github3.py Documentation

Release 0.9.6

Ian Cordasco

September 24, 2016

Contents

Release v0.9.6.

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.user()
# <User [sigmavirus24:Ian Cordasco]>

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

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

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

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

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

1.1 More Examples

1.1.1 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 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.

1.1.2 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.

1.1.3 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.

1.1.4 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'
```

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

1.1.6 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
```

1.1.7 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.

1.1.8 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.

1.1.9 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

2.1 API

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

2.1.1 Module Contents

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

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

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

Construct and return an authenticated GitHub session.

This will return a GitHubEnterprise session if a url is provided.

Parameters

- **`username`** (*`str`*) – login name
- **`password`** (*`str`*) – password for the login
- **`token`** (*`str`*) – OAuth token
- **`url`** (*`str`*) – (optional), URL of a GitHub Enterprise instance
- **`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**(*login*, *password*, *scopes*, *note=''*, *note_url=''*, *client_id=''*, *client_secret=''*, *two_factor_callback=None*)

Obtain an authorization token for the GitHub API.

Parameters

- **login** (*str*) – (required)
- **password** (*str*) – (required)
- **scopes** (*list*) – (required), areas you want this token to apply to, i.e., 'gist', 'user'
- **note** (*str*) – (optional), note about the authorization
- **note_url** (*str*) – (optional), url for the application
- **client_id** (*str*) – (optional), 20 character OAuth client key for which to create a token
- **client_secret** (*str*) – (optional), 40 character OAuth client secret for which to create the token
- **two_factor_callback** (*func*) – (optional), function to call when a Two-Factor Authentication code needs to be provided by the user.

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

Parameters

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

Returns generator of Repository

---

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

Parameters

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

Returns generator of *User*

---

github3.**iter_events**(*number=-1*, *etag=None*)
Iterate over public events.

Parameters

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

Returns generator of *Event*

---

github3.**iter_followers**(*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.**iter_following**(*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.**iter_gists**(*username=None*, *number=-1*, *etag=None*)
Iterate over public gists or gists for the provided username.

Parameters

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

Returns generator of `Gist`

---

github3.**iter_orgs**(*username*, *number=-1*, *etag=None*)
List the organizations associated with `username`.

Parameters

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

Returns generator of *Organization*

---

github3.**iter_user_repos**(*login*, *type=None*, *sort=None*, *direction=None*, *number=-1*, *etag=None*)
List public repositories for the specified `login`.

New in version 0.6.

Note: This replaces github3.iter_repos

Parameters

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

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

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

Iterate over repositories starred by `username`.

Parameters

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

Returns generator of `Repository`

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

Iterate over repositories subscribed to by `username`.

Parameters

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

Returns generator of `Repository`

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

Render an arbitrary markdown document.

Parameters

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

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`**(*login*)

Returns a Organization object for the login name

Parameters **login** (*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.`**`ratelimit_remaining`**()

Get the remaining number of requests allowed.

Returns int

---

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`**(*login*)

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

Parameters **login** (*str*) – (optional)

Returns *User*

`github3.`**`zen`**()

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

Returns str

2.1.2 Enterprise Use

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

```
from github import GitHubEnterprise

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

2.2 Authorization

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

class github3.auths.**Authorization**(*auth*, *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

app = None

Details about the application (name, url)

created_at = None

datetime object representing when the authorization was created.

delete()

delete this authorization

from_json(*json*)

Return an instance of `cls` formed from `json`.

id = None

Unique id of the authorization

name = None

App name

note = None

Note about the authorization

note_url = None

URL about the note

ratelimit_remaining

Number of requests before GitHub imposes a ratelimit.

Returns int

refresh(*conditional=False*)

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

scopes = None
List of scopes this applies to

to_json()
Return the json representing this object.

token = None
Returns the Authorization token

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

Parameters

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

Returns bool

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

2.3 Events

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

2.3.1 Event Objects

class github3.events.**Event**(*event*, *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
```

actor = None
User object representing the actor.

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

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

id = None
Unique id of the event

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

> **Warning:** This will be deprecated in 0.6

Returns bool – True if event is pubic, False otherwise

static **list_types**()
List available payload types

org = None
List all possible types of Events

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

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

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

repo = None
Return `tuple(owner, repository_name)`

to_json()
Return the json representing this object.

type = None
Event type http://developer.github.com/v3/activity/events/types/

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.

2.4 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.

2.4.1 Gist Objects

class github3.gists.gist.**Gist**(*data*, *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/

comments = None
Number of comments on this gist

comments_url = None
Comments URL (not a template)

commits_url = None
Commits URL (not a template)

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

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

Returns *GistComment*

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

delete()
Delete this gist.

Returns bool – whether the deletion was successful

description = None
Description of the gist

edit(*description=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 = None
Number of files in this gist.

fork()
Fork this gist.

Returns *Gist* if successful, None otherwise

forks = None
The number of forks of this gist.

forks_url = None
Forks URL (not a template)

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

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

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

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

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

id = None
Unique id for this gist.

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

Returns bool – True if public, False if private

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

Returns bool – True if it is starred, False otherwise

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

Parameters

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

Returns generator of *GistComment*

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

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

New in version 0.6.

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*

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

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

iter_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*

owner = None
User object representing the owner of the gist.

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

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

star()
Star this gist.

Returns bool – True if successful, False otherwise

to_json()
Return the json representing this object.

truncated = None
Whether the content of this Gist has been truncated or not

unstar()
Un-star this gist.

Returns bool – True if successful, False otherwise

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

---

class github3.gists.comment.**GistComment**(*comment*, *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/

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_json(*json*)
Return an instance of `cls` 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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

user = None
User who made the comment Unless it is not associated with an account

---

class github3.gists.file.**GistFile**(*attributes*)
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.

content = None
The content of the file.

filename = None
The name of the file.

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

language = None
The language associated with the file.

name = None
The name of the file.

raw_url = None
The raw URL for the file at GitHub.

size = None
The size of the file.

to_json()
Return the json representing this object.

