
- new\_session() (github3.github.GitHub method), 62
- new\_session() (github3.issues.comment.IssueComment method), 85
- new\_session() (github3.issues.event.IssueEvent method), 87
- new\_session() (github3.issues.issue.Issue method), 83
- new\_session() (github3.issues.issue.ShortIssue method), 78
- new\_session() (github3.issues.label.Label method), 89
- new\_session() (github3.issues.milestone.Milestone method), 91
- new\_session() (github3.models.GitHubCore method), 239
- new\_session() (github3.notifications.RepositorySubscription method), 96
- new\_session() (github3.notifications.Thread method), 93
- new\_session() (github3.notifications.ThreadSubscription method), 95
- new\_session() (github3.orgs.Membership method), 105
- new\_session() (github3.orgs.Organization method), 117
- new\_session() (github3.orgs.ShortOrganization method), 110
- new\_session() (github3.orgs.ShortTeam method), 100
- new\_session() (github3.orgs.Team method), 103
- new\_session() (github3.pulls.PullDestination method), 131
- new\_session() (github3.pulls.PullFile method), 133
- new\_session() (github3.pulls.PullRequest method), 129
- new\_session() (github3.pulls.ReviewComment method), 135
- new\_session() (github3.pulls.ShortPullRequest method), 124
- new\_session() (github3.repos.repo.Repository method), 156
- new\_session() (github3.repos.repo.ShortRepository method), 181
- new\_session() (github3.repos.repo.StarredRepository method), 187
- new\_session() (github3.structs.GitHubIterator method), 206
- new\_session() (github3.structs.SearchIterator method), 208
- new\_session() (github3.users.AuthenticatedUser method), 222
- new\_session() (github3.users.Collaborator method), 226
- new\_session() (github3.users.Contributor method), 231
- new\_session() (github3.users.Email method), 237
- new\_session() (github3.users.Key method), 234
- new\_session() (github3.users.Plan method), 236
- new\_session() (github3.users.ShortUser method), 212
- new\_session() (github3.users.User method), 217
- note (github3.auths.Authorization attribute), 39
- note\_url (github3.auths.Authorization attribute), 39
- notifications() (github3.github.GitHub method), 62
- notifications() (github3.repos.repo.Repository method), 156
- notifications() (github3.repos.repo.ShortRepository method), 181
- notifications\_url (github3.repos.repo.ShortRepository attribute), 163
- number (github3.events.EventPullRequest attribute), 45
- number (github3.issues.issue.ShortIssue attribute), 76
- number (github3.issues.milestone.Milestone attribute), 90
- number (github3.pulls.ShortPullRequest attribute), 121
- ## O
- object (github3.git.Reference attribute), 52
- object (github3.git.Tag attribute), 52
- octocat() (github3.github.GitHub method), 63
- octocat() (in module github3), 34
- open\_issues\_count (github3.issues.milestone.Milestone attribute), 90
- open\_issues\_count (github3.repos.repo.Repository attribute), 138
- org (github3.events.Event attribute), 42
- Organization (class in github3.orgs), 113
- organization (github3.orgs.Membership attribute), 104
- organization (github3.orgs.Team attribute), 101
- organization() (github3.github.GitHub method), 63
- organization() (in module github3), 34
- organization\_events() (github3.users.AuthenticatedUser method), 222
- organization\_events() (github3.users.Collaborator method), 226
- organization\_events() (github3.users.Contributor method), 231
- organization\_events() (github3.users.ShortUser method), 212
- organization\_events() (github3.users.User method), 217
- organization\_issues() (github3.github.GitHub method), 63
- organization\_memberships() (github3.github.GitHub method), 64
- organization\_url (github3.orgs.Membership attribute), 104
- organizations() (github3.github.GitHub method), 64
- organizations() (github3.users.AuthenticatedUser method), 222
- organizations() (github3.users.Collaborator method), 227
- organizations() (github3.users.Contributor method), 231

- organizations() (github3.users.ShortUser method), 212
  - organizations() (github3.users.User method), 217
  - organizations\_url (github3.users.ShortUser attribute), 209
  - organizations\_with() (github3.github.GitHub method), 64
  - organizations\_with() (in module github3), 33
  - original (github3.structs.GitHubIterator attribute), 207
  - original\_assets (github3.repos.release.Release attribute), 197
  - original\_commit\_id (github3.events.EventReviewComment attribute), 45
  - original\_commit\_id (github3.pulls.ReviewComment attribute), 135
  - original\_content (github3.gists.file.GistFile attribute), 48
  - original\_forks (github3.gists.gist.Gist attribute), 48
  - original\_labels (github3.issues.issue.ShortIssue attribute), 76
  - original\_license (github3.repos.repo.Repository attribute), 137
  - original\_position (github3.events.EventReviewComment attribute), 45
  - original\_position (github3.pulls.ReviewComment attribute), 135
  - original\_protection (github3.repos.branch.Branch attribute), 192
  - original\_teams (github3.repos.branch.ProtectionRestrictions attribute), 193
  - original\_users (github3.repos.branch.ProtectionRestrictions attribute), 193
  - owned\_private\_repos\_count (github3.users.AuthenticatedUser attribute), 219
  - owner (github3.gists.gist.GistFork attribute), 48
  - owner (github3.gists.gist.ShortGist attribute), 47
  - owner (github3.repos.repo.ShortRepository attribute), 163
- P**
- pages() (github3.repos.repo.Repository method), 156
  - pages() (github3.repos.repo.ShortRepository method), 182
  - pages\_builds() (github3.repos.repo.Repository method), 156
  - pages\_builds() (github3.repos.repo.ShortRepository method), 182
  - PagesBuild (class in github3.repos.pages), 200
  - PagesInfo (class in github3.repos.pages), 199
  - params (github3.structs.GitHubIterator attribute), 207
  - parent (github3.repos.repo.Repository attribute), 138
  - parents (github3.git.Commit attribute), 50
  - patch (github3.pulls.PullFile attribute), 133
  - patch() (github3.pulls.PullRequest method), 129
  - patch() (github3.pulls.ShortPullRequest method), 124
  - patch() (github3.repos.comparison.Comparison method), 196
  - patch\_url (github3.pulls.ShortPullRequest attribute), 121
  - patch\_url (github3.repos.comparison.Comparison attribute), 195
  - path (github3.events.EventReviewComment attribute), 46
  - path (github3.git.Hash attribute), 51
  - path (github3.repos.comment.ShortComment attribute), 200
  - path (github3.repos.contents.Contents attribute), 189
  - path (github3.search.CodeSearchResult attribute), 204
  - payload (github3.events.Event attribute), 42
  - payload (github3.repos.deployment.Deployment attribute), 201
  - pending (github3.orgs.Membership attribute), 104
  - permalink\_url (github3.repos.comparison.Comparison attribute), 195
  - permission (github3.orgs.ShortTeam attribute), 97
  - permissions (github3.users.Collaborator attribute), 224
  - ping() (github3.repos.hook.Hook method), 190
  - Plan (class in github3.users), 235
  - plan (github3.users.AuthenticatedUser attribute), 219
  - position (github3.events.EventReviewComment attribute), 46
  - position (github3.repos.comment.ShortComment attribute), 200
  - prerelease (github3.repos.release.Release attribute), 198
  - primary (github3.users.Email attribute), 236
  - private (github3.repos.repo.ShortRepository attribute), 163
  - private\_repos\_count (github3.users.Plan attribute), 235
  - project() (github3.github.GitHub method), 64
  - project() (github3.orgs.Organization method), 118
  - project() (github3.orgs.ShortOrganization method), 111
  - project() (github3.repos.repo.Repository method), 156
  - project() (github3.repos.repo.ShortRepository method), 182
  - project\_card() (github3.github.GitHub method), 64
  - project\_column() (github3.github.GitHub method), 64
  - projects() (github3.orgs.Organization method), 118
  - projects() (github3.orgs.ShortOrganization method), 111
  - projects() (github3.repos.repo.Repository method), 156
  - projects() (github3.repos.repo.ShortRepository method), 182
  - promote() (github3.users.AuthenticatedUser method), 222
  - promote() (github3.users.Collaborator method), 227
  - promote() (github3.users.Contributor method), 231
  - promote() (github3.users.ShortUser method), 212
  - promote() (github3.users.User method), 217
  - protected (github3.repos.branch.Branch attribute), 192
  - protection\_url (github3.repos.branch.Branch attribute), 192
  - ProtectionEnforceAdmins (class in github3.repos.branch), 193

- ProtectionRequiredPullRequestReviews (class in github3.repos.branch), 193
- ProtectionRequiredStatusChecks (class in github3.repos.branch), 193
- ProtectionRestrictions (class in github3.repos.branch), 193
- public (github3.events.Event attribute), 42
- public (github3.gists.gist.ShortGist attribute), 47
- public\_events() (github3.orgs.Organization method), 118
- public\_events() (github3.orgs.ShortOrganization method), 111
- public\_gists() (github3.github.GitHub method), 64
- public\_gists() (in module github3), 32
- public\_gists\_count (github3.users.User attribute), 215
- public\_members() (github3.orgs.Organization method), 118
- public\_members() (github3.orgs.ShortOrganization method), 111
- public\_members\_url (github3.orgs.ShortOrganization attribute), 106
- public\_repos\_count (github3.orgs.Organization attribute), 113
- publicize\_member() (github3.orgs.Organization method), 118
- publicize\_member() (github3.orgs.ShortOrganization method), 111
- published\_at (github3.repos.release.Release attribute), 198
- pubsubhubbub() (github3.github.GitHub method), 65
- pull\_request() (github3.github.GitHub method), 65
- pull\_request() (github3.issues.issue.Issue method), 83
- pull\_request() (github3.issues.issue.ShortIssue method), 79
- pull\_request() (github3.repos.repo.Repository method), 157
- pull\_request() (github3.repos.repo.ShortRepository method), 182
- pull\_request() (in module github3), 34
- pull\_request\_url (github3.events.EventReviewComment attribute), 46
- pull\_request\_url (github3.pulls.ReviewComment attribute), 135
- pull\_request\_urls (github3.issues.issue.ShortIssue attribute), 76
- pull\_requests() (github3.repos.repo.Repository method), 157
- pull\_requests() (github3.repos.repo.ShortRepository method), 183
- PullDestination (class in github3.pulls), 131
- PullFile (class in github3.pulls), 132
- PullRequest (class in github3.pulls), 126
- pulls\_url (github3.repos.repo.ShortRepository attribute), 164
- pushed\_at (github3.repos.repo.Repository attribute), 138
- pusher (github3.repos.pages.PagesBuild attribute), 200
- ## R
- rate\_limit() (github3.github.GitHub method), 65
- rate\_limit() (in module github3), 34
- ratelimit\_remaining (github3.auths.Authorization attribute), 41
- ratelimit\_remaining (github3.events.Event attribute), 43
- ratelimit\_remaining (github3.github.GitHub attribute), 65
- ratelimit\_remaining (github3.issues.comment.IssueComment attribute), 85
- ratelimit\_remaining (github3.issues.event.IssueEvent attribute), 87
- ratelimit\_remaining (github3.issues.issue.Issue attribute), 83
- ratelimit\_remaining (github3.issues.issue.ShortIssue attribute), 79
- ratelimit\_remaining (github3.issues.label.Label attribute), 89
- ratelimit\_remaining (github3.issues.milestone.Milestone attribute), 91
- ratelimit\_remaining (github3.models.GitHubCore attribute), 239
- ratelimit\_remaining (github3.notifications.RepositorySubscription attribute), 96
- ratelimit\_remaining (github3.notifications.Thread attribute), 93
- ratelimit\_remaining (github3.notifications.ThreadSubscription attribute), 95
- ratelimit\_remaining (github3.orgs.Membership attribute), 105
- ratelimit\_remaining (github3.orgs.Organization attribute), 118
- ratelimit\_remaining (github3.orgs.ShortOrganization attribute), 111
- ratelimit\_remaining (github3.orgs.ShortTeam attribute), 100
- ratelimit\_remaining (github3.orgs.Team attribute), 103
- ratelimit\_remaining (github3.pulls.PullDestination attribute), 132
- ratelimit\_remaining (github3.pulls.PullFile attribute), 133
- ratelimit\_remaining (github3.pulls.PullRequest attribute), 129
- ratelimit\_remaining (github3.pulls.ReviewComment attribute), 136
- ratelimit\_remaining (github3.pulls.ShortPullRequest attribute), 124
- ratelimit\_remaining (github3.repos.repo.Repository attribute), 157
- ratelimit\_remaining (github3.repos.repo.ShortRepository attribute), 183
- ratelimit\_remaining (github3.repos.repo.StarredRepository attribute), 188

- ratelimit\_remaining (github3.structs.GitHubIterator attribute), 207
- ratelimit\_remaining (github3.structs.SearchIterator attribute), 208
- ratelimit\_remaining (github3.users.AuthenticatedUser attribute), 222
- ratelimit\_remaining (github3.users.Collaborator attribute), 227
- ratelimit\_remaining (github3.users.Contributor attribute), 232
- ratelimit\_remaining (github3.users.Email attribute), 237
- ratelimit\_remaining (github3.users.Key attribute), 234
- ratelimit\_remaining (github3.users.Plan attribute), 236
- ratelimit\_remaining (github3.users.ShortUser attribute), 212
- ratelimit\_remaining (github3.users.User attribute), 217
- raw\_url (github3.gists.file.ShortGistFile attribute), 48
- raw\_url (github3.pulls.PullFile attribute), 133
- readme() (github3.repos.repo.Repository method), 157
- readme() (github3.repos.repo.ShortRepository method), 183
- reason (github3.notifications.RepositorySubscription attribute), 96
- reason (github3.notifications.Thread attribute), 92
- reason (github3.notifications.ThreadSubscription attribute), 94
- received\_events() (github3.users.AuthenticatedUser method), 222
- received\_events() (github3.users.Collaborator method), 227
- received\_events() (github3.users.Contributor method), 232
- received\_events() (github3.users.ShortUser method), 212
- received\_events() (github3.users.User method), 217
- received\_events\_url (github3.users.ShortUser attribute), 209
- ref (github3.git.Reference attribute), 52
- ref (github3.pulls.PullDestination attribute), 131
- ref (github3.repos.deployment.Deployment attribute), 201
- ref() (github3.repos.repo.Repository method), 157
- ref() (github3.repos.repo.ShortRepository method), 183
- Reference (class in github3.git), 52
- refresh() (github3.auths.Authorization method), 41
- refresh() (github3.events.Event method), 43
- refresh() (github3.github.GitHub method), 66
- refresh() (github3.issues.comment.IssueComment method), 85
- refresh() (github3.issues.event.IssueEvent method), 87
- refresh() (github3.issues.issue.Issue method), 83
- refresh() (github3.issues.issue.ShortIssue method), 79
- refresh() (github3.issues.label.Label method), 89
- refresh() (github3.issues.milestone.Milestone method), 91
- refresh() (github3.models.GitHubCore method), 239
- refresh() (github3.notifications.RepositorySubscription method), 96
- refresh() (github3.notifications.Thread method), 93
- refresh() (github3.notifications.ThreadSubscription method), 95
- refresh() (github3.orgs.Membership method), 105
- refresh() (github3.orgs.Organization method), 119
- refresh() (github3.orgs.ShortOrganization method), 112
- refresh() (github3.orgs.ShortTeam method), 100
- refresh() (github3.orgs.Team method), 103
- refresh() (github3.pulls.PullDestination method), 132
- refresh() (github3.pulls.PullFile method), 134
- refresh() (github3.pulls.PullRequest method), 129
- refresh() (github3.pulls.ReviewComment method), 136
- refresh() (github3.pulls.ShortPullRequest method), 125
- refresh() (github3.repos.repo.Repository method), 158
- refresh() (github3.repos.repo.ShortRepository method), 183
- refresh() (github3.repos.repo.StarredRepository method), 188
- refresh() (github3.structs.GitHubIterator method), 207
- refresh() (github3.structs.SearchIterator method), 208
- refresh() (github3.users.AuthenticatedUser method), 223
- refresh() (github3.users.Collaborator method), 227
- refresh() (github3.users.Contributor method), 232
- refresh() (github3.users.Email method), 237
- refresh() (github3.users.Key method), 234
- refresh() (github3.users.Plan method), 236
- refresh() (github3.users.ShortUser method), 213
- refresh() (github3.users.User method), 218
- refs() (github3.repos.repo.Repository method), 158
- refs() (github3.repos.repo.ShortRepository method), 184
- Release (class in github3.repos.release), 197
- release() (github3.repos.repo.Repository method), 158
- release() (github3.repos.repo.ShortRepository method), 184
- release\_from\_tag() (github3.repos.repo.Repository method), 158
- release\_from\_tag() (github3.repos.repo.ShortRepository method), 184
- releases() (github3.repos.repo.Repository method), 159
- releases() (github3.repos.repo.ShortRepository method), 184
- releases\_url() (github3.repos.repo.ShortRepository attribute), 164
- remove\_all\_labels() (github3.issues.issue.Issue method), 83
- remove\_all\_labels() (github3.issues.issue.ShortIssue method), 79
- remove\_collaborator() (github3.repos.repo.Repository method), 159
- remove\_collaborator() (github3.repos.repo.ShortRepository method), 185



- remove\_contexts() (github3.repos.branch.ProtectionRequiredStatusCheck method), 194
- remove\_label() (github3.issues.issue.Issue method), 83
- remove\_label() (github3.issues.issue.ShortIssue method), 79
- remove\_member() (github3.orgs.Organization method), 119
- remove\_member() (github3.orgs.ShortOrganization method), 112
- remove\_member() (github3.orgs.ShortTeam method), 100
- remove\_member() (github3.orgs.Team method), 103
- remove\_repository() (github3.orgs.Organization method), 119
- remove\_repository() (github3.orgs.ShortOrganization method), 112
- remove\_repository() (github3.orgs.ShortTeam method), 100
- remove\_repository() (github3.orgs.Team method), 104
- remove\_scopes() (github3.auths.Authorization method), 41
- rename() (github3.users.AuthenticatedUser method), 223
- rename() (github3.users.Collaborator method), 228
- rename() (github3.users.Contributor method), 232
- rename() (github3.users.ShortUser method), 213
- rename() (github3.users.User method), 218
- reopen() (github3.issues.issue.Issue method), 83
- reopen() (github3.issues.issue.ShortIssue method), 79
- reopen() (github3.pulls.PullRequest method), 130
- reopen() (github3.pulls.ShortPullRequest method), 125
- replace\_contexts() (github3.repos.branch.ProtectionRequiredStatusCheck method), 194
- replace\_labels() (github3.issues.issue.Issue method), 84
- replace\_labels() (github3.issues.issue.ShortIssue method), 80
- replace\_scopes() (github3.auths.Authorization method), 41
- reply() (github3.pulls.ReviewComment method), 136
- repo (github3.events.Event attribute), 42
- repo (github3.pulls.PullDestination attribute), 131
- RepoComment (class in github3.repos.comment), 201
- RepoCommit (class in github3.repos.commit), 195
- repos\_count (github3.orgs.ShortTeam attribute), 97
- repos\_count (github3.orgs.Team attribute), 101
- repos\_url (github3.orgs.ShortOrganization attribute), 106
- repos\_url (github3.users.ShortUser attribute), 209
- repositories() (github3.github.GitHub method), 66
- repositories() (github3.orgs.Organization method), 119
- repositories() (github3.orgs.ShortOrganization method), 112
- repositories() (github3.orgs.ShortTeam method), 100
- repositories() (github3.orgs.Team method), 104
- repositories\_by() (github3.github.GitHub method), 66
- repositories\_by() (in module github3), 33
- Repository (class in github3.repos.repo), 136
- repository (github3.notifications.Thread attribute), 92
- repository (github3.pulls.PullDestination attribute), 131
- repository (github3.pulls.ShortPullRequest attribute), 121
- repository (github3.repos.repo.StarredRepository attribute), 187
- repository (github3.repos.status.CombinedStatus attribute), 203
- repository (github3.search.CodeSearchResult attribute), 204
- repository() (github3.github.GitHub method), 67
- repository() (in module github3), 35
- repository\_url (github3.notifications.RepositorySubscription attribute), 96
- repository\_url (github3.repos.issue\_import.ImportedIssue attribute), 191
- repository\_with\_id() (github3.github.GitHub method), 67
- RepositoryIssueEvent (class in github3.issues.event), 87
- RepositorySearchResult (class in github3.search), 205
- RepositorySubscription (class in github3.notifications), 95
- RepoTag (class in github3.repos.tag), 191
- required\_pull\_request\_reviews (github3.repos.branch.BranchProtection attribute), 192
- required\_status\_checks (github3.repos.branch.BranchProtection attribute), 193
- requires\_app\_credentials() (in module github3.decorators), 238
- requires\_basic\_auth() (in module github3.decorators), 238
- requires\_status\_checks() (in module github3.decorators), 238
- restrictions (github3.repos.branch.BranchProtection attribute), 192
- review\_comment\_url (github3.pulls.ShortPullRequest attribute), 121
- review\_comments() (github3.pulls.PullRequest method), 130
- review\_comments() (github3.pulls.ShortPullRequest method), 125
- review\_comments\_count (github3.pulls.PullRequest attribute), 126
- review\_comments\_url (github3.pulls.ShortPullRequest attribute), 121
- ReviewComment (class in github3.pulls), 134
- reviews() (github3.pulls.PullRequest method), 130
- reviews() (github3.pulls.ShortPullRequest method), 125
- revoke\_authorization() (github3.github.GitHub method), 67
- revoke\_authorizations() (github3.github.GitHub method), 67
- revoke\_impersonation() (github3.users.AuthenticatedUser method), 223

revoke\_impersonation() (github3.users.Collaborator method), 228  
 revoke\_impersonation() (github3.users.Contributor method), 232  
 revoke\_impersonation() (github3.users.ShortUser method), 213  
 revoke\_impersonation() (github3.users.User method), 218  
 revoke\_membership() (github3.orgs.ShortTeam method), 101  
 revoke\_membership() (github3.orgs.Team method), 104

## S

scopes (github3.auths.Authorization attribute), 39  
 score (github3.search.CodeSearchResult attribute), 204  
 score (github3.search.IssueSearchResult attribute), 205  
 score (github3.search.RepositorySearchResult attribute), 205  
 score (github3.search.UserSearchResult attribute), 205  
 search\_code() (github3.github.GitHub method), 67  
 search\_code() (in module github3), 35  
 search\_issues() (github3.github.GitHub method), 68  
 search\_issues() (in module github3), 36  
 search\_repositories() (github3.github.GitHub method), 69  
 search\_repositories() (in module github3), 37  
 search\_users() (github3.github.GitHub method), 70  
 search\_users() (in module github3), 37  
 SearchIterator (class in github3.structs), 207  
 set() (github3.notifications.RepositorySubscription method), 97  
 set() (github3.notifications.ThreadSubscription method), 95  
 set\_client\_id() (github3.github.GitHub method), 70  
 set\_subscription() (github3.notifications.Thread method), 93  
 set\_user\_agent() (github3.github.GitHub method), 71  
 sha (github3.git.Blob attribute), 50  
 sha (github3.git.Commit attribute), 50  
 sha (github3.git.CommitTree attribute), 51  
 sha (github3.git.GitObject attribute), 52  
 sha (github3.git.Hash attribute), 51  
 sha (github3.git.Tag attribute), 52  
 sha (github3.git.Tree attribute), 51  
 sha (github3.pulls.PullDestination attribute), 131  
 sha (github3.pulls.PullFile attribute), 133  
 sha (github3.repos.contents.Contents attribute), 189  
 sha (github3.repos.deployment.Deployment attribute), 201  
 sha (github3.repos.status.CombinedStatus attribute), 203  
 sha (github3.search.CodeSearchResult attribute), 204  
 ShortBranch (class in github3.repos.branch), 192  
 ShortComment (class in github3.repos.comment), 200  
 ShortCommit (class in github3.git), 50  
 ShortCommit (class in github3.repos.commit), 195

ShortGist (class in github3.gists.gist), 46  
 ShortGistFile (class in github3.gists.file), 48  
 ShortIssue (class in github3.issues.issue), 75  
 ShortOrganization (class in github3.orgs), 106  
 ShortPullRequest (class in github3.pulls), 120  
 ShortRepository (class in github3.repos.repo), 161  
 ShortStatus (class in github3.repos.status), 202  
 ShortTeam (class in github3.orgs), 97  
 ShortUser (class in github3.users), 209  
 site\_admin (github3.users.ShortUser attribute), 209  
 size (github3.gists.file.ShortGistFile attribute), 49  
 size (github3.git.Blob attribute), 50  
 size (github3.git.Hash attribute), 51  
 size (github3.repos.contents.Contents attribute), 189  
 size (github3.repos.release.Asset attribute), 196  
 size (github3.repos.repo.Repository attribute), 138  
 slug (github3.orgs.ShortTeam attribute), 98  
 source (github3.repos.repo.Repository attribute), 138  
 space (github3.users.Plan attribute), 235  
 ssh\_url (github3.repos.repo.Repository attribute), 138  
 star() (github3.github.GitHub method), 71  
 stargazers() (github3.repos.repo.Repository method), 159  
 stargazers() (github3.repos.repo.ShortRepository method), 185  
 stargazers\_count (github3.repos.repo.Repository attribute), 138  
 stargazers\_url (github3.repos.repo.ShortRepository attribute), 164  
 starred() (github3.github.GitHub method), 71  
 starred\_at (github3.repos.repo.StarredRepository attribute), 187  
 starred\_by() (github3.github.GitHub method), 71  
 starred\_by() (in module github3), 33  
 starred\_repositories() (github3.users.AuthenticatedUser method), 223  
 starred\_repositories() (github3.users.Collaborator method), 228  
 starred\_repositories() (github3.users.Contributor method), 232  
 starred\_repositories() (github3.users.ShortUser method), 213  
 starred\_repositories() (github3.users.User method), 218  
 starred\_url (github3.users.ShortUser attribute), 209  
 StarredRepository (class in github3.repos.repo), 187  
 state (github3.events.EventPullRequest attribute), 45  
 state (github3.issues.issue.ShortIssue attribute), 76  
 state (github3.issues.milestone.Milestone attribute), 90  
 state (github3.orgs.Membership attribute), 104  
 state (github3.pulls.ShortPullRequest attribute), 121  
 state (github3.repos.deployment.DeploymentStatus attribute), 202  
 state (github3.repos.release.Asset attribute), 196  
 state (github3.repos.status.CombinedStatus attribute), 203

- state (github3.repos.status.ShortStatus attribute), 203
  - Status (class in github3.repos.status), 203
  - status (github3.pulls.PullFile attribute), 133
  - status (github3.repos.comparison.Comparison attribute), 195
  - status (github3.repos.issue\_import.ImportedIssue attribute), 191
  - status (github3.repos.pages.PagesBuild attribute), 200
  - status (github3.repos.pages.PagesInfo attribute), 200
  - status() (github3.github.GitHubStatus method), 75
  - statuses (github3.repos.status.CombinedStatus attribute), 203
  - statuses() (github3.repos.deployment.Deployment method), 202
  - statuses() (github3.repos.repo.Repository method), 159
  - statuses() (github3.repos.repo.ShortRepository method), 185
  - statuses\_url (github3.repos.deployment.Deployment attribute), 201
  - statuses\_urlt (github3.repos.repo.ShortRepository attribute), 164
  - subject (github3.notifications.Thread attribute), 92
  - submodule\_git\_url (github3.repos.contents.Contents attribute), 189
  - subscribe() (github3.repos.repo.Repository method), 159
  - subscribe() (github3.repos.repo.ShortRepository method), 185
  - subscribed (github3.notifications.RepositorySubscription attribute), 96
  - subscribed (github3.notifications.ThreadSubscription attribute), 94
  - subscribers() (github3.repos.repo.Repository method), 160
  - subscribers() (github3.repos.repo.ShortRepository method), 185
  - subscribers\_count (github3.repos.repo.Repository attribute), 138
  - subscribers\_url (github3.repos.repo.ShortRepository attribute), 164
  - subscription() (github3.notifications.Thread method), 94
  - subscription() (github3.repos.repo.Repository method), 160
  - subscription() (github3.repos.repo.ShortRepository method), 186
  - subscription\_url (github3.repos.repo.ShortRepository attribute), 164
  - subscriptions() (github3.github.GitHub method), 71
  - subscriptions() (github3.users.AuthenticatedUser method), 224
  - subscriptions() (github3.users.Collaborator method), 228
  - subscriptions() (github3.users.Contributor method), 233
  - subscriptions() (github3.users.ShortUser method), 213
  - subscriptions() (github3.users.User method), 219
  - subscriptions\_for() (github3.github.GitHub method), 72
  - subscriptions\_for() (in module github3), 34
  - subscriptions\_url (github3.users.ShortUser attribute), 209
  - suspend() (github3.users.AuthenticatedUser method), 224
  - suspend() (github3.users.Collaborator method), 228
  - suspend() (github3.users.Contributor method), 233
  - suspend() (github3.users.ShortUser method), 214
  - suspend() (github3.users.User method), 219
  - svn\_url (github3.repos.repo.Repository attribute), 138
- ## T
- Tag (class in github3.git), 52
  - tag (github3.git.Tag attribute), 52
  - tag() (github3.repos.repo.Repository method), 160
  - tag() (github3.repos.repo.ShortRepository method), 186
  - tag\_name (github3.repos.release.Release attribute), 198
  - tagger (github3.git.Tag attribute), 52
  - tags() (github3.repos.repo.Repository method), 160
  - tags() (github3.repos.repo.ShortRepository method), 186
  - tags\_url (github3.repos.repo.ShortRepository attribute), 164
  - tarball\_url (github3.repos.release.Release attribute), 198
  - tarball\_url (github3.repos.tag.RepoTag attribute), 191
  - target (github3.repos.contents.Contents attribute), 189
  - target\_commitish (github3.repos.release.Release attribute), 198
  - target\_url (github3.repos.deployment.DeploymentStatus attribute), 202
  - target\_url (github3.repos.status.ShortStatus attribute), 203
  - Team (class in github3.orgs), 101
  - team() (github3.orgs.Organization method), 119
  - team() (github3.orgs.ShortOrganization method), 112
  - teams() (github3.orgs.Organization method), 120
  - teams() (github3.orgs.ShortOrganization method), 113
  - teams() (github3.repos.branch.ProtectionRestrictions method), 193
  - teams() (github3.repos.repo.Repository method), 160
  - teams() (github3.repos.repo.ShortRepository method), 186
  - teams\_url (github3.repos.branch.ProtectionRestrictions attribute), 193
  - teams\_url (github3.repos.repo.ShortRepository attribute), 164
  - test() (github3.repos.hook.Hook method), 190
  - text\_matches (github3.search.CodeSearchResult attribute), 205
  - text\_matches (github3.search.IssueSearchResult attribute), 205
  - text\_matches (github3.search.RepositorySearchResult attribute), 205
  - text\_matches (github3.search.UserSearchResult attribute), 205
  - Thread (class in github3.notifications), 92

- thread\_url (github3.notifications.ThreadSubscription attribute), 94
- ThreadSubscription (class in github3.notifications), 94
- title (github3.events.EventPullRequest attribute), 45
- title (github3.issues.issue.ShortIssue attribute), 76
- title (github3.issues.milestone.Milestone attribute), 90
- title (github3.pulls.ShortPullRequest attribute), 121
- token (github3.auths.Authorization attribute), 40
- token\_last\_eight (github3.auths.Authorization attribute), 40
- total (github3.repos.stats.ContributorStats attribute), 204
- total\_commits (github3.repos.comparison.Comparison attribute), 195
- total\_count (github3.repos.status.CombinedStatus attribute), 203
- total\_count (github3.structs.SearchIterator attribute), 208
- totoal (github3.gists.history.GistHistory attribute), 49
- Tree (class in github3.git), 51
- tree (github3.git.ShortCommit attribute), 50
- tree (github3.git.Tree attribute), 51
- tree() (github3.repos.repo.Repository method), 161
- tree() (github3.repos.repo.ShortRepository method), 186
- trees\_url (github3.repos.repo.ShortRepository attribute), 164
- truncated (github3.gists.file.GistFile attribute), 48
- truncated (github3.gists.gist.Gist attribute), 48
- type (github3.events.Event attribute), 42
- type (github3.gists.file.ShortGistFile attribute), 49
- type (github3.git.GitObject attribute), 52
- type (github3.git.Hash attribute), 51
- type (github3.orgs.ShortOrganization attribute), 106
- type (github3.repos.contents.Contents attribute), 189
- type (github3.users.ShortUser attribute), 209
- ## U
- unfollow() (github3.github.GitHub method), 72
- unlock() (github3.issues.issue.Issue method), 84
- unlock() (github3.issues.issue.ShortIssue method), 80
- unread (github3.notifications.Thread attribute), 92
- unstar() (github3.github.GitHub method), 72
- unsuspend() (github3.users.AuthenticatedUser method), 224
- unsuspend() (github3.users.Collaborator method), 228
- unsuspend() (github3.users.Contributor method), 233
- unsuspend() (github3.users.ShortUser method), 214
- unsuspend() (github3.users.User method), 219
- update() (github3.issues.label.Label method), 89
- update() (github3.issues.milestone.Milestone method), 91
- update() (github3.pulls.PullRequest method), 130
- update() (github3.pulls.ShortPullRequest method), 125
- update() (github3.repos.branch.ProtectionRequiredStatusCheck method), 194
- update() (github3.repos.contents.Contents method), 189
- update() (github3.users.Key method), 235
- update\_me() (github3.github.GitHub method), 72
- updated\_at (github3.auths.Authorization attribute), 40
- updated\_at (github3.events.EventIssueComment attribute), 46
- updated\_at (github3.events.EventReviewComment attribute), 46
- updated\_at (github3.gists.gist.GistFork attribute), 48
- updated\_at (github3.gists.gist.ShortGist attribute), 47
- updated\_at (github3.issues.comment.IssueComment attribute), 84
- updated\_at (github3.issues.issue.ShortIssue attribute), 76
- updated\_at (github3.issues.milestone.Milestone attribute), 90
- updated\_at (github3.notifications.Thread attribute), 92
- updated\_at (github3.orgs.Team attribute), 101
- updated\_at (github3.pulls.ReviewComment attribute), 135
- updated\_at (github3.pulls.ShortPullRequest attribute), 121
- updated\_at (github3.repos.comment.ShortComment attribute), 201
- updated\_at (github3.repos.deployment.Deployment attribute), 201
- updated\_at (github3.repos.hook.Hook attribute), 190
- updated\_at (github3.repos.issue\_import.ImportedIssue attribute), 191
- updated\_at (github3.repos.pages.PagesBuild attribute), 200
- updated\_at (github3.repos.release.Asset attribute), 197
- updated\_at (github3.repos.repo.Repository attribute), 138
- updated\_at (github3.repos.status.ShortStatus attribute), 203
- updated\_at (github3.users.User attribute), 215
- upload\_asset() (github3.repos.release.Release method), 199
- upload\_url (github3.repos.release.Release attribute), 198
- url (github3.gists.gist.GistFork attribute), 48
- url (github3.gists.gist.ShortGist attribute), 47
- url (github3.gists.history.GistHistory attribute), 49
- url (github3.orgs.ShortOrganization attribute), 106
- url (github3.pulls.ShortPullRequest attribute), 120
- url (github3.repos.repo.ShortRepository attribute), 161
- url (github3.structs.GitHubIterator attribute), 207
- url (github3.users.ShortUser attribute), 209
- User (class in github3.users), 214
- user (github3.events.EventIssueComment attribute), 46
- user (github3.events.EventReviewComment attribute), 46
- user (github3.gists.history.GistHistory attribute), 49
- user (github3.issues.comment.IssueComment attribute), 84
- user (github3.issues.issue.ShortIssue attribute), 76
- user (github3.pulls.PullDestination attribute), 131
- user (github3.pulls.ReviewComment attribute), 135
- user (github3.pulls.ShortPullRequest attribute), 121

user (github3.repos.comment.ShortComment attribute),  
201  
user (github3.search.UserSearchResult attribute), 205  
user() (github3.github.GitHub method), 73  
user() (in module github3), 38  
user\_issues() (github3.github.GitHub method), 73  
user\_teams() (github3.github.GitHub method), 73  
user\_with\_id() (github3.github.GitHub method), 73  
users() (github3.repos.branch.ProtectionRestrictions  
method), 193  
users\_url (github3.repos.branch.ProtectionRestrictions  
attribute), 193  
UserSearchResult (class in github3.search), 205

## V

verification (github3.git.Commit attribute), 50  
verified (github3.users.Email attribute), 236  
version (github3.gists.history.GistHistory attribute), 49

## W

watchers\_count (github3.repos.repo.Repository at-  
tribute), 138  
weekly\_commit\_count() (github3.repos.repo.Repository  
method), 161  
weekly\_commit\_count() (github3.repos.repo.ShortRepository  
method), 186  
weeks (github3.repos.stats.ContributorStats attribute),  
204

## Z

zen() (github3.github.GitHub method), 73  
zen() (in module github3), 38  
zipball\_url (github3.repos.release.Release attribute), 198  
zipball\_url (github3.repos.tag.RepoTag attribute), 191