github3.py Documentation

Release 1.0.0a4

Ian Cordasco

Jul 12, 2017

Contents

Release v1.0.0a4.

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. Let me demonstrate this with a code example.

CHAPTER 1

Example

Let's get information about a user:

```
from github3 import login

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

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

print(sigmavirus24.name)
# Ian 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')]
```

More Examples

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 prompt 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. (**VERY VERY BAD**)
3. Ask the user to supply their credentials once, get an authorization token and store that for later use.

The first isn't a bad method at all, it just unfortunately may lead to unhappy users, this should always be an option though. The second (as I already noted) is a bad idea. Even if you obfuscate the username and password, they can still be discovered and no level of obfuscation is clever enough. (May I also take this moment to remind people that base64 is **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.

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'
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
```

Hopefully this helps someone.

Gist Code Examples

Examples with `Gists`

Listing gists after authenticating

```
from github3 import login
```

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

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)
```

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.

Git Code Examples

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

Creating a Blob Object

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

```
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'
```

Creating a Tag Object

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

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

GitHub Examples

Examples using the *GitHub* object.

Assumptions

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

```
from github3 import login, GitHub
from getpass import getpass, getuser
import sys
try:
    import readline
except ImportError:
    pass

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

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

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

g = login(user, password)
```

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

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

```
anon = GitHub()
```

Also let's define the following constants:

```
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

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.')
```

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))
```

Creating a new repository

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

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

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

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

Follow another user on GitHub

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

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

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

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

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

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

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

Changing your user information

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

```
new_name = 'J. Smith'
blog = 'http://www.example.com/'
company = 'Vandelay Industries'
bio = """# J. Smith
```

```
A simple man working at a latex factory
"""

if g.update_user(new_name, blog, company, bio=bio):
    print('Profile updated.')
```

This is the same as:

```
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`

Administrating Issues

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

```
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')
```

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()
```

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
```

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)
```

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)
```

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)

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

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
            MMMMMMMMMMMMMMMMMM
            MMMMMMMMMMMMMMMMMM      ___________
           MMMMMMMMMMMMMMMMMMMM    |           |
          MMMMMMMMMMMMMMMMMMMMMM   | Hey Ian! |
         MMMMMMMMMMMMMMMMMMMMMMMM   |_   _______|
         MMMM::- -:::::::- -::MMMM    |/
          MM~:~   ~::::::~   ~:~MM
     .. MMMMM::. .:::+:::. .::MMMMM ..
           .MM::::: ._. :::::MM.
              MMMM;::::::;MMMM
       -MM         MMMMMMM
       ^  M+     MMMMMMMMM
           MMMMMMM MM MM MM
                MM MM MM MM
                MM MM MM MM
             .~~MM~MM~MM~MM~~.
          ~~~~MM:~MM~~~MM~:MM~~~~
         ~~~~~~==~==~~~~==~==~~~~~~
          ~~~~~~==~==~==~==~~~~~~
              :~==~==~==~==~~

What do you think about github3.py?

            MMM.            .MMM
            MMMMMMMMMMMMMMMMMM
            MMMMMMMMMMMMMMMMMM      ______________________
           MMMMMMMMMMMMMMMMMMMM    |                      |
          MMMMMMMMMMMMMMMMMMMMMM   | github3.py rocks! |
         MMMMMMMMMMMMMMMMMMMMMMMM   |_   ________________|
         MMMM::- -:::::::- -::MMMM    |/
          MM~:~   ~::::::~   ~:~MM
     .. MMMMM::. .:::+:::. .::MMMMM ..
           .MM::::: ._. :::::MM.
              MMMM;::::::;MMMM
       -MM         MMMMMMM
       ^  M+     MMMMMMMMM
           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 2

Modules

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 the user (developer).

To authenticate, you simply 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 create a new *GitHub* object, e.g.,

```
from github3 import GitHub

gh = GitHub()
```

Or you can simply use the following 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

---

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 Issues

---

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 *Organization*

---

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*)
Returns a Organization object for the login name

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

Returns *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**()
Returns 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 dict

---

github3.**repository**(*owner*, *repository*)
Returns a Repository object for the specified combination of owner and repository

Parameters

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

Returns `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 qualifers:

- `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 qualifers:

- `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 qualifers:

- `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 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 *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')
```

Authorization

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

class github3.auths.**Authorization**(*json*, *session=None*)

The *Authorization* object.

Two authorization instances can be checked like so:

```
a1 == a2
a1 != a2
```

And is equivalent to:

```
a1.id == a2.id
a1.id != a2.id
```

See also: http://developer.github.com/v3/oauth/#oauth-authorizations-api

add_scopes(*scopes*, *note=None*, *note_url=None*)
Adds 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.

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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

Events

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

Event Objects

class github3.events.**Event**(*json*, *session=None*)
The *Event* object. 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
```

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

static list_types()
List available payload types.

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']
<Repository [eweap/redactor-js]>
```

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

Gists

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

Gist Objects

class github3.gists.gist.**Gist**(*json*, *session=None*)

This object holds all the information returned by Github about a gist.

With it you can comment on or fork the gist (assuming you are authenticated), edit or delete the gist (assuming you own it). You can also "star" or "unstar" the gist (again assuming you have authenticated).

Two gist instances can be checked like so:

```
g1 == g2
g1 != g2
```

And is equivalent to:

```
g1.id == g2.id
g1.id != g2.id
```

See also: http://developer.github.com/v3/gists/

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

comments(*number=-1*, *etag=None*)
Iterate over comments on this gist.

Parameters

- **number** (*int*) – (optional), number of comments to iterate over. Default: -1 will iterate over all comments on the gist
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of *GistComment*

commits(*number=-1*, *etag=None*)
Iterate over the commits on this gist.

These commits will be requested from the API and should be the same as what is in `Gist.history`.

New in version 0.6.

Changed in version 0.9: Added param `etag`.

Parameters

- **number** (*int*) – (optional), number of commits to iterate over. Default: -1 will iterate over all commits associated with this gist.
- **etag** (*str*) – (optional), ETag from a previous request to this endpoint.

Returns generator of *GistHistory*

create_comment(*body*)
Create a comment on this gist.

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

Returns *GistComment*

delete()
Delete this gist.

Returns bool – whether the deletion was successful

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

Parameters

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

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

Returns bool – whether the edit was successful

files()
Iterator over the files stored in this gist.

Returns generator of :class'GistFile <github3.gists.file.GistFile>'

fork()
Fork this gist.

Returns *Gist* if successful, None otherwise

forks(*number=-1*, *etag=None*)
Iterator of forks of this gist.

Changed in version 0.9: Added params number and etag.

Parameters

- **number** (*int*) – (optional), number of forks to iterate over. Default: -1 will iterate over all forks of this gist.
- **etag** (*str*) – (optional), ETag from a previous request to this endpoint.

Returns generator of *Gist*

from_dict(*json_dict*)
Return an instance of this class formed from json_dict.

from_json(*json*)
Return an instance of this class formed from json.

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

Returns bool – True if it is starred, False otherwise

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

star()
Star this gist.

Returns bool – True if successful, False otherwise

unstar()
Un-star this gist.

Returns bool – True if successful, False otherwise

---

class github3.gists.comment.**GistComment**(*json*, *session=None*)
This object represents a comment on a gist.

Two comment instances can be checked like so:

```
c1 == c2
c1 != c2
```

And is equivalent to:

```
c1.id == c2.id
c1.id != c2.id
```

See also: http://developer.github.com/v3/gists/comments/

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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.gists.file.**GistFile**(*json*, *session=None*)
This represents the file object returned by interacting with gists.

It stores the raw url of the file, the file name, language, size and content.

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

content()
Retrieve contents of file from key 'raw_url' if there is no 'content' key in Gist object.

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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.gists.history.**GistHistory**(*json*, *session=None*)
Thisobject represents one version (or revision) of a gist.

Two history instances can be checked like so:

```
h1 == h2
h1 != h2
```

And is equivalent to:

```
h1.version == h2.version
h1.version != h2.version
```

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

get_gist()
Retrieve the gist at this version.

Returns *Gist*

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

Git

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

- *Blob*
- *Commit*
- *GitData*

- *GitObject*
- *Hash*
- *Reference*
- *Tag*
- *Tree*

Git Objects

class github3.git.**Blob**(*json*, *session=None*)

The *Blob* object.

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

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.git.**Commit**(*json*, *session=None*)

The *Commit* object. This represents a commit made in a repository.

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

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.git.**GitData**(*json*, *session=None*)

The *GitData* object. This isn't directly returned to the user (developer) ever. This is used to prevent duplication of some common items among other Git Data objects.

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.git.**GitObject**(*json*, *session=None*)

The *GitObject* object.

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.git.**Hash**(*json*, *session=None*)

The *Hash* object.

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

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.git.**Reference**(*json*, *session=None*)

The *Reference* object. This represents a reference created on a repository.

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

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 reference.

Returns bool

from_dict(*json_dict*)

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

from_json(*json*)

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

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(*sha*, *force=False*)
Update this reference.

Parameters

- **sha** (*str*) – (required), sha of the reference
- **force** (*bool*) – (optional), force the update or not

Returns bool

---

class github3.git.**Tag**(*json*, *session=None*)
The *Tag* object.

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

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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.git.**Tree**(*json*, *session=None*)
The *Tree* object.

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

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

recurse()
Recurse into the tree.

Returns *Tree*

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

GitHub

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

Examples

Examples utilizing this object can be found *here*.

GitHub Object

class github3.github.**GitHub**(*username=u''*, *password=u''*, *token=u''*)
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 email addresses in `addresses` to the authenticated user's account.

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

Returns list of `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 *Event*s

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 this 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 *Organization*

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 this 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 `Repository`

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 `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. 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`

authorize(*username*, *password*, *scopes=None*, *note=u''*, *note_url=u''*, *client_id=u''*, *client_secret=u''*)
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 `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 bool

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

If no login was provided, it will be anonymous.

Parameters

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

Returns `Gist`

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

`body`, `assignee`, `milestone`, `labels` are all optional.

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

Returns `Issue` if successful

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 *Key*

create_repository(*name*, *description=u''*, *homepage=u''*, *private=False*, *has_issues=True*, *has_wiki=True*, *auto_init=False*, *gitignore_template=u''*)

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 `Repository`

delete_email_addresses(*addresses=[]*)

Delete the email addresses in `addresses` from the authenticated user's account.

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

Returns bool

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

Iterate over email addresses for the authenticated user.

Parameters

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

Returns generator of dicts

emojis()

Retrieves 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

follow(*username*)
Make the authenticated user follow the provided username.

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

Returns 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 `ShortUsers`

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 `ShortUsers`

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 `ShortUsers`

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 ShortUsers

from_dict(*json_dict*)
Return an instance of this class formed from json_dict.

from_json(*json*)
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 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 Gists

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 Gists

gitignore_template(*language*)
Return the template for language.

Returns str

gitignore_templates()
Return the list of available templates.

Returns list of template names

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 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 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 Issue

issues(*filter=u''*, *state=u''*, *labels=u''*, *sort=u''*, *direction=u''*, *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 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 Issue

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 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 `Issues`

key(*id_num*)
Gets the authenticated user's key specified by id_num.

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

Returns *Key*

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 *Key*s

license(*name*)
Retrieve the license specified by the name.

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

Returns `License`

licenses(*number=-1*, *etag=None*)
Iterate over open source licenses.

Returns generator of `License`

login(*username=None*, *password=None*, *token=None*, *two_factor_callback=None*)
Logs 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=u''*, *context=u''*, *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 (or unicode on Python 2) – HTML formatted text

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.

meta()
Returns a dictionary with arrays of addresses in CIDR format specifying theaddresses that the incoming service hooks will originate from.

New in version 0.5.

notifications(*all=False*, *participating=False*, *number=-1*, *etag=None*)
Iterate over the user's notification.

Parameters

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

Returns generator of *Thread*

octocat(*say=None*)
Returns an easter egg of the API.

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

Returns ascii art of Octocat

Return type str (or unicode on Python 3)

organization(*username*)
Returns a Organization object for the login name

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

Returns *Organization*

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

Parameters

- **name** (*str*) – (required), name of the organization

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

Returns generator of `Issue`

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 `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 *Organization*s

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 *Organization*s

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 `Gists`

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

Parameters

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

Returns bool

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

Parameters

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

Returns `PullRequest`

rate_limit()
Returns 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 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 `Repository` objects

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 `Repository` objects

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

Parameters

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

Returns `Repository`

repository_with_id(*number*)
Returns the Repository with id `number`.

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

Returns `Repository`

revoke_authorization(**args*, ***kwargs*)
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 bool – True if successful, False otherwise

revoke_authorizations(**args*, ***kwargs*)
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 bool – True if successful, False otherwise

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 *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 qualifers:

- `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*

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 qualifers:

- `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_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 qualifers:

- `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 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*

set_client_id(*id*, *secret*)
Allows the developer to set their client_id and client_secret for their OAuth application.

Parameters

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

set_user_agent(*user_agent*)
Allows the user to set their own user agent string to identify with the API.

Parameters **user_agent** (*str*) – String used to identify your application. Library default: "github3.py/{version}", e.g., "github3.py/0.5"

star(*username*, *repo*)
Star to username/repo

Parameters

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

Returns bool

starred(*sort=None*, *direction=None*, *number=-1*, *etag=None*)
Iterate over repositories starred by the authenticated user.

Changed in version 1.0: 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 `Repository`

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 `Repository`

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 `Repository`

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`

unfollow(*username*)
Make the authenticated user stop following username

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

Returns bool

unstar(*username*, *repo*)
Unstar username/repo.

Parameters

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

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

Returns 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 whether the operation was successful or not

Return type bool

user(*username*)

Retrieve a User object for the specified user name.

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

Returns *User*

user_issues(*filter=u''*, *state=u''*, *labels=u''*, *sort=u''*, *direction=u''*, *since=None*, *per_page=None*, *number=-1*, *etag=None*)

List only the authenticated user's issues. Will not list organization's 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 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 `Issue`

user_teams(*number=-1*, *etag=None*)
Gets 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 *Team* objects

user_with_id(*number*)
Get the user's information with id `number`.

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

Returns *User*

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

Returns str (on Python 3, unicode on Python 2)

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=u''*, *password=u''*, *token=u''*, *verify=True*)
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*)
This is a simple way to get statistics about your system.

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

Returns dict

create_user(*login*, *email*)
Create a new user. 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 *User*, if successful

GitHubStatus Object

class `github3.github.`**GitHubStatus**
A sleek interface to the GitHub System Status API. This will only ever return the JSON objects returned by the API.

api()
GET /api.json

last_message()
GET /api/last-message.json

messages()
GET /api/messages.json

status()
GET /api/status.json

Issue

This part of the documentation covers the module which handles *Issue*s and their related objects:

- *IssueComment*
- *IssueEvent*
- *Milestone*
- *Label*.

Issue Objects

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

Two issue instances can be checked like so:

```
i1 == i2
i1 != i2
```

And is equivalent to:

```
i1.id == i2.id
i1.id != i2.id
```

add_labels(**args*)
Add labels to this issue.

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

Returns list of Labels

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*)
Assigns user `username` to this issue. This is a short cut for `issue.edit`.

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

Returns bool

close()
Close this issue.

Returns 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 *IssueComment*

comments(*number=-1*, *sort=u''*, *direction=u''*, *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 valuees: ('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 *IssueComment*s

create_comment(*body*)
Create a comment on this issue.

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

Returns *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[1], or 0 to remove the milestone
- **labels** (*list*) – list of labels to apply this to
- **assignees** (*list of strings*) – (optional), login of the users to assign the issue to

Returns 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 *IssueEvent*s

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

is_closed()
Checks if the issue is closed.

Returns 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 *Label*s

lock()
Lock an issue.

Returns bool

pull_request()
Retrieve the pull request associated with this issue.

Returns *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:

[1] Milestone numbering starts at 1, i.e. the first milestone you create is 1, the second is 2, etc.

```
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 an empty list if successful

remove_label(*name*)
Removes label `name` from this issue.

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

Returns list of `Label`

reopen()
Re-open a closed issue.

Returns bool

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

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

Returns list of `Label`

unlock()
Unlock an issue.

Returns bool

---

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

Two comment instances can be checked like so:

```
c1 == c2
c1 != c2
```

And is equivalent to:

```
c1.id == c2.id
c1.id != c2.id
```

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

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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.**IssueEvent**(*json*, *session=None*)

The *IssueEvent* object. This specifically deals with events described in the Issues>Events section of the GitHub API.

Two event instances can be checked like so:

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

And is equivalent to:

```
e1.commit_id == e2.commit_id
e1.commit_id != e2.commit_id
```

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.milestone.**Milestone**(*json*, *session=None*)

The *Milestone* object. This is a small class to handle information about milestones on repositories and issues.

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

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 bool

from_dict(*json_dict*)

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

from_json(*json*)

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 *Label*s

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 bool

---

class github3.issues.label.**Label**(*json*, *session=None*)
The *Label* object. Succintly represents a label that exists in a repository.

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

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 bool

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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 bool

Models

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

Objects

class `github3.models.`**`GitHubCore`**(*json*, *session=None*)

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*)

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

classmethod `from_json`(*json*)

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

`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.models.**BaseAccount**(*json*, *session=None*)

This class holds the commonalities of Organizations and Users.

The *BaseAccount* object is used to do the heavy lifting for *Organization* and *User* objects.

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.models.**BaseComment**(*json*, *session=None*)

A basic class for Gist, Issue and Pull Request Comments.

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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.models.**BaseCommit**(*json*, *session=None*)

This abstracts a lot of the common attributes for commit-like objects.

The *BaseCommit* object serves as the base for the various types of commit objects returned by the 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

from_dict(*json_dict*)

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

from_json(*json*)

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

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

Notifications

This part of the documentation covers the *Thread* and *Subscription* objects.

Notification Objects

class github3.notifications.**Thread**(*json*, *session=None*)

The *Thread* object wraps notification threads. This contains information about the repository generating the notification, the subject, and the reason.

Two thread instances can be checked like so:

```
t1 == t2
t1 != t2
```

And is equivalent to:

```
t1.id == t2.id
t1.id != t2.id
```

See also: http://developer.github.com/v3/activity/notifications/#view-a-single-thread

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 bool

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

is_unread()
Tells you if the thread is unread or not.

mark()
Mark the thread as read.

Returns 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

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 *Subscription*

subscription()
Checks the status of the user's subscription to this thread.

Returns *Subscription*

---

class github3.notifications.**Subscription**(*json*, *session=None*)
This object wraps thread and repository subscription information.

See also: developer.github.com/v3/activity/notifications/#get-a-thread-subscription

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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.

Organization

This section of the documentation covers:

- *Organization*s
- *Team*s

Organization Objects

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

Two organization instances can be checked like so:

```
o1 == o2
o1 != o2
```

And is equivalent to:

```
o1.id == o2.id
o1.id != o2.id
```

See also: http://developer.github.com/v3/orgs/

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 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 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 *Event*s

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 bool

create_repository(*name*, *description=u''*, *homepage=u''*, *private=False*, *has_issues=True*, *has_wiki=True*, *team_id=0*, *auto_init=False*, *gitignore_template=u''*, *license_template=u''*)

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.

Parameters

- **name** (*str*) – (required), name of the repository
- **description** (*str*) – (optional)
- **homepage** (*str*) – (optional)
- **private** (*bool*) – (optional), If `True`, create a private repository. API default: `False`
- **has_issues** (*bool*) – (optional), If `True`, enable issues for this repository. API default: `True`
- **has_wiki** (*bool*) – (optional), If `True`, enable the wiki for this repository. API default: `True`
- **team_id** (*int*) – (optional), id of the team that will be granted access to this repository
- **auto_init** (*bool*) – (optional), auto initialize the repository.
- **gitignore_template** (*str*) – (optional), name of the template; this is ignored if auto_int = False.
- **license_template** (*str*) – (optional), name of the license; this is ignored if auto_int = False.

Returns `Repository`

create_team(*name*, *repo_names=[]*, *permission=u''*)

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 *Team*

edit(*billing_email=None*, *company=None*, *email=None*, *location=None*, *name=None*)

Edit this organization.

Parameters

- **billing_email** (*str*) – (optional) Billing email address (private)
- **company** (*str*) – (optional)
- **email** (*str*) – (optional) Public email address
- **location** (*str*) – (optional)
- **name** (*str*) – (optional)

Returns bool

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

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 *Event*s

Deprecated: Use `public_events` instead.

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

from_json(*json*)
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 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 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 *User*s

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 *Event*s

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 *User*s

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 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 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)

Returns bool

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

Parameters

- **type** (*str*) – (optional), accepted values: ('all', 'public', 'member', 'private', 'forks', 'sources'), API default: 'all'
- **number** (*int*) – (optional), number of members to return. Default: -1 will return all available.
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of `Repository`

team(*team_id*)
Return the team specified by `team_id`.

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

Returns *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 *Team*s

---

class `github3.orgs.`**Team**(*json*, *session=None*)
The *Team* object.

Two team instances can be checked like so:

```
t1 == t2
t1 != t2
```

And is equivalent to:

```
t1.id == t2.id
t1.id != t2.id
```

See also: http://developer.github.com/v3/orgs/teams/

add_member(*username*)
Add `username` to this team.

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

Returns bool

add_repository(*repository*, *permission=u'‘*)
Add `repository` to this team.

Parameters

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

Returns 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 bool

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

Parameters

- **name** (*str*) – (required)
- **permission** (*str*) – (optional), (‘pull’, ‘push’, ‘admin’)

Returns bool

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

from_json(*json*)
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 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), user to invite to join this team.

Returns dictionary

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

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

Returns 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 *User*s

membership_for(*username*)
Retrieve the membership information for the user.

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

Returns dictionary

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 bool

remove_repository(*repository*)
Remove `repository` from this team.

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

Returns 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 `Repository` objects

revoke_membership(*username*)
Revoke this user's team membership.

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

Returns bool

Pull Request

This section of the documentation covers:

- *PullRequest*
- *ReviewComment*
- *PullDestination*
- *PullFile*

Pull Request Objects

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

Two pull request instances can be checked like so:

```
p1 == p2
p1 != p2
```

And is equivalent to:

```
p1.id == p2.id
p1.id != p2.id
```

See also: http://developer.github.com/v3/pulls/

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 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 *RepoCommit*s

create_comment(*body*)
Create a comment on this pull request's issue.

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

Returns *IssueComment*

create_review_comment(*body*, *commit_id*, *path*, *position*)
Create a review comment on this pull request.

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 bytestring representation of the diff.

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 *PullFile*s

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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

Returns bool

issue()
Retrieve the issue associated with this pull request.

Returns `Issue`

issue_comments(*number=-1*, *etag=None*)
Iterate over the issue 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 `IssueComments`

merge(*commit_message=None*, *sha=None*, *squash=False*)
Merge this pull request.

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.
- **squash** (*bool*) – (optional), commit a single commit to the head branch.

Returns bool

patch()
Return the patch.

Returns bytestring representation of the patch

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 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 *ReviewComment*s

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 `PullReviews`

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

Parameters

- **title** (*str*) – (optional), title of the pull
- **body** (*str*) – (optional), body of the pull request
- **state** (*str*) – (optional), ('open', 'closed')

Returns bool

class github3.pulls.**ReviewComment**(*json*, *session=None*)

The *ReviewComment* object.

This is used to represent comments on pull requests.

Two comment instances can be checked like so:

```
c1 == c2
c1 != c2
```

And is equivalent to:

```
c1.id == c2.id
c1.id != c2.id
```

See also: http://developer.github.com/v3/pulls/comments/

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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*

---

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

See also: http://developer.github.com/v3/pulls/#get-a-single-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

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

label = None
label of the destination

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

ref = None
Full reference string of the object

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

sha = None
SHA of the commit at the head

user = None
User representing the owner

---

class github3.pulls.**PullFile**(*json*, *session=None*)
The *PullFile* object.

See also: http://developer.github.com/v3/pulls/#list-pull-requests-files

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 *Contents*

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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

Repository

This part of the documentation covers:

- *Repository*
- *StarredRepository*
- *Asset*

- *Branch*
- *Contents*
- *Deployment*
- *DeploymentStatus*
- *Hook*
- *ImportedIssue*
- *PagesInfo*
- *PagesBuild*
- *Release*
- *RepoTag*
- *RepoComment*
- *RepoCommit*
- *Comparison*
- *Status*
- *CombinedStatus*
- *ContributorStats*

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

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

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

Repository Objects

class github3.repos.repo.**Repository**(*json*, *session=None*)

The *Repository* object.

It represents how GitHub sends information about repositories.

Two repository instances can be checked like so:

```
r1 == r2
r1 != r2
```

And is equivalent to:

```
r1.id == r2.id
r1.id != r2.id
```

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 bool – True if successful, False otherwise

archive(*format*, *path=u''*, *ref=u'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 bool – True if successful, False otherwise

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 *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 *User*s

blob(*sha*)
Get the blob indicated by `sha`.

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

Returns *Blob* if successful, otherwise None

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

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

Returns *Branch*

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 *Branch*es

code_frequency(*number=-1*, *etag=None*)
Iterate over the code frequency per week.

Returns a weekly aggregate of the number of additions and deletions pushed to this repository.

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]`

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.

New in version 0.7.

collaborators(*number=-1*, *etag=None*)
Iterate over the collaborators of this repository.

Parameters

- **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 `ShortUser`s

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 *RepoComment*s

commit(*sha*)
Get a single (repo) commit.

See *git_commit()* for the Git Data Commit.

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

Returns *RepoCommit* if successful, otherwise None

commit_activity(*number=-1*, *etag=None*)
Iterate over last year of commit activity by week.

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.

New in version 0.7.

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

commit_comment(*comment_id*)
Get a single commit comment.

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

Returns *RepoComment* if successful, otherwise None

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 string*) – (optional), Only commits after this date will be returned. This can be a `datetime` or an `ISO8601` formatted date string.
- **until** (*datetime or string*) – (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 *RepoCommit*s

compare_commits(*base*, *head*)
Compare two commits.

Parameters

- **base** (*str*) – (required), base for the comparison
- **head** (*str*) – (required), compare this against base

Returns *Comparison* if successful, else None

contributor_statistics(*number=-1*, *etag=None*)
Iterate over the contributors list.

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.

New in version 0.7.

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 *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 `ShortUsers`

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

Parameters

- **content** (*str*) – (required), content of the blob
- **encoding** (*str*) – (required), ('base64', 'utf-8')

Returns string of the SHA returned

create_comment(*body*, *sha*, *path=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 *RepoComment* if successful, otherwise None

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 ommitted, GitHub will use the author parameters. Should be the same format as the author parameter.

Returns *Commit* if successful, else None

create_deployment(*ref*, *required_contexts=None*, *payload=u''*, *auto_merge=False*, *description=u''*, *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 *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

{ 'content': *Contents*:, 'commit': *Commit*}

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

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

Returns *Repository* if successful, else None

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

Parameters

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

Returns *Hook* if successful, otherwise None

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

Returns *Issue* if successful, otherwise None

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 *Key* if successful, else None

create_label(*name*, *color*)
Create a label for this repository.

Parameters

- **name** (*str*) – (required), name to give to the label
- **color** (*str*) – (required), value of the color to assign to the label, e.g., '#fafafa' or 'fafafa' (the latter is what is sent)

Returns *Label* if successful, else None

create_milestone(*title*, *state=None*, *description=None*, *due_on=None*)
Create a milestone for this repository.

Parameters

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

Returns *Milestone* if successful, otherwise None

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 *PullRequest* if successful, else None

create_pull_from_issue(*issue*, *base*, *head*)
Create a pull request from issue #``issue``.

Parameters

- **issue** (*int*) – (required), issue number
- **base** (*str*) – (required), e.g., 'master'
- **head** (*str*) – (required), e.g., 'username:branch'

Returns *PullRequest* if successful, else None

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

Parameters

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

Returns *Reference* if successful else None

create_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 *Release*

create_status(*sha*, *state*, *target_url=None*, *description=None*, *context=u'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 status created if successful

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

delete()
Delete this repository.

Returns bool – True if successful, False otherwise

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

Returns bool – True if successful, False otherwise

delete_subscription()
Delete the user's subscription to this repository.

Returns bool

deployment(*id*)
Retrieve the deployment identified by `id`.

Parameters **id** (*int*) – (required), id for deployments.

Returns *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 *Deployment*s

directory_contents(*directory_path*, *ref=None*, *return_as=<type '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 list((str, *Contents*))

edit(*name*, *description=None*, *homepage=None*, *private=None*, *has_issues=None*, *has_wiki=None*, *has_downloads=None*, *default_branch=None*)
Edit this repository.

Parameters

- **name** (*str*) – (required), name of the repository
- **description** (*str*) – (optional), If not `None`, change the description for this repository. API default: `None` - leave value unchanged.
- **homepage** (*str*) – (optional), If not `None`, change the homepage for this repository. API default: `None` - leave value unchanged.
- **private** (*bool*) – (optional), If `True`, make the repository private. If `False`, make the repository public. API default: `None` - leave value unchanged.
- **has_issues** (*bool*) – (optional), If `True`, enable issues for this repository. If `False`, disable issues for this repository. API default: `None` - leave value unchanged.
- **has_wiki** (*bool*) – (optional), If `True`, enable the wiki for this repository. If `False`, disable the wiki for this repository. API default: `None` - leave value unchanged.
- **has_downloads** (*bool*) – (optional), If `True`, enable downloads for this repository. If `False`, disable downloads for this repository. API default: `None` - leave value unchanged.
- **default_branch** (*str*) – (optional), If not `None`, change the default branch for this repository. API default: `None` - leave value unchanged.

Returns bool – True if successful, False otherwise

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 *Event*s

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=u''*, *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 *Repository*

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

from_json(*json*)
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 *Commit* if successful, otherwise None

hook(*hook_id*)
Get a single hook.

Parameters **hook_id** (*int*) – (required), id of the hook

Returns *Hook* if successful, otherwise None

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 *Hook*s

ignore()
Ignore notifications from this repository for the user.

New in version 1.0.

This replaces `Repository#set_subscription`.

Returns *Subscription*

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** (*timestamp*) – (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 `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 `Imported Issue`

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 `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 bool – True if successful, False otherwise

issue(*number*)
Get the issue specified by `number`.

Parameters **number** (*int*) – (required), number of the issue on this repository

Returns *Issue* if successful, otherwise None

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 *IssueEvent*s

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 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. By default all issues are returned
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of *Issue*s

key(*id_num*)

Get the specified deploy key.

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

Returns *Key* if successful, else None

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 *Key*s

label(*name*)

Get the label specified by `name`.

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

Returns *Label* if successful, else None

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 *Label*s

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

latest_pages_build()
Get the build information for the most recent Pages build.

Returns *PagesBuild*

latest_release()
Get the latest release.

Draft releases and prereleases are not returned by this endpoint.

Returns *Release*

license()
Get the contents of a license for the repo

Returns License

mark_notifications(*last_read=u''*)
Mark all notifications in this repository as read.

Parameters **last_read** (*str*) – (optional), Describes the last point that notifications were checked. Anything updated since this time will not be updated. Default: Now. Expected in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ. Example: "2012-10-09T23:39:01Z".

Returns bool

merge(*base*, *head*, *message=u''*)
Perform a merge from head into base.

Parameters

- **base** (*str*) – (required), where you're merging into
- **head** (*str*) – (required), where you're merging from
- **message** (*str*) – (optional), message to be used for the commit

Returns *RepoCommit*

milestone(*number*)
Get the milestone indicated by number.

Parameters **number** (*int*) – (required), unique id number of the milestone

Returns *Milestone*

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 *Milestone*s

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 *Event*s

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 string*) – (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 *Thread*

pages()
Get information about this repository's pages site.

Returns *PagesInfo*

pages_builds(*number=-1*, *etag=None*)
Iterate over pages builds of this repository.

Returns generator of *PagesBuild*

pull_request(*number*)
Get the pull request indicated by `number`.

Parameters **number** (*int*) – (required), number of the pull request.

Returns *PullRequest*

pull_requests(*state=None*, *head=None*, *base=None*, *sort=u'created'*, *direction=u'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 *PullRequest*s

ratelimit_remaining

Number of requests before GitHub imposes a ratelimit.

Returns int

readme()

Get the README for this repository.

Returns *Contents*

ref(*ref*)

Get a reference pointed to by `ref`.

The most common will be branches and tags. For a branch, you must specify 'heads/branchname' and for a tag, 'tags/tagname'. Essentially, the system should return any reference you provide it in the namespace, including notes and stashes (provided they exist on the server).

Parameters **ref** (*str*) – (required)

Returns *Reference*

refresh(*conditional=False*)

Re-retrieve the information for this object.

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=u''*, *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 *Reference*s

release(*id*)
Get a single release.

Parameters **id** (*int*) – (required), id of release

Returns *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 *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 *Release*s

remove_collaborator(*username*)
Remove collaborator `username` from the repository.

Parameters **username** (str or *User*) – (required), login name of the collaborator

Returns 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 ShortUsers

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 *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 *Subscription*

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 ShortUser

subscription()
Return subscription for this Repository.

Returns *Subscription*

tag(*sha*)
Get an annotated tag.

http://learn.github.com/p/tagging.html

Parameters **sha** (*str*) – (required), sha of the object for this tag

Returns *Tag*

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 *RepoTag*s

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 *Team*s

tree(*sha*)
Get a tree.

Parameters **sha** (*str*) – (required), sha of the object for this tree

Returns *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 emtpy 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 dict

---

class github3.repos.repo.**StarredRepository**(*json*, *session=None*)
The *StarredRepository* object.

It represents how GitHub sends back a repository a user has starred, e.g., from *starred_repositories()*.

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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.branch.**Branch**(*json*, *session=None*)
The *Branch* object. It holds the information GitHub returns about a branch on a *Repository*.

latest_sha(*differs_from=u''*)
Check if SHA-1 is the same as remote branch

See: https://git.io/vaqIw

Parameters **differs_from** (*str*) – (optional), sha to compare against

Returns string of the SHA or None

protect(*enforcement=None*, *status_checks=None*)
Enable force push protection and configure status check enforcement.

See: http://git.io/v4Gvu

Parameters

- **enforcement** (*str*) – (optional), Specifies the enforcement level of the status checks. Must be one of 'off', 'non_admins', or 'everyone'. Use *None* or omit to use the already associated value.
- **status_checks** (*list*) – (optional), An list of strings naming status checks that must pass before merging. Use *None* or omit to use the already associated value.

unprotect()
Disable force push protection on this branch.

---

class github3.repos.contents.**Contents**(*json*, *session=None*)
The *Contents* object. It holds the information concerning any content in a repository requested via the API.

Two content instances can be checked like so:

```
c1 == c2
c1 != c2
```

And is equivalent to:

```
c1.sha == c2.sha
c1.sha != c2.sha
```

See also: http://developer.github.com/v3/repos/contents/

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.deployment.**Deployment**(*json*, *session=None*)

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 partial *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 *DeploymentStatus*es

---

class github3.repos.deployment.**DeploymentStatus**(*json*, *session=None*)

---

class github3.repos.release.**Release**(*json*, *session=None*)
The *Release* object.

It holds the information GitHub returns about a release from a *Repository*.

archive(*format*, *path=u''*)
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 bool – True if successful, False otherwise

asset(*asset_id*)
Retrieve the asset from this release with `asset_id`.

Parameters **asset_id** (*int*) – ID of the Asset to retrieve

Returns *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

delete()
Users with push access to the repository can delete a release.

Returns True if successful; False if not successful

edit(*tag_name=None*, *target_commitish=None*, *name=None*, *body=None*, *draft=None*, *prerelease=None*)
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

upload_asset(*content_type*, *name*, *asset*, *label=None*)
Upload an asset to this release.

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 *Asset*

---

class github3.repos.release.**Asset**(*json*, *session=None*)

delete()
Delete this asset if the user has push access.

Returns True if successful; False if not successful

Return type boolean

download(*path=u''*)
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 boolean

---

class github3.repos.hook.**Hook**(*json*, *session=None*)
The *Hook* object. This handles the information returned by GitHub about hooks set on a repository.

Two hook instances can be checked like so:

```
h1 == h2
h1 != h2
```

And is equivalent to:

```
h1.id == h2.id
h1.id != h2.id
```

See also: http://developer.github.com/v3/repos/hooks/

delete()
Delete this hook.

Returns 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 bool

ping()
Ping this hook.

Returns bool

test()
Test this hook

Returns bool

---

class github3.repos.issue_import.**ImportedIssue**(*json*, *session=None*)
The ImportedIssue object. This represents information from the Import Issue API.

See also: https://gist.github.com/jonmagic/5282384165e0f86ef105

---

class github3.repos.pages.**PagesInfo**(*json*, *session=None*)

---

class github3.repos.pages.**PagesBuild**(*json*, *session=None*)

---

class github3.repos.tag.**RepoTag**(*json*, *session=None*)
The *RepoTag* object. This stores the information representing a tag that was created on a repository.

See also: http://developer.github.com/v3/repos/#list-tags

More information about this class can be found in the official documentation about comments.

class github3.repos.comment.**RepoComment**(*json*, *session=None*)
The *RepoComment* object. This stores the information about a comment on a file in a repository.

Two comment instances can be checked like so:

```
c1 == c2
c1 != c2
```

And is equivalent to:

```
c1.id == c2.id
c1.id != c2.id
```

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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(*body*)
Update this comment.

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

Returns bool

class github3.repos.commit.**RepoCommit**(*json*, *session=None*)

The *RepoCommit* object. This represents a commit as viewed by a Repository. This is different from a Commit object returned from the git data section.

Two commit instances can be checked like so:

```
c1 == c2
c1 != c2
```

And is equivalent to:

```
c1.sha == c2.sha
c1.sha != c2.sha
```

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

Iterate over comments for this commit.

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 *RepoComment*s

diff()

Retrieve the diff for this commit.

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

status()

Retrieve the combined status for this commit.

Returns the combined status for this commit

Return type *CombinedStatus*

statuses()

Retrieve the statuses for this commit.

Returns the statuses for this commit

Return type *Status*

---

class github3.repos.comparison.**Comparison**(*json*, *session=None*)

The *Comparison* object. This encapsulates the information returned by GitHub comparing two commit objects in a repository.

Two comparison instances can be checked like so:

```
c1 == c2
c1 != c2
```

And is equivalent to:

```
c1.commits == c2.commits
c1.commits != c2.commits
```

See also: http://developer.github.com/v3/repos/commits/#compare-two-commits

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

---

class github3.repos.status.**Status**(*json*, *session=None*)
The *Status* object.

This represents information from the Repo Status API.

See also: http://developer.github.com/v3/repos/statuses/

---

class github3.repos.status.**CombinedStatus**(*json*, *session=None*)
The *CombinedStatus* object.

This represents combined information from the Repo Status API.

See also: http://developer.github.com/v3/repos/statuses/

---

class github3.repos.stats.**ContributorStats**(*json*, *session=None*)
This object provides easy access to information returned by the statistics section of the API.

See http://developer.github.com/v3/repos/statistics/ for specifics.

Search Structures

These classes are meant to expose the entirety of an item returned as a search result by GitHub's Search API.

Objects

class github3.search.**CodeSearchResult**(*json*, *session=None*)

class github3.search.**IssueSearchResult**(*json*, *session=None*)

class github3.search.**RepositorySearchResult**(*json*, *session=None*)

class github3.search.**UserSearchResult**(*json*, *session=None*)

Structures

Developed for github3.py

As of right now, there exists only one class in this section, and it is of only limited importance to users of github3.py. The *GitHubIterator* class is used to return the results of calls to almost all of the calls to `iter_` methods on objects. When conditional refreshing was added to objects, there was a noticable gap in having conditional calls to those `iter_` methods. GitHub provides the proper headers on those calls, but there was no easy way to add that to what github3.py returned so it could be used properly. This was the best compromise - an object the behaves like an iterator regardless but can also be `refresh`ed to get results since the last request conditionally.

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

from_dict(*json_dict*)

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

from_json(*json*)

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

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

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

items = None
Items array returned in the last request

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

total_count = None
Total count returned by GitHub

User

This part of the documentation covers:

- *User*
- *Key*
- *Plan*

None of these objects should ever be instantiated by the user (developer).

When listing users, GitHub only sends a handful of the object's attributes. To retrieve all of the object's attributes, you must call the refresh() method. This unfortunately requires another call to the API, so use it sparingly if you have a low limit

User Modules

class github3.users.**User**(*json*, *session=None*)
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.

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 *Event*s

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 *User*s

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 *User*s

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

from_json(*json*)
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 *Key*s

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 *Event*s

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 *Event*s

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 *Event*s

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 dos 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 dos for more information.

Returns bool – True if successful, False otherwise

class github3.users.**Key**(*json*, *session=None*)
The *Key* object.

Please see GitHub's Key Documentation for more information.

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.

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

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=None*)
The *Plan* object.

Please see GitHub's Authenticated User documentation for more details.

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

from_dict(*json_dict*)
Return an instance of this class formed from `json_dict`.

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

is_free()
Check if this is a free plan.

Returns 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

Internals

For objects you're not likely to see in practice. This is useful if you ever feel the need to contribute to the project.

Decorators

This part of the documentation covers the decorators module which contains all of the decorators used in github3.py.

Contents

github3.decorators.**requires_auth**(*x*)

CHAPTER 3

Installation

```
$ pip install github3.py
# OR:
$ git clone git://github.com/sigmavirus24/github3.py.git github3.py
$ cd github3.py
$ python setup.py install
```

Dependencies

- requests by Kenneth Reitz
- uritemplate by Ian Cordasco

CHAPTER 4

Contributing

I'm maintaining two public copies of the project. The first can be found on GitHub and the second on BitBucket. I would prefer pull requests to take place on GitHub, but feel free to do them via BitBucket. Please make sure to add yourself to the list of contributors in AUTHORS.rst, especially if you're going to be working on the list below.

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.

Running the Unittests

The tests are generally run using tox. Tox can be installed lke so:

```
pip install tox
```

We test against PyPy and the following versions of Python:

- 2.6
- 2.7
- 3.2
- 3.3
- 3.4

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 py34
```

And if you want to run tests against a specific file, you can do:

```
tox -e py34 -- tests/uni/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 py34 -- -vv
```

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 5

Contact

- Twitter: @sigmavirus24
- Private email: graffatcolmingov [at] gmail
- Mailing list: github3.py [at] librelist.com

CHAPTER 6

Latest Version's Changes

Unreleased

- Add `Organization#all_events`.
- Deprecate `Organization#events` in favor of `Organization#public_events`.
- Fix test failtures on windows caused by unclosed file handles.
- Add `Tag.tagger_as_User` which attempts to return the tagger as as User.
- Add `Repo.statuses` and a corresponding `repo.status.CombinedStatus` to get a combined view of commit statuses for a given ref.

1.0.0a4: 2016-02-19

Features Added (since 1.0.0a3)

- Add support for the Issue locking API currently in Preview Mode

Bugs Fixed (since 1.0.0a3)

- 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.

1.0.0a3: 2016-02-11

Breaking Changes (since 1.0.0a2)

- 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`.

Features Added (since 1.0.0a2)

- 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

Bugs Fixed (since 1.0.0a2)

- 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`.

1.0.0a2: 2015-07-14

Breaking Changes (since 1.0.0a1)

- 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`.

Features Added (since 1.0.0a1)

- 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 (since 1.0.0a1)

- 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.

1.0.0a1: 2014-12-07

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 *very* carefully.

Breaking Changes

- `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
github3.iter_all_repos	github3.all_repositories
github3.iter_all_users	github3.all_users
github3.iter_events	github3.all_events
github3.iter_followers	github3.followers_of
github3.iter_following	github3.followed_by
github3.iter_repo_issues	github3.issues_on
github3.iter_orgs	github3.organizations_with
github3.iter_user_repos	github3.repositories_by
github3.iter_starred	github3.starred_by
github3.iter_subscriptions	github3.subscriptions_for
Deployment#iter_statuses	Deployment#statuses
Gist#iter_comments	Gist#comments
Gist#iter_commits	Gist#commits
Gist#iter_files	Gist#files
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#reposistories
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

Continued on next page

Table 6.1 – continued from previous page

Old name	New name
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
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()
```

The full history of the project is available as well.

Changelog

Unreleased

- Add `Organization#all_events`.
- Deprecate `Organization#events` in favor of `Organization#public_events`.
- Fix test failtures on windows caused by unclosed file handles.
- Add `Tag.tagger_as_User` which attempts to return the tagger as as User.
- Add `Repo.statuses` and a corresponding `repo.status.CombinedStatus` to get a combined view of commit statuses for a given ref.

1.0.0a4: 2016-02-19

Features Added (since 1.0.0a3)

- Add support for the Issue locking API currently in Preview Mode

Bugs Fixed (since 1.0.0a3)

- 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.

1.0.0a3: 2016-02-11

Breaking Changes (since 1.0.0a2)

- 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`.

Features Added (since 1.0.0a2)

- 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

Bugs Fixed (since 1.0.0a2)

- 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`.

1.0.0a2: 2015-07-14

Breaking Changes (since 1.0.0a1)

- 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`.

Features Added (since 1.0.0a1)

- 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 (since 1.0.0a1)

- 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.

1.0.0a1: 2014-12-07

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 *very* carefully.

Breaking Changes

- `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
`github3.iter_all_repos`	`github3.all_repositories`
`github3.iter_all_users`	`github3.all_users`
`github3.iter_events`	`github3.all_events`
`github3.iter_followers`	`github3.followers_of`
`github3.iter_following`	`github3.followed_by`
`github3.iter_repo_issues`	`github3.issues_on`
`github3.iter_orgs`	`github3.organizations_with`
`github3.iter_user_repos`	`github3.repositories_by`
`github3.iter_starred`	`github3.starred_by`
`github3.iter_subscriptions`	`github3.subscriptions_for`
`Deployment#iter_statuses`	`Deployment#statuses`
`Gist#iter_comments`	`Gist#comments`
`Gist#iter_commits`	`Gist#commits`
`Gist#iter_files`	`Gist#files`
`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#reposistories`
`GitHub#iter_user_repos`	`GitHub#repositories_by`
`GitHub#iter_user_teams`	`GitHub#user_teams`
`Issue#iter_comments`	`Issue#comments`
`Issue#iter_events`	`Issue#events`

Continued on next page

Table 6.2 – continued from previous page

Old name	New name
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
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

Continued on next page

Table 6.2 – continued from previous page

Old name	New name
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()
```

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 stargarzers_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 >= 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
- `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: 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 retuns 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: 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: 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

r = github3.repository('sigmavirus24', 'github3.py')

template = """Some kind of template where you mention this repository
{0}"""

print(template.format(r))
# 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: 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: 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

CHAPTER 7

Testimonials

Python Module Index

g

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

Z