---

class github3.gists.history.**GistHistory**(*history*, *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
```

additions = None
number of additions made

change_status = None
dict containing the change status; see also: deletions, additions, total

committed_at = None
datetime representation of when the commit was made

deletions = None
number of deletions made

from_json(*json*)
Return an instance of `cls` 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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

total = None
total number of changes made

user = None
user who made these changes

version = None
SHA of the commit associated with this version

2.5 Git

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

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

2.5.1 Git Objects

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

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

content = None
Raw content of the blob.

decoded = None
Decoded content of the blob.

encoding = None
Encoding of the raw content.

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

sha = None
SHA1 of the blob

size = None
Size of the blob in bytes

to_json()
Return the json representing this object.

---

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

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

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

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

committer = None
dict containing similar information to the author attribute

committer_as_User()
Attempt to return the committer attribute as a *User* object. No guarantees are made about the validity of this object.

from_json(*json*)
Return an instance of `cls` 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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

tree = None
Tree the commit belongs to.

---

class github3.git.**GitData**(*data*, *session=None*)

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

from_json(*json*)

Return an instance of `cls` 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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

sha = None

SHA of the object

to_json()

Return the json representing this object.

---

class github3.git.**GitObject**(*obj*)

The *GitObject* object.

from_json(*json*)

Return an instance of `cls` 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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

type = None
The type of object.

---

class github3.git.**Hash**(*info*)
The *Hash* object.

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

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

mode = None
File mode

path = None
Path to file

sha = None
SHA of the hash

size = None
Size of hash

to_json()
Return the json representing this object.

type = None
Type of hash, e.g., blob

url = None
URL of this object in the GitHub API

---

class github3.git.**Reference**(*ref*, *session=None*)
The *Reference* object. This represents a reference created on a repository.

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

delete()
Delete this reference.

Returns bool

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

object = None
GitObject the reference points to

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

update(*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**(*tag*)
The *Tag* object.

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

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

message = None
Commit message for the tag

object = None
GitObject for the tag

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

tag = None
String of the tag

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

to_json()
Return the json representing this object.

---

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

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

from_json(*json*)
Return an instance of `cls` 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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

tree = None
list of *Hash* objects

2.6 GitHub

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

2.6.1 Examples

Examples utilizing this object can be found here.

2.6.2 GitHub Object

class github3.github.**GitHub**(*login=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.

authorization(*id_num*)
Get information about authorization id.

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

Returns Authorization

authorize(*login*, *password*, *scopes=None*, *note=u''*, *note_url=u''*, *client_id=u''*, *client_secret=u''*)
Obtain an authorization token from the GitHub API for the GitHub API.

Parameters

- **login** (*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*)
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.

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, else None

create_key(*title*, *key*)
Create a new key for the authenticated user.

Parameters

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

Returns *Key*

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

Returns `Repository`

delete_key(*key_id*)
Delete user key pointed to by `key_id`.

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

Returns bool

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(*login*)
Make the authenticated user follow login.

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

Returns bool

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

gist(*id_num*)
Gets the gist using the specified id number.

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

Returns Gist

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

Returns str

gitignore_templates()
Returns the list of available templates.

Returns list of template names

is_following(*login*)
Check if the authenticated user is following login.

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

Returns bool

is_starred(*login*, *repo*)
Check if the authenticated user starred login/repo.

Parameters

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

Returns bool

is_subscribed(*login*, *repo*)
Check if the authenticated user is subscribed to login/repo.

Parameters

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

Returns bool

issue(*owner*, *repository*, *number*)
Fetch issue #:number: from https://github.com/:owner:/:repository:

Parameters

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

Returns Issue

iter_all_repos(*number=-1*, *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`

iter_all_users(*number=-1*, *etag=None*, *per_page=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
- **per_page** (*int*) – (optional), number of users to list per request

Returns generator of *User*

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

Parameters

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

Returns generator of `Authorizations`

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

Parameters

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

Returns generator of dicts

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

Parameters

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

Returns generator of *Event*s

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

Parameters

- **login** (*str*) – (optional), login of the user to check
- **number** (*int*) – (optional), number of followers to return. Default: -1 returns all followers

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

Returns generator of *User*s

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

Parameters

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

Returns generator of *User*s

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

Parameters

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

Returns generator of `Gists`

iter_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`

iter_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

iter_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*

iter_org_issues(*name*, *filter=u''*, *state=u''*, *labels=u''*, *sort=u''*, *direction=u''*, *since=None*, *number=-1*, *etag=None*)
Iterate over the organnization'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`

iter_orgs(*login=None*, *number=-1*, *etag=None*)
Iterate over public organizations for login if provided; otherwise iterate over public and private organizations for the authenticated user.

Parameters

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

Returns generator of *Organization*s

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

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

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`

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

List public repositories for the authenticated user.

Changed in version 0.6: Removed the login parameter for correctness. Use iter_user_repos 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

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

Iterate over repositories starred by `login` or the authenticated user.

Changed in version 0.5: Added sort and direction parameters (optional) as per the change in GitHub's API.

Parameters

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

Returns generator of `Repository`

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

Parameters

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

Returns generator of `Repository`

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

iter_user_repos(*login*, *type=None*, *sort=None*, *direction=None*, *number=-1*, *etag=None*)
List public repositories for the specified `login`.

New in version 0.6.

Parameters

- **login** (*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

iter_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

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

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

Returns *Key*

login(*username=None*, *password=None*, *token=None*, *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 – HTML formatted text

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.

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

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

Returns ascii art of Octocat

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

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

Returns *Organization*

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

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 Issue

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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

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

Parameters

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

Returns `Repository`

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(*login*, *repo*)
Star to login/repo

Parameters

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

Returns bool

subscribe(*login*, *repo*)
Subscribe to login/repo

Parameters

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

Returns bool

to_json()
Return the json representing this object.

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

Parameters **login** (*str*) – (required)

Returns bool

unstar(*login*, *repo*)
Unstar to login/repo

Parameters

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

Returns bool

unsubscribe(*login*, *repo*)
Unsubscribe to login/repo

Parameters

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

Returns bool

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

Parameters

- **name** (*str*) – e.g., 'John Smith', not login name
- **email** (*str*) – e.g., 'john.smith@example.com'
- **blog** (*str*) – e.g., 'http://www.example.com/jsmith/blog'

- **company** (*str*) – company name
- **location** (*str*) – where you are located
- **hireable** (*bool*) – defaults to False
- **bio** (*str*) – GitHub flavored markdown

Returns bool

user(*login=None*)
Returns a User object for the specified login name if provided. If no login name is provided, this will return a User object for the authenticated user.

Parameters **login** (*str*) – (optional)

Returns *User*

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

Returns str

2.6.3 GitHubEnterprise Object

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

class github3.github.**GitHubEnterprise**(*url*, *login=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

2.6.4 GitHubStatus Object

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

api()
GET /api.json

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

messages()
GET /api/messages.json

status()
GET /api/status.json

2.7 Issue

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

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

2.7.1 Issue Objects

class github3.issues.issue.**Issue**(*issue*, *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

assign(*login*)

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

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

Returns bool

assignee = None

User representing the user the issue was assigned to.

body = None

Body (description) of the issue.

body_html = None

HTML formatted body of the issue.

body_text = None

Plain text formatted body of the issue.

close()

Close this issue.

Returns bool

closed_at = None

datetime object representing when the issue was closed.

closed_by = None

User who closed the issue.

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

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

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

Returns *IssueComment*

comments = None
Number of comments on this issue.

comments_url = None
Comments url (not a template)

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

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

Returns *IssueComment*

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

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

Parameters

- **title** (*str*) – Title of the issue
- **body** (*str*) – markdown formatted body (description) of the issue
- **assignee** (*str*) – login name of user the issue should be assigned to
- **state** (*str*) – accepted values: ('open', 'closed')
- **milestone** (*int*) – the NUMBER (not title) of the milestone to assign this to [1], or 0 to remove the milestone
- **labels** (*list*) – list of labels to apply this to

Returns bool

events_url = None
Events url (not a template)

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

html_url = None
URL to view the issue at GitHub.

id = None
Unique ID for the issue.

is_closed()
Checks if the issue is closed.

Returns bool

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

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

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

Returns iterator of *IssueComment*s

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

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

Returns generator of *IssueEvent*s

iter_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

labels = None
Returns the list of *Label*s on this issue.

labels_urlt = None
Labels URL Template. Expand with `name`

milestone = None
Milestone this issue was assigned to.

number = None
Issue number (e.g. #15)

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

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

remove_all_labels()
Remove all labels from this issue.

Returns an empty list if successful

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

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

Returns bool

reopen()
Re-open a closed issue.

Returns bool

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

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

Returns bool

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

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

title = None
Title of the issue.

to_json()
Return the json representing this object.

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

user = None
User who opened the issue.

---

class github3.issues.comment.**IssueComment**(*comment*, *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/

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

issue_url = None
Issue url (not a template)

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

user = None
User who made the comment

---

class github3.issues.event.**IssueEvent**(*event*, *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
```

actor = None
User that generated the event.

comments = None
Number of comments

commit_id = None
SHA of the commit.

