pygithub3 Documentation

Release 0.5.1

David Medina

June 29, 2016

Contents

pygithub3 is a Github APIv3 python wrapper.

You can consume the API with several Services (users, repos...) like you see in Github API v3 documentation.

When you do an API request, **pygithub3** map the result into Resources which can do its own related requests in future releases.

CHAPTER 1

Fast sample

```
from pygithub3 import Github

gh = Github(login='octocat', password='password')

octocat = gh.users.get() # Auth required
copitux = gh.users.get('copitux')
# copitux = <User (copitux)>

octocat_followers = gh.users.followers.list().all()
copitux_followers = gh.users.followers.list('copitux').all()
# copitux_followers = [<User (ahmontero)>, <User (alazaro)>, ...]
```

CHAPTER 2

Others

You must apologize my English level. I'm trying to do my best

2.1 Installation

```
pip install pygithub3

or

easy_install pygithub3
```

2.1.1 Dependencies

Required

This library depends only on the requests module.

If you install `pygithub3` with `pip` all is done. This is the best option.

Optional

The test suite uses nose, mock, and unittest2 (python 2.6). Compiling the documentation requires sphinx.

Install all of these by running `pip install -r test_requirements.txt`. Then just run `nosetests` to run the tests.

2.2 Github

This is the main entrance of **pygithub3**

class `pygithub3.`**`Github`**(***config*)

You can preconfigure all services globally with a `config` dict. See *`Service`*

Example:

```
    gh = Github(user='kennethreitz', token='ABC...', repo='requests')
```

events
Events service

gists
Gists service

git_data
Git Data service

issues
Issues service

orgs
Orgs service

pull_requests
Pull Requests service

remaining_requests
Limit of Github API v3

repos
Repos service

users
Users service

2.3 Services

Github class is a glue to all of them and the recommended option to start

2.3.1 Overview

You can access to the API requests through the different services.

If you take a look at github API v3 documentation, you'll see a few sections in the sidebar.

pygithub3 has one service per each section of request-related

For example:

```
repos => services.repos.repo
    collaborators => services.repos.collaborators
    commits => services.repos.commits
    ....
```

Each service has the functions to throw the API requests and **is isolated from the rest**.

2.3.2 Config each service

Each service can be configurated with some variables (behind the scenes, each service has her client which is configurated with this variables).

Note: Also you can configure Github as a service

class pygithub3.services.base.**Service**(**config*)
You can configure each service with this keyword variables:

Parameters

- **login** (*str*) – Username to authenticate
- **password** (*str*) – Username to authenticate
- **user** (*str*) – Default username in requests
- **repo** (*str*) – Default repository in requests
- **token** (*str*) – Token to OAuth
- **per_page** (*int*) – Items in each page of multiple returns
- **base_url** (*str*) – To support another github-related API (untested)
- **verbose** (*stream*) – Stream to write debug logs
- **float** (*timeout*) – Timeout for requests

You can configure the **authentication** with BasicAuthentication (login and password) and with OAuth (token). If you include `login`, `password` and `token` in config; Oauth has precedence

Some API requests need `user` and/or `repo` arguments (e.g *repos service*). You can configure the default value here to avoid repeating

Some API requests return multiple resources with pagination. You can configure how many items has each page.

You can configure `verbose` logging like requests library

set_credentials(*login*, *password*)
Set Basic Authentication

Parameters

- **login** (*str*) – Username to authenticate
- **password** (*str*) – Username to authenticate

set_repo(*repo*)
Set repository

Parameters **repo** (*str*) – Default repository in requests

set_token(*token*)
Set OAuth token

Parameters **token** (*str*) – Token to OAuth

set_user(*user*)
Set user

Parameters **user** (*str*) – Default username in requests

2.3.3 MimeTypes

Some services supports mimetypes

With them the Resources will have `body`, `body_text`, `body_html` attributes or all of them.

class pygithub3.services.base.**MimeTypeMixin**
Mimetype support to Services

Adds 4 public functions to service:

set_full()
Resource will have body, body_text and body_html attributes

set_html()
Resource will have body_html attribute

set_raw()
Resource will have body attribute

set_text()
Resource will have body_text attribute

Fast example:

```
from pygithub3 import Github

gh = Github()

gh.gists.comments.set_html()
comment = gh.gists.comments.list(1).all()[0]
print comment.body, comment.body_text, comment.body_html
```

2.3.4 List of services

Users services

Fast sample:

```
from pygithub3 import Github

auth = dict(login='octocat', password='pass')
gh = Github(**auth)

# Get copitux user
gh.users.get('copitux')

# Get copitux's followers
gh.users.followers.list('copitux')

# Get octocat's emails
gh.users.emails.list()
```

User

class pygithub3.services.users.**User**(***config*)
Consume Users API

emails
Emails

keys
Keys

followers
Followers

get(*user=None*)
Get a single user

Parameters **user** (*str*) – Username

If you call it without user and you are authenticated, get the authenticated user.