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

event = None
The type of event, e.g., closed

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

issue = None
`Issue` where this comment was made.

pull_request = None
Dictionary of links for the pull request

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

---

class github3.issues.milestone.**Milestone** (*mile*, *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/

closed_issues = None
The number of closed issues associated with this milestone.

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

creator = None
User object representing the creator of the milestone.

delete()
Delete this milestone.

Returns bool

description = None
Description of this milestone.

due_on = None
datetime representing when this milestone is due.

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

iter_labels(*number=-1*, *etag=None*)
Iterate over the labels for every issue associated with this milestone.

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

number = None
Identifying number associated with milestone.

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

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

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

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

to_json()
Return the json representing this object.

update(*title=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

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

---

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

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

color = None
Color of the label, e.g., 626262

delete()
Delete this label.

Returns bool

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

name = None
Name of the label, e.g., 'bug'

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

update(*name*, *color*)
Update this label.

Parameters

- **name** (*str*) – (required), new name of the label
- **color** (*str*) – (required), color code, e.g., 626262, no leading '#'

Returns bool

2.8 Models

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

2.8.1 Objects

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

from_json(*json*)
Return an instance of `cls` 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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

class github3.models.**BaseAccount**(*acct*, *session*)

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

avatar_url = None
URL of the avatar at gravatar

bio = None
Markdown formatted biography

blog = None
URL of the blog

company = None
Name of the company

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

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

followers = None
Number of followers

following = None
Number of people the user is following

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

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

id = None
Unique ID of the account

location = None
Location of the user/org

login = None
login name of the user/org

name = None
Real name of the user/org

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

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

type = None
Tells you what type of account this is

---

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

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

body_html = None
Body of the comment formatted as html.

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

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

delete()
Delete this comment.

Returns bool

edit(*body*)
Edit this comment.

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

Returns bool

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

html_url = None
The url of this comment at GitHub

id = None
Unique ID of the comment.

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

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json ()
Return the json representing this object.

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

---

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

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

html_url = None
URL to view the commit on GitHub

message = None
Commit message

parents = None
List of parents to this commit.

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

sha = None
SHA of this commit.

to_json()
Return the json representing this object.

2.9 Notifications

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

2.9.1 Notification Objects

class github3.notifications.**Thread**(*notif*, *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

comment = None
Comment responsible for the notification

delete_subscription()
Delete subscription for this thread.

Returns bool

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

id = None
Id of the thread

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

last_read_at = None
datetime object representing the last time the user read the thread

mark()
Mark the thread as read.

Returns bool

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

reason = None
The reason you're receiving the notification

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

repository = None
Repository the comment was made on

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*

subject = None
Subject of the Notification, e.g., which issue/pull/diff is this in relation to. This is a dictionary

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

Returns *Subscription*

thread = None
Thread information

to_json()
Return the json representing this object.

updated_at = None
When the thread was last updated

urls = None
Dictionary of urls for the thread

class github3.notifications.**Subscription**(*sub*, *session=None*)

The *Subscription* object wraps thread and repository subscription information.

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

created_at = None

datetime representation of when the subscription was created

from_json(*json*)

Return an instance of `cls` formed from `json`.

ratelimit_remaining

Number of requests before GitHub imposes a ratelimit.

Returns int

reason = None

reason user is subscribed to this thread/repository

refresh(*conditional=False*)

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

repository_url = None

API url of the repository if it exists

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.

thread_url = None

API url of the thread if it exists

to_json()

Return the json representing this object.

2.10 Organization

This section of the documentation covers:

- *Organization*s
- *Team*s

2.10.1 Organization Objects

class github3.orgs.**Organization**(*org*, *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(*login*, *team*)

Add `login` 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.

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

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

Parameters

- **login** (*str*) – (required), login name of the user to be added
- **team** (*str*) – (required), team name

Returns bool

add_repo(*repo*, *team*)

Add `repo` to `team`.

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

Parameters

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

- **team** (*str*) – (required), team name

conceal_member(*login*)
Conceal `login`'s membership in this organization.

Returns bool

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

Parameters

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

Returns `Repository`

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

Parameters

- **name** (*str*) – (required), name to be given to the team
- **repo_names** (*list*) – (optional) repositories, e.g. ['github/dotfiles']
- **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_url = None
Events url (not a template)

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

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

Returns bool

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

Returns bool

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

Parameters

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

Returns generator of *Event*s

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

Parameters

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

Returns generator of *User*s

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

Parameters

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

Returns generator of *User*s

iter_repos(*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`

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

Parameters

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

Returns generator of *Team*s

members_urlt = None
Members URL Template. Expands with `member`

private_repos = None
Number of private repositories.

public_members_urlt = None
Public Members URL Template. Expands with `member`

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

Returns bool

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

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

Returns bool

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

Parameters

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

Returns bool

repos_url = None
Repositories url (not a template)

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

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

Returns *Team*

to_json()
Return the json representing this object.

---

class github3.orgs.**Team**(*team*, *session=None*)
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(*login*)
Add `login` to this team.

Returns bool

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

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

Returns bool

delete()
Delete this team.

Returns bool

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

Parameters

- **name** (*str*) – (required)

- **permission** (*str*) – (optional), ('pull', 'push', 'admin')

Returns bool

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

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

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

Returns bool

id = None
Unique ID of the team.

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

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

Returns bool

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

Parameters

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

Returns generator of *User*s

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

Parameters

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

Returns generator of `Repository` objects

members_count = None
Number of members in this team.

members_urlt = None
Members URL Template. Expands with `member`

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

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

Returns dictionary

name = None
This team's name.

permission = None
Permission leve of the group

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

remove_member(*login*)
Remove `login` from this team.

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

Returns bool

remove_repo(*repo*)
Remove `repo` from this team.

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

Returns bool

repos_count = None
Number of repos owned by this team.

repositories_url = None
Repositories url (not a template)

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

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

Returns bool

to_json()
Return the json representing this object.

2.11 Pull Request

This section of the documentation covers:

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

2.11.1 Pull Request Objects

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

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/

additions = None
Number of additions on this pull request

assignee = None
User object representing the assignee of the pull request

base = None
Base of the merge

body = None
Body of the pull request message

body_html = None
Body of the pull request as HTML

body_text = None
Body of the pull request as plain text

close()
Closes this Pull Request without merging.

Returns bool

closed_at = None
datetime object representing when the pull was closed

comments = None
Number of comments

comments_url = None
Comments url (not a template)

commits = None
Number of commits

commits_url = None
GitHub.com url of commits in this pull request

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*

created_at = None
datetime object representing when the pull was created

deletions = None
Number of deletions on this pull request

diff()
Return the diff

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

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

head = None
The new head after the pull request

html_url = None
The URL of the pull request

id = None
The unique id of the pull request

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

Returns bool

issue_url = None
The URL of the associated issue

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

Parameters

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

Returns generator of *ReviewComment*s

iter_commits(*number=-1*, *etag=None*)
Iterates 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 *Commit*s

iter_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

iter_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`

links = None
Dictionary of _links

merge(*commit_message=u''*, *sha=None*)
Merge this pull request.

Parameters **commit_message** (*str*) – (optional), message to be used for the merge commit

Returns bool

merge_commit_sha = None
SHA of the merge commit. DEPRECATED

mergeable = None
Whether the pull is deemed mergeable by GitHub

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

merged_at = None
datetime object representing when the pull was merged

merged_by = None
User who merged this pull

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

patch()
Return the patch

patch_url = None
The URL of the patch

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

reopen()
Re-open a closed Pull Request.

Returns bool

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

review_comment_url = None
Review comment URL Template. Expands with `number`

review_comments(*number=-1*, *etag=None*)
Iterate over the review comments on this pull request. :param int number: (optional), number of comments to return. Default:

-1 returns all available comments.

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

Returns generator of *ReviewComment*s

review_comments_count = None
Number of review comments on the pull request

review_comments_url = None
GitHub.com url for review comments (not a template)

state = None
The state of the pull

statuses_url = None
Statuses URL

title = None
The title of the request

to_json()
Return the json representing this object.

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

Parameters

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

Returns bool

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

user = None
User object representing the creator of the pull request

---

class github3.pulls.**ReviewComment**(*comment*, *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/

commit_id = None
SHA of the commit the comment is on

delete()
Delete this comment.

Returns bool

diff_hunk = None
The diff hunk

edit(*body*)
Edit this comment.

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

Returns bool

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

original_commit_id = None
Original commit SHA

original_position = None
Original position inside the file

path = None
Path to the file

position = None
Position within the commit

pull_request_url = None
API URL for the Pull Request

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

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*

to_json ()
Return the json representing this object.

user = None
User who made the comment

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

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

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

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

label = None
label of the destination

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

ref = None
Full reference string of the object

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

sha = None
SHA of the commit at the head

to_json()
Return the json representing this object.

user = None
User representing the owner

---

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

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

additions = None
Number of additions on this file

blob_url = None
URL to view the blob for this file

changes = None
Number of changes made to this file

deletions = None
Number of deletions on this file

filename = None
Name of the file

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

patch = None
Patch generated by this pull request

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

sha = None
SHA of the commit

status = None
Status of the file, e.g., 'added'

to_json()
Return the json representing this object.

2.12 Repository

This part of the documentation covers:

- *Repository*
- *Asset*
- *Branch*
- *Contents*
- *Deployment*
- *DeploymentStatus*
- *Hook*
- *PagesInfo*
- *PagesBuild*
- *Release*
- *RepoTag*
- *RepoComment*
- *RepoCommit*
- *Comparison*
- *Status*
- *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.

2.12.1 Repository Objects

class `github3.repos.repo.`**Repository**(*repo*, *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(*login*)

Add `login` as a collaborator to a repository.

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

Returns bool – True if successful, False otherwise

archive(*format*, *path=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

archive_urlt = None

Archive URL Template. Expand with `archive_format` and `ref`

asset(*id*)

Returns a single Asset.

Parameters **id** (*int*) – (required), id of the asset

Returns *Asset*

assignees_urlt = None

Assignees URL Template. Expand with `user`

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

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

Returns *Blob* if successful, otherwise None

blobs_urlt = None
Blobs URL Template. Expand with `sha`

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

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

Returns *Branch*

branches_urlt = None
Branches URL Template. Expand with `branch`

clone_url = None
URL used to clone via HTTPS.

comments_urlt = None
Comments URL Template. Expand with `number`

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

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

Returns *RepoCommit* if successful, otherwise None

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

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

Returns *RepoComment* if successful, otherwise None

commits_urlt = None
Commits URL Template. Expand with `sha`

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

compare_urlt = None
Comparison URL Template. Expand with `base` and `head`

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

If the path provided is actually a directory, you will receive a dictionary back of the form:

```
{
    'filename.md': Contents(),  # Where Contents an instance
    'github.py': Contents(),
}
```

Parameters

- **path** (*str*) – (required), path to file, e.g. github3/repo.py
- **ref** (*str*) – (optional), the string name of a commit/branch/tag. Default: master

Returns *Contents* or dict if successful, else None

contents_urlt = None
Contents URL Template. Expand with `path`

contributors_url = None
Contributors url (not a template)

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={}*, *committer={}*)
Create a commit on this repository.

Parameters

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

Returns *Commit* if successful, else None

create_deployment(*ref*, *force=False*, *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.
- **force** (*bool*) – Optional parameter to bypass any ahead/behind checks or commit status checks. Default: False
- **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*)
Creates an issue on this repository.

Parameters

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

Returns *Issue* if successful, otherwise None

create_key(*title*, *key*)
Create a deploy key.

Parameters

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

Returns *Key* if successful, else None

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

Parameters

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

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=u''*)
Create a tree on this repository.

Parameters

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

Returns *Tree* if successful, else None

created_at = None
`datetime` object representing when the Repository was created.

default_branch = None
default branch for the repository

delete()
Delete this repository.

Returns bool – True if successful, False otherwise

delete_file(*path*, *message*, *sha*, *branch=None*, *committer=None*, *author=None*)
Delete the file located at `path`.

This is part of the Contents CrUD (Create Update Delete) API. See http://developer.github.com/v3/repos/contents/#delete-a-file for more information.

Parameters

- **path** (*str*) – (required), path to the file being removed
- **message** (*str*) – (required), commit message for the deletion
- **sha** (*str*) – (required), blob sha of the file being removed
- **branch** (*str*) – (optional), if not provided, uses the repository's default branch
- **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 *Commit* if successful

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

description = None
Description of the repository.

download_url = None
Downloads url (not a template)

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_url = None
Events url (not a template)

fork = None
Is this repository a fork?

forks = None
The number of forks made of this repository. DEPRECATED

forks_count = None
The number of forks of this repository.

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

full_name = None
Full name as login/name

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

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

Returns *Commit* if successful, otherwise None

git_commits_urlt = None
Git commits URL Template. Expand with `sha`

git_refs_urlt = None
Git refs URL Template. Expand with `sha`

git_tags_urlt = None
Git tags URL Template. Expand with `sha`

git_url = None
Plain git url for an anonymous clone.

has_downloads = None
Whether or not this repository has downloads enabled

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

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

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

hook(*id_num*)
Get a single hook.

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

Returns *Hook* if successful, otherwise None

hooks_url = None
Hooks url (not a template)

html_url = None
URL of the project at GitHub.

id = None
Unique id of the repository.

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

Returns `bool`

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

Parameters **login** (*str*) – (required), login for the user

Returns bool – True if successful, False otherwise

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

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

Returns *Issue* if successful, otherwise None

issue_comment_urlt = None
Issue comment URL Template. Expand with `number`

issue_events_urlt = None
Issue events URL Template. Expand with `number`

issues_urlt = None
Issues URL Template. Expand with `number`

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

Parameters

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

Returns generator of *User*s

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

Parameters

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

Returns generator of *Branch*es

iter_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.

iter_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 *User*s

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

Parameters

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

Returns generator of *RepoComment*s

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

Parameters

- **sha** (*str*) – (required), sha of the commit to list comments on
- **number** (*int*) – (optional), number of comments to return. Default: -1 returns all comments
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of *RepoComment*s

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

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

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

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.

iter_commits(*sha=None*, *path=None*, *author=None*, *number=-1*, *etag=None*, *since=None*, *until=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 commits to return. Default: -1 returns all commits
- **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.

Returns generator of *RepoCommit*s

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

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

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*

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.

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

Parameters

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

Returns generator of *User*s

iter_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

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

Parameters

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

Returns generator of *Event*s

iter_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*

iter_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

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

Parameters

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

Returns generator of *IssueEvent*s

iter_issues(*milestone=None*, *state=None*, *assignee=None*, *mentioned=None*, *labels=None*, *sort=None*, *direction=None*, *since=None*, *number=-1*, *etag=None*)
Iterate over issues on this repo based upon parameters passed.

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

iter_keys(*number=-1*, *etag=None*)
Iterates over deploy keys on this repository.

Parameters

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

Returns generator of *Key*s

iter_labels(*number=-1*, *etag=None*)
Iterates over labels on this repository.

Parameters

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

Returns generator of *Label*s

iter_languages(*number=-1*, *etag=None*)
Iterate over the programming languages used in the repository.

Parameters

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

Returns generator of tuples

iter_milestones(*state=None*, *sort=None*, *direction=None*, *number=-1*, *etag=None*)
Iterates over the milestones on this repository.

Parameters

- **state** (*str*) – (optional), state of the milestones, accepted values: ('open', 'closed')
- **sort** (*str*) – (optional), how to sort the milestones, accepted values: ('due_date', 'completeness')
- **direction** (*str*) – (optional), direction to sort the milestones, accepted values: ('asc', 'desc')
- **number** (*int*) – (optional), number of milestones to return. Default: -1 returns all milestones
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of *Milestone*s

iter_network_events(*number=-1*, *etag=None*)
Iterates over events on a network of repositories.

Parameters

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

Returns generator of *Event*s

iter_notifications(*all=False*, *participating=False*, *since=None*, *number=-1*, *etag=None*)
Iterates over the notifications for this repository.

Parameters

- **all** (*bool*) – (optional), show all notifications, including ones marked as read
- **participating** (*bool*) – (optional), show only the notifications the user is participating in directly
- **since** (*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*

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

Returns generator of *PagesBuild*

iter_pulls(*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

iter_refs(*subspace=u''*, *number=-1*, *etag=None*)
Iterates over references for this repository.

Parameters

- **subspace** (*str*) – (optional), e.g. 'tags', 'stashes', 'notes'
- **number** (*int*) – (optional), number of refs to return. Default: -1 returns all available refs
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of *Reference*s

iter_releases(*number=-1*, *etag=None*)
Iterates 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

iter_stargazers(*number=-1*, *etag=None*)
List users who have starred this repository.

Parameters

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

Returns generator of *User*s

iter_statuses(*sha*, *number=-1*, *etag=None*)
Iterates over the statuses for a specific SHA.

Parameters

- **sha** (*str*) – SHA of the commit to list the statuses of
- **number** (*int*) – (optional), return up to number statuses. Default: -1 returns all available statuses.
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of *Status*

iter_subscribers(*number=-1*, *etag=None*)
Iterates over users subscribed to this repository.

Parameters

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

Returns generator of *User*

iter_tags(*number=-1*, *etag=None*)
Iterates over tags on this repository.

Parameters

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

Returns generator of *RepoTag*s

iter_teams(*number=-1*, *etag=None*)
Iterates over teams with access to this repository.

Parameters

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

Returns generator of *Team*s

key(*id_num*)
Get the specified deploy key.

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

Returns *Key* if successful, else None

label(*name*)
Get the label specified by `name`

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

Returns *Label* if successful, else None

labels_urlt = None
Labels URL Template. Expand with `name`

language = None
Language property.

languages_url = None
Languages url (not a template)

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

Returns *PagesBuild*

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

master_branch = None
master (default) branch for the repository

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*

merges_url = None
Merges url (not a template)

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

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

Returns *Milestone*

milestones_urlt = None
Milestones URL Template. Expand with `number`

mirror_url = None
Mirror property.

name = None
Name of the repository.

notifications_urlt = None
Notifications URL Template. Expand with `since`, `all`, `participating`

open_issues = None
Number of open issues on the repository. DEPRECATED

open_issues_count = None
Number of open issues on the repository

owner = None
User object representing the repository owner.

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

Returns *PagesInfo*

parent = None
Parent of this fork, if it exists *Repository*

permissions = None
Permissions for this repository

private = None
Is this repository private?

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

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

Returns *PullRequest*

pulls_urlt = None
Pull Requests URL Template. Expand with `number`

pushed_at = None
`datetime` object representing the last time commits were pushed to the repository.

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 and returns the refreshed instance.

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

release(*id*)

Get a single release.

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

Returns *Release*

remove_collaborator(*login*)

Remove collaborator `login` from the repository.

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

Returns bool

set_subscription(*subscribed*, *ignored*)

Set the user's subscription for this repository

Parameters

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

Returns *Subscription*

size = None

Size of the repository.

source = None

Parent of this fork, if it exists *Repository*

ssh_url = None

URL to clone the repository via SSH.

stargazers = None
Number of users who starred the repository

stargazers_url = None
Stargazers url (not a template)

statuses_urlt = None
Statuses URL Template. Expand with `sha`

subscribers_url = None
Subscribers url (not a template)

subscription()
Return subscription for this Repository.

Returns *Subscription*

subscription_url = None
Subscription url (not a template)

svn_url = None
If it exists, url to clone the repository via SVN.

tag(*sha*)
Get an annotated tag.

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

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

Returns *Tag*

tags_url = None
Tags url (not a template)

teams_url = None
Teams url (not a template)

to_json()
Return the json representing this object.

tree(*sha*)
Get a tree.

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

Returns *Tree*

trees_urlt = None
Trres URL Template. Expand with `sha`

update_file(*path*, *message*, *content*, *sha*, *branch=None*, *author=None*, *committer=None*)
Update the file `path` with `content`.

This is part of the Contents CrUD (Create Update Delete) API. See http://developer.github.com/v3/repos/contents/#update-a-file for more information.

Parameters

- **path** (*str*) – (required), path to the file being updated
- **message** (*str*) – (required), commit message
- **content** (*str*) – (required), updated contents of the file
- **sha** (*str*) – (required), blob sha of the file being updated

- **branch** (*str*) – (optional), uses the default branch on the repository if not provided.
- **author** (*dict*) – (optional), if omitted this will be filled in with committer information. If passed, you must specify both a name and email.

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

update_label(*name*, *color*, *new_name=u''*)
Update the label `name`.

Parameters

- **name** (*str*) – (required), name of the label
- **color** (*str*) – (required), color code
- **new_name** (*str*) – (optional), new name of the label

Returns bool

updated_at = None
`datetime` object representing the last time the repository was updated.

watchers = None
Number of users watching the repository.

weekly_commit_count()
Returns the total commit counts.

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

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

---

class github3.repos.branch.**Branch**(*branch*, *session=None*)
The *Branch* object. It holds the information GitHub returns about a branch on a *Repository*.

commit = None
Returns the branch's *RepoCommit* or `None`.

links = None
Returns '_links' attribute.

name = None
Name of the branch.

---

class github3.repos.contents.**Contents**(*content*, *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/

content = None
Base64-encoded content of the file.

decoded = None
Decoded content of the file as a bytes object. If we try to decode to character set for you, we might encounter an exception which will prevent the object from being created. On python2 this is the same as a string, but on python3 you should call the decode method with the character set you wish to use, e.g., `content.decoded.decode('utf-8')`. .. versionchanged:: 0.5.2

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 *Commit*

encoding = None
Returns encoding used on the content.

git_url = None
URL for the git api pertaining to the README

html_url = None
URL of the README on github.com

links = None
Dictionary of links

name = None
Name of the content.

path = None
Path to the content.

sha = None
SHA string.

size = None
Size of the content

submodule_git_url = None
git:// URL of the content if it is a submodule

target = None
Target will only be set of type is a symlink. This is what the link points to

type = None
Type of content. ('file', 'symlink', 'submodule')

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 *Commit*

---

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

create_status(*state*, *target_url=u''*, *description=u''*)
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*

created_at = None
Date the Deployment was created

creator = None
User object representing the creator of the deployment

description = None
Description of the deployment

environment = None
Target for the deployment, e.g., 'production', 'staging'

id = None
GitHub's id of this deployment

iter_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

payload = None
JSON string payload of the Deployment

ref = None
The reference used to create the Deployment, e.g., *deploy-20140526*

sha = None
SHA of the branch on GitHub

statuses_url = None
URL to get the statuses of this deployment

updated_at = None
Date the Deployment was updated

---

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

created_at = None
Date the deployment status was created

creator = None
Creater of the deployment status

deployment = None
Deployment representing the deployment this status is associated with

deployment_url = None
URL for the deployment this status is associated with

description = None
Description of the deployment

id = None
GitHub's id for this deployment status

payload = None
JSON payload as a string

state = None
State of the deployment status

target_url = None
Target URL of the deployment

updated_at = None
Date the deployment status was updated

---

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

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

assets = None
List of *Asset* objects for this release

assets_url = None
URL for uploaded assets

body = None
Body of the release (the description)

created_at = None
Date the release was created

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

Returns True if successful; False if not successful

draft = None
Boolean whether value is True or False

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

html_url = None
HTML URL of the release

id = None
GitHub id

iter_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

name = None
Name given to the release

published_at = None
Date the release was published

tag_name = None
Name of the tag

target_commitish = None
"Commit" that this release targets

upload_asset(*content_type*, *name*, *asset*)
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.

Returns *Asset*

upload_urlt = None
URITemplate to upload an asset with

---

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

content_type = None
Content-Type provided when the asset was created

created_at = None
Date the asset was created

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 bool – True if successful, False otherwise

download_count = None
Number of times the asset was downloaded

download_url = None
URL to download the asset. Request headers must include `Accept: application/octet-stream`.

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

id = None
GitHub id of the asset

label = None
Short description of the asset

name = None
Name of the asset

size = None
Size of the asset

state = None
State of the asset, e.g., "uploaded"

updated_at = None
Date the asset was updated

---

class github3.repos.hook.**Hook**(*hook*, *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/

active = None
Whether or not this Hook is marked as active on GitHub

config = None
Dictionary containing the configuration for the Hook.

created_at = None
datetime object representing the date the hook was created.

delete()
Delete this hook.

Returns bool

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 remvoed from the list of events that this hook triggers for
- **active** (*bool*) – (optional), should this event be active

Returns bool

events = None
Events which trigger the hook.

id = None
Unique id of the hook.

name = None
The name of the hook.

ping()
Ping this hook.

Returns bool

test()
Test this hook

Returns bool

updated_at = None
datetime object representing when this hook was last updated.

---

class github3.repos.pages.**PagesInfo**(*info*)

cname = None
CName used for the pages site

custom_404 = None
Boolean indicating whether there is a custom 404 for the pages site

status = None
Status of the pages site, e.g., built

---

class github3.repos.pages.**PagesBuild**(*build*)

commit = None
SHA of the commit that triggered the build

created_at = None
Datetime the build was created

duration = None
Time the build took to finish

error = None
Error dictionary containing the error message

pusher = None
User representing who pushed the commit

status = None
Status of the pages build, e.g., building

updated_at = None
Datetime the build was updated

---

class github3.repos.tag.**RepoTag**(*tag*)
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

commit = None
Dictionary containing the SHA and URL of the commit.

name = None
Name of the tag.

tarball_url = None
URL for the GitHub generated tarball associated with the tag.

zipball_url = None
URL for the GitHub generated zipball associated with the tag.

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

class github3.repos.comment.**RepoComment**(*comment*, *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
```

commit_id = None
Commit id on which the comment was made.

delete()
Delete this comment.

Returns bool

edit(*body*)
Edit this comment.

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

Returns bool

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

html_url = None
URL of the comment on GitHub.

line = None
The line number where the comment is located.

path = None
The path to the file where the comment was made.

position = None
The position in the diff where the comment was made.

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

to_json()
Return the json representing this object.

update(*body*)
Update this comment.

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

Returns bool

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

user = None
Login of the user who left the comment.

---

class github3.repos.commit.**RepoCommit**(*commit*, *session=None*)
The *RepoCommit* object. This represents a commit as viewed by a `Repository`. This is different from a Commit object returned from the git data section.

Two commit instances can be checked like so:

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

And is equivalent to:

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

additions = None
The number of additions made in the commit.

author = None
User who authored the commit.

commit = None
Commit.

committer = None
User who committed the commit.

deletions = None
The number of deletions made in the commit.

diff()
Return the diff

files = None
The files that were modified by this commit.

patch()
Return the patch

total = None
Total number of changes in the files.

---

class github3.repos.comparison.**Comparison**(*compare*)
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

ahead_by = None
Number of commits ahead by.

base_commit = None
RepoCommit object representing the base of comparison.

behind_by = None
Number of commits behind by.

commits = None
List of *RepoCommit* objects.

diff()
Return the diff

diff_url = None
URL to see the diff between the two commits.

files = None
List of dicts describing the files modified.

html_url = None
URL to view the comparison at GitHub

patch()
Return the patch

patch_url = None
Patch URL at GitHub for the comparison.

permalink_url = None
Permanent link to this comparison.

status = None
Behind or ahead.

total_commits = None
Number of commits difference in the comparison.

---

class github3.repos.status.**Status**(*status*)
The *Status* object. This represents information from the Repo Status API.

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

created_at = None
datetime object representing the creation of the status object

creator = None
User who created the object

description = None
Short description of the Status

id = None
GitHub ID for the status object

state = None
State of the status, e.g., 'success', 'pending', 'failed', 'error'

target_url = None
URL to view more information about the status

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

---

class github3.repos.stats.**ContributorStats**(*stats_object*, *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.

alt_weeks = None
Alternative collection of weekly dictionaries This provides a datetime object and easy to remember keys for each element in the list. 'w' -> 'start of week', 'a' -> 'Number of additions', 'd' -> 'Number of deletions', 'c' -> 'Number of commits'

author = None
Contributor in particular that this relates to

total = None
Total number of commits authored by `author`.

weeks = None
List of weekly dictionaries.

2.13 Search Structures

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

2.13.1 Objects

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

git_url = None
URL to the Git blob endpoint

html_url = None
URL to the HTML view of the blob

name = None
Filename the match occurs in

path = None
Path in the repository to the file

repository = None
Repository the code snippet belongs to

score = None
Score of the result

sha = None
SHA in which the code can be found

text_matches = None
Text matches

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

issue = None
Issue object

score = None
Score of the result

text_matches = None
Text matches

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

repository = None
Repository object

score = None
Score of the result

text_matches = None
Text matches

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

score = None
Score of this search result

text_matches = None
Text matches

user = None
User object matching the search

2.14 Structures

2.14.1 Developed for github3.py

As of right now, there exists only one class in this section, and it is of only limited importance to users of github3.py. The *GitHubIterator* class is used to return the results of calls to almost all of the calls to `iter_` methods on objects. When conditional refreshing was added to objects, there was a 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.

2.14.2 Objects

class `github3.structs.`**`GitHubIterator`**(*count*, *url*, *cls*, *session*, *params=None*, *etag=None*, *headers=None*)

The *GitHubIterator* class powers all of the iter_* methods.

`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_json`(*json*)
Return an instance of `cls` 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

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

`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.

from_json(*json*)

Return an instance of `cls` formed from `json`.

items = None

Items array returned in the last request

ratelimit_remaining

Number of requests before GitHub imposes a ratelimit.

Returns int

to_json()

Return the json representing this object.

total_count = None

Total count returned by GitHub

2.15 User

This part of the documentation covers:

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

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

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

2.15.1 User Modules

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

The *User* object. This handles and structures information in the User section.

Two user instances can be checked like so:

```
u1 == u2
u1 != u2
```

And is equivalent to:

```
u1.id == u2.id
u1.id != u2.id
```

add_email_address(*address*)

Add the single email address to the authenticated user's account.

Parameters **address** (*str*) – (required), email address to add

Returns list of email addresses

add_email_addresses(*addresses=[]*)
Add the email addresses in `addresses` to the authenticated user's account.

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

Returns list of email addresses

delete_email_address(*address*)
Delete the email address from the user's account.

Parameters **address** (*str*) – (required), email address to delete

Returns bool

delete_email_addresses(*addresses=[]*)
Delete the email addresses in `addresses` from the authenticated user's account.

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

Returns bool

disk_usage = None
How much disk consumed by the user

events_urlt = None
Events URL Template. Expands with `privacy`

followers_url = None
Followers URL (not a template)

following_urlt = None
Following URL Template. Expands with `other_user`

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

gists_urlt = None
Gists URL Template. Expands with `gist_id`

gravatar_id = None
ID of the user's image on Gravatar

hireable = None
True – for hire, False – not for hire

is_assignee_on(*login*, *repository*)
Checks if this user can be assigned to issues on login/repository.

Returns `bool`

is_following(*login*)
Checks if this user is following `login`.

Parameters **login** (*str*) – (required)

Returns bool

iter_events(*public=False*, *number=-1*, *etag=None*)
Iterate over events performed by this user.

Parameters

- **public** (*bool*) – (optional), only list public events for the authenticated user
- **number** (*int*) – (optional), number of events to return. Default: -1 returns all available events.

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

Returns list of *Event*s

iter_followers(*number=-1*, *etag=None*)
Iterate over the followers of this user.

Parameters

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

Returns generator of *User*s

iter_following(*number=-1*, *etag=None*)
Iterate over the users being followed by this user.

Parameters

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

Returns generator of *User*s

iter_keys(*number=-1*, *etag=None*)
Iterate over the public keys of this user.

New in version 0.5.

Parameters

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

Returns generator of *Key*s

iter_org_events(*org*, *number=-1*, *etag=None*)
Iterate over events as they appear on the user's organization dashboard. You must be authenticated to view this.

Parameters

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

Returns list of *Event*s

iter_orgs(*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 list of *Event*s

iter_received_events(*public=False*, *number=-1*, *etag=None*)
Iterate over events that the user has received. If the user is the authenticated user, you will see private and public events, otherwise you will only see public events.

Parameters

- **public** (*bool*) – (optional), determines if the authenticated user sees both private and public or just public
- **number** (*int*) – (optional), number of events to return. Default: -1 returns all events available
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of *Event*s

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

Changed in version 0.5: Added sort and direction parameters (optional) as per the change in GitHub's API.

Parameters

- **number** (*int*) – (optional), number of starred repos to return. Default: -1, returns all available repos
- **sort** (*str*) – (optional), either 'created' (when the star was created) or 'updated' (when the repository was last pushed to)
- **direction** (*str*) – (optional), either 'asc' or 'desc'. Default: 'desc'
- **etag** (*str*) – (optional), ETag from a previous request to the same endpoint

Returns generator of `Repository`

iter_subscriptions(*number=-1*, *etag=None*)
Iterate over repositories subscribed to by this user.

Parameters

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

Returns generator of `Repository`

organizations_url = None
Organizations URL (not a template)

owned_private_repos = None
Number of private repos owned by this user

plan = None
Which plan this user is on

public_gists = None
Number of public gists

ratelimit_remaining
Number of requests before GitHub imposes a ratelimit.

Returns int

received_events_url = None
Received Events URL (not a template)

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

repos_url = None
Repostories URL (not a template)

starred_urlt = None
Starred URL Template. Expands with `owner` and `repo`

subscriptions_url = None
Subscriptions URL (not a template)

to_json()
Return the json representing this object.

total_private_gists = None
Number of private gists owned by this user

total_private_repos = None
Total number of private repos

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

Parameters

- **name** (*str*) – e.g., 'John Smith', not login name
- **email** (*str*) – e.g., 'john.smith@example.com'
- **blog** (*str*) – e.g., 'http://www.example.com/jsmith/blog'
- **company** (*str*) –
- **location** (*str*) –
- **hireable** (*bool*) – defaults to False
- **bio** (*str*) – GitHub flavored markdown

Returns bool

class github3.users.**Key**(*key*, *session=None*)

The *Key* object. Please see GitHub's Key Documentation for more information.

Two key instances can be checked like so:

```
k1 == k2
k1 != k2
```

And is equivalent to:

```
k1.id == k2.id
k1.id != k2.id
```

delete()

Delete this Key

from_json(*json*)

Return an instance of `cls` formed from `json`.

id = None

The unique id of the key at GitHub

key = None

The text of the actual key

ratelimit_remaining

Number of requests before GitHub imposes a ratelimit.

Returns int

refresh(*conditional=False*)

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

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

Returns self

The reasoning for the return value is the following example:

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

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

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

Which is really an anti-pattern.

Changed in version 0.5.

title = None

The title the user gave to the key

to_json()

Return the json representing this object.

update(*title*, *key*)

Update this key.

Parameters

- **title** (*str*) – (required), title of the key

- **key** (*str*) – (required), text of the key file

Returns bool

---

class github3.users.**Plan**(*plan*)

The *Plan* object. This makes interacting with the plan information about a user easier. Please see GitHub's Authenticated User documentation for more specifics.

collaborators = None
Number of collaborators

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

is_free()
Checks if this is a free plan.

Returns bool

name = None
Name of the plan

private_repos = None
Number of private repos

space = None
Space allowed

to_json()
Return the json representing this object.

2.16 Internals

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

2.16.1 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
```

3.1 Dependencies

- requests by Kenneth Reitz
- uritemplate.py 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.

4.1 Contributor Friendly Work

In order of importance:

Documentation

> I know I'm not the best at writing documentation so if you want to clarify or correct something, please do so.

Examples

> Have a clever example that takes advantage of github3.py? Feel free to share it.

4.2 Running the Unittests

```
mkdir -p /path/to/virtualenv/github3.py
cd /path/to/virtualenv/github3.py
virtualenv .
cd /path/to/github3.py_repo/
pip install -r dev-requirements.txt
# Or you could run make test-deps
make tests
```

4.2.1 Writing Tests for github3.py

Unit Tests

> In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application.
>
> —Unit Testing on Wikipedia

In github3.py we use unit tests to make assertions about how the library behaves without making a request to the internet. For example, one assertion we might write would check if custom information is sent along in a request to GitHub.

An existing test like this can be found in `tests/unit/test_repos_release.py`:

```
def test_delete(self):
    self.instance.delete()
    self.session.delete.assert_called_once_with(
        self.example_data['url'],
        headers={'Accept': 'application/vnd.github.manifold-preview'}
    )
```

In this test, we check that the library passes on important headers to the API to ensure the request will work properly. `self.instance` is created for us and is an instance of the `Release` class. The test then calls `delete` to make a request to the API. `self.session` is a mock object which fakes out a normal session. It does not allow the request through but allows us to verify how github3.py makes a request. We can see that github3.py called `delete` on the session. We assert that it was only called once and that the only parameters sent were a URL and the custom headers that we are concerned with.

Mocks

Above we talked about mock objects. What are they?

> In object-oriented programming, mock objects are simulated objects that mimic the behavior of real objects in controlled ways. A programmer typically creates a mock object to test the behavior of some other object, in much the same way that a car designer uses a crash test dummy to simulate the dynamic behavior of a human in vehicle impacts.
>
> —Mock Object on Wikipedia

We use mocks in github3.py to prevent the library from talking directly with GitHub. The mocks we use intercept requests the library makes so we can verify the parameters we use. In the example above, we were able to check that certain parameters were the only ones sent to a session method because we mocked out the session.

You may have noticed in the example above that we did not have to set up the mock object. There is a convenient helper written in `tests/unit/helper.py` to do this for you.

Example - Testing the Release Object

Here's a full example of how we test the `Release` object in `tests/unit/test_repos_release.py`.

Our first step is to import the `UnitHelper` class from `tests/unit/helper.py` and the `Release` object from `github3/repos/release.py`.

```
from .helper import UnitHelper
from github3.repos.release import Release
```

Then we construct our test class and indicate which class we will be testing (or describing).

```
class TestRelease(UnitHelper):
    described_class = Release
```

We can then use the GitHub API documentation about Releases to retrieve example release data. We then can use that as example data for our 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_PASS="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

History/Changelog

6.1 0.9.6: 2016-09-24

- Add branch parameter to Repository Contents delete and update calls.
- Cap dependency on uritemplate.py to always be less than 3.0

6.2 0.9.5: 2016-02-15

- Fix Validation Error stemming from `Repository#create_status` having poorly chosen default arguments and not removing `None` values from the body of the request.

6.3 0.9.4: 2015-04-17

- In `PullRequest#create_review_comment` coerce the position argument to an integer instead of coercing it to a string. Reported by Paul Tagliamonte in #374.
- Backport support for the `context` parameter in `Repository#create_status`
- Add support for `Repository.permissions` attribute
- Backport of support for allowing `Events` to keep the same session as other objects.
- Skip objects that are `None` while iterating over them (see issues #304 and #305) reported by Marc Abramowitz
- Fix URL regular expression for GitHub Enterprise instances by Marc Abramowitz

6.4 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

6.5 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`.

6.6 0.9.1: 2014-08-10

- Correct Repository attribute `fork_count` should be `forks_count`

6.7 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`.

6.8 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

6.9 0.8.1: 2014-01-26

- Add documentation for using Two Factor Authentication
- Fix oversight where `github3.login` could not be used for 2FA

6.10 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

6.11 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.

6.12 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')
```

6.13 0.6.1: 2013-04-06

- Add equality for labels courtesy of Alejandro Gomez (@alejandrogomez)

6.14 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.

6.15 0.5.3: 2013-03-19

- Add missing optional parameter to Repository.contents. Thanks @tpetr

6.16 0.5.2: 2013-03-02

- Stop trying to decode the byte strings returned by `b64decode`. Fixes #72

6.17 0.5.1: 2013-02-21

- Hot fix an issue when a user doesn't have a real name set

6.18 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.

6.19 0.4: 2013-01-16

- In github3.legacy.LegacyRepo
  - `has_{downloads,issues,wiki}` are now attributes.
  - `is_private()` and the `private` attribute return the same thing `is_private()` will be deprecated in the next release.
- In github3.repos.Repository
  - `is_fork()` is now deprecated in favor of the `fork` attribute
  - `is_private()` is now deprecated in favor of the `private` attribute
- In github3.repos.Hook
  - `is_active()` is now deprecated in favor of the `active` attribute
- In github3.pulls.PullRequest
  - `is_mergeable()` is now deprecated in favor of the `mergeable` attribute
- In github3.notifications.Thread
  - `is_unread()` is now deprecated in favor of the `unread`
- `pubsubhubbub()` is now present on the `GitHub` object and will be removed from the `Repository` object in the next release
- 70% test coverage

6.20 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

6.21 0.2: 2012-11-21

- MAJOR API CHANGES:
  - `GitHub.iter_subscribed` -> `GitHub.iter_subscriptions`
  - Broken `list_*` functions in github3.api have been renamed to the correct `iter_*` methods on `GitHub`.
  - Removed `list_*` functions from `Repository`, `Gist`, `Organization`, and `User` objects
- Added zen of GitHub method.
- More tests
- Changed the way `Repository.edit` works courtesy of Kristian Glass (@doismellburning)
- Changed `Repository.contents` behaviour when acting on a 404.
- 50% test coverage via mock tests

6.22 0.1: 2012-11-13

- Add API for GitHub Enterprise customers.

6.23 0.1b2: 2012-11-10

- Handle 500 errors better, courtesy of Kristian Glass (@doismellburning)
- Handle sending json with % symbols better, courtesy of Kristian Glass
- Correctly handle non-GitHub committers and authors courtesy of Paul Swartz (@paulswartz)
- Correctly display method signatures in documentation courtesy of (@seveas)

6.24 0.1b1: 2012-10-31

- unit tests implemented using mock instead of hitting the GitHub API (#37)
- removed `list_*` functions from GitHub object
- Notifications API coverage

6.25 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

g

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z