> **Warning:** If you aren't authenticated and call without user, it returns 403

```
user_service.get('copitux')
user_service.get()
```

update(*data*)
Update the authenticated user

Parameters **data** (*dict*) – Input to update

```
user_service.update(dict(name='new_name', bio='new_bio'))
```

Emails

class pygithub3.services.users.**Emails**(***config*)
Consume Emails API

> **Warning:** You must be authenticated for all requests

add(**emails*)
Add emails

Parameters **emails** (list) – Emails to add

> **Note:** It rejects non-valid emails

```
email_service.add('test1@xample.com', 'test2@xample.com')
```

delete(**emails*)
Delete emails

Parameters **emails** (list) – List of emails

```
email_service.delete('test1@xample.com', 'test2@xample.com')
```

list()
Get user's emails

Returns A Result

Keys

class pygithub3.services.users.**Keys**(***config*)
Consume Keys API

> **Warning:** You must be authenticated for all requests

add(*data*)
Add a public key

Parameters **data** (*dict*) – Key (title and key attributes required)

```
key_service.add(dict(title='host', key='ssh-rsa AAA...'))
```

delete(*key_id*)
Delete a public key

Parameters **key_id** (*int*) – Key id

get(*key_id*)
Get a public key

Parameters **key_id** (*int*) – Key id

list()
Get public keys

Returns A Result

update(*key_id*, *data*)
Update a public key

Parameters

- **key_id** (*int*) – Key id
- **data** (*dict*) – Key (title and key attributes required)

```
key_service.update(42, dict(title='host', key='ssh-rsa AAA...'))
```

Followers

class pygithub3.services.users.**Followers**(***config*)
Consume Followers API

follow(*user*)
Follow a user

Parameters **user** (*str*) – Username

> **Warning:** You must be authenticated

is_following(*user*)
Check if you are following a user

Parameters **user** (*str*) – Username

list(*user=None*)
Get user's followers

Parameters **user** (*str*) – Username

Returns A Result

If you call it without user and you are authenticated, get the authenticated user's followers

> **Warning:** If you aren't authenticated and call without user, it returns 403

```
followers_service.list()
followers_service.list('octocat')
```

list_following(*user=None*)
Get who a user is following

Parameters **user** (*str*) – Username

Returns A Result

If you call it without user and you are authenticated, get the authenticated user's followings

> **Warning:** If you aren't authenticated and call without user, it returns 403

```
followers_service.list_following()
followers_service.list_following('octocat')
```

unfollow(*user*)
Unfollow a user

Parameters **user** (*str*) – Username

> **Warning:** You must be authenticated

Repos services

Fast sample:

```
from pygithub3 import Github

gh = Github()

django_compressor = gh.repos.get(user='jezdez', repo='django_compressor')
requests_collaborators = gh.repos.collaborators(user='kennethreitz',
    repo='requests')
```

Config precedence

Some request always need `user` and `repo` parameters, both, to identify a *repository*. Because there are a lot of requests which need that parameters, you can *Config each service* with `user` and `repo` globally.

So several requests follow a simple precedence `user_in_arg > user_in_config | repo_in_arg > repo_in_config`

You can see it better with an example:

```
from pygithub3 import Github

gh = Github(user='octocat', repo='oct_repo')
oct_repo = gh.repos.get()
another_repo_from_octocat = gh.repos.get(repo='another_repo')
```

```
django_compressor = gh.repos.get(user='jezdez', repo='django_compressor')
```

Note: Remember that each service is isolated from the rest

```
# continue example...
gh.repos.commits.set_user('copitux')
oct_repo = gh.repos.get()
oct_repo_collaborators = gh.repos.collaborators.list().all()

# Fail because copitux/oct_repo doesn't exist
gh.repos.commits.list_comments()
```

Repo

class pygithub3.services.repos.**Repo**(***config*)

Consume Repos API

collaborators

Collaborators

commits

Commits

downloads

Downloads

forks

Forks

keys

Keys

watchers

Watchers

create(*data*, *in_org=None*)

Create a new repository

Parameters

- **data** (*dict*) – Input. See github repos doc
- **in_org** (*str*) – Organization where create the repository (optional)

Warning: You must be authenticated
If you use in_org arg, the authenticated user must be a member of <in_org>

```
repo_service.create(dict(name='new_repo', description='desc'))
repo_service.create(dict(name='new_repo_in_org', team_id=2300),
    in_org='myorganization')
```

delete(*user=None*, *repo=None*)

Delete a single repo

Parameters

- **user** (*str*) – Username

- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get (*user=None*, *repo=None*)
Get a single repo

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list (*user=None*, *type='all'*)
Get user's repositories

Parameters

- **user** (*str*) – Username
- **type** (*str*) – Filter by type (optional). See github repos doc

Returns A Result

If you call it without user and you are authenticated, get the authenticated user's repositories

Warning: If you aren't authenticated and call without user, it returns 403

```
repo_service.list('copitux', type='owner')
repo_service.list(type='private')
```

list_branches (*user=None*, *repo=None*)
Get repository's branches

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_by_org (*org*, *type='all'*)
Get organization's repositories

Parameters

- **org** (*str*) – Organization name
- **type** (*str*) – Filter by type (optional). See github repos doc

Returns A Result

```
repo_service.list_by_org('myorganization', type='member')
```

list_contributors (*user=None*, *repo=None*)
Get repository's contributors

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_contributors_with_anonymous (*user=None*, *repo=None*)
Like *list_contributors* plus anonymous

list_languages (*user=None*, *repo=None*)
Get repository's languages

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_tags (*user=None*, *repo=None*)
Get repository's tags

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_teams (*user=None*, *repo=None*)
Get repository's teams

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

update (*data*, *user=None*, *repo=None*)
Update a single repository

Parameters

- **data** (*dict*) – Input. See github repos doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated

```
repo_service.update(dict(has_issues=True), user='octocat',
    repo='oct_repo')
```

Collaborators

class pygithub3.services.repos.**Collaborators**(***config*)

Consume Repo Collaborators API

add(*collaborator*, *user=None*, *repo=None*)

Add collaborator to a repository

Parameters

- **collaborator** (*str*) – Collaborator's username
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated and have perms in repository

delete(*collaborator*, *user=None*, *repo=None*)

Remove collaborator from repository

Parameters

- **collaborator** (*str*) – Collaborator's username
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated and have perms in repository

is_collaborator(*collaborator*, *user=None*, *repo=None*)

Check if a user is collaborator on repository

Parameters

- **collaborator** (*str*) – Collaborator's username

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list(*user=None*, *repo=None*)
Get repository's collaborators

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

Commits

class pygithub3.services.repos.**Commits**(***config*)
Consume Commits API

Note: This service support *MimeTypes* configuration

compare(*base*, *head*, *user=None*, *repo=None*)
Compare two commits

Parameters

- **base** (*str*) – Base commit sha
- **head** (*str*) – Head commit sha
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

```
commits_service.compare('6dcb09', 'master', user='octocat',
    repo='oct_repo')
```

create_comment(*data*, *sha*, *user=None*, *repo=None*)
Create a commit comment

Parameters

- **data** (*dict*) – Input. See github commits doc
- **sha** (*str*) – Commit's sha
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated

```
data = {
    "body": "Nice change",
    "commit_id": "6dcb09b5b57875f334f61aebed695e2e4193db5e",
    "line": 1,
    "path": "file1.txt",
    "position": 4
}
commits_service.create_comment(data, '6dcb09', user='octocat',
    repo='oct_repo')
```

delete_comment (*cid*, *user=None*, *repo=None*)
Delete a single commit comment

Parameters

- **cid** (*int*) – Commit comment id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get (*sha*, *user=None*, *repo=None*)
Get a single commit

Parameters

- **sha** (*str*) – Commit's sha
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get_comment (*cid*, *user=None*, *repo=None*)
Get a single commit comment

Parameters

- **cid** (*int*) – Commit comment id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list (*user=None*, *repo=None*, *sha=None*, *path=None*)
Get repository's commits

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository
- **sha** (*str*) – Sha or branch to start listing commits from
- **path** (*str*) – Only commits containing this file path will be returned

Returns A Result

Note: Remember *Config precedence*

Warning: Usually a repository has thousand of commits, so be careful when consume the result. You should filter with `sha` or directly clone the repository

```
commits_service.list(user='octocat', repo='oct_repo')
commits_service.list(user='octocat', repo='oct_repo', sha='dev')
commits_service.list(user='django', repo='django', sha='master',
    path='django/db/utils.py')
```

list_comments(*sha=None*, *user=None*, *repo=None*)

Get commit's comments

Parameters

- **sha** (*str*) – Commit's sha
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

If you call it without `sha`, get all commit's comments of a repository

```
commits_service.list_comments('6dcb09', user='octocat',
    repo='oct_repo')
commits_service.list_comments(user='octocat', repo='oct_repo')
```

update_comment(*data*, *cid*, *user=None*, *repo=None*)

Update a single commit comment

Parameters

- **data** (*dict*) – Input. See github commits doc
- **cid** (*int*) – Commit comment id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

> **Warning:** You must be authenticated

```
commits_service.update_comment(dict(body='nice change'), 42,
    user='octocat', repo='oct_repo')
```

Downloads

class pygithub3.services.repos.**Downloads**(***config*)
Consume Downloads API

create(*data*, *user=None*, *repo=None*)
Create a new download

Parameters

- **data** (*dict*) – Input. See github downloads doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

> **Note:** Remember *Config precedence*

It is a two step process. After you create the download, you must call the `upload` function of `Download` resource with `file_path`

> **Warning:** In *alpha* state

```
# Step 1
download = downloads_service.create(
    dict(name='new_download', size=1130),
    user='octocat', repo='oct_repo')

# Step 2
download.upload('/home/user/file.ext')
```

delete(*id*, *user=None*, *repo=None*)
Delete a download

Parameters

- **id** (*int*) – Download id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

> **Note:** Remember *Config precedence*

get(*id*, *user=None*, *repo=None*)
Get a single download

Parameters

- **id** (*int*) – Download id

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list(*user=None*, *repo=None*)
Get repository's downloads

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

Forks

class pygithub3.services.repos.**Forks**(***config*)
Consume Forks API

create(*user=None*, *repo=None*, *org=None*)
Fork a repository

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository
- **org** (*str*) – Organization name (optional)

Note: Remember *Config precedence*

Warning: You must be authenticated

If you call it with `org`, the repository will be forked into this organization.

```
forks_service.create(user='octocat', repo='oct_repo')
forks_service.create(user='octocat', repo='oct_repo',
    org='myorganization')
```

list(*user=None*, *repo=None*, *sort='newest'*)
Get repository's forks

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository
- **sort** (*str*) – Order resources (optional). See github forks doc

Returns A Result

Note: Remember *Config precedence*

```
forks_service.list(user='octocat', repo='oct_repo', sort='oldest')
```

Keys

class pygithub3.services.repos.**Keys**(***config*)
Consume Deploy keys API

create(*data*, *user=None*, *repo=None*)
Create a repository key

Parameters

- **data** (*dict*) – Input. See github keys doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated and have perms in the repository

```
keys_service.create(dict(title='new key', key='ssh-rsa AAA...'),
    user='octocat', repo='oct_repo')
```

delete(*id*, *user=None*, *repo=None*)
Delete a repository key

Parameters

- **id** (*int*) – Repository key id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get(*id*, *user=None*, *repo=None*)
Get a single repository key

Parameters

- **id** (*int*) – Repository key id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list (*user=None*, *repo=None*)
Get repository's keys

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

update (*id*, *data*, *user=None*, *repo=None*)
Update a repository key

Parameters

- **id** (*int*) – Repository key id
- **data** (*dict*) – Input. See github keys doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated and have perms in the repository

```
keys_service.update(42, dict(title='new title'),
    user='octocat', repo='oct_repo')
```

Watchers

class pygithub3.services.repos.**Watchers** (***config*)
Consume Watching API

is_watching (*user=None*, *repo=None*)
Check if authenticated user is watching a repository

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated

list (*user=None*, *repo=None*)
Get repository's watchers

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_repos(*user=None*)
Get repositories being watched by a user

Parameters **user** (*str*) – Username

Returns A Result

If you call it without user and you are authenticated, get the repositories being watched by the authenticated user.

Warning: If you aren't authenticated and call without user, it returns 403

```
watchers_service.list_repos('copitux')
watchers_service.list_repos()
```

unwatch(*user=None*, *repo=None*)
Stop watching a repository

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated

watch(*user=None*, *repo=None*)
Watch a repository

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Warning: You must be authenticated

Hooks

class pygithub3.services.repos.**Hooks**(***config*)
Consume Hooks API

Warning: You must be authenticated and have repository's admin-permission

create (*data*, *user=None*, *repo=None*)
Create a hook

Parameters

- **data** (*dict*) – Input. See github hooks doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

```
data = {
    "name": "acunote",
    "active": True,
    "config": {
        'token': 'AAA...',
    },
    "events": ['push', 'issues'],
}
hooks_service.create(data, user='octocat', repo='oct_repo')
```

delete (*hook_id*, *user=None*, *repo=None*)
Delete a single hook

Parameters

- **hook_id** (*int*) – Hook id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get (*hook_id*, *user=None*, *repo=None*)
Get a single hook

Parameters

- **hook_id** (*int*) – Hook id
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list (*user=None*, *repo=None*)
Get repository's hooks

Parameters

- **user** (*str*) – Username

- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

test(*hook_id*, *user=None*, *repo=None*)
Test a hook

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

This will trigger the hook with the latest push to the current repository.

update(*hook_id*, *data*, *user=None*, *repo=None*)
Update a single hook

Parameters

- **hook_id** (*int*) – Hook id
- **data** (*dict*) – Input. See github hooks doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

```
hooks_service.update(42, dict(active=False), user='octocat',
    repo='oct_repo')
```

Gists services

Fast sample:

```
from pygithub3 import Github

auth = dict(login='octocat', password='pass')
gh = Github(**auth)

octocat_gists = gh.gists.list()
the_first_gist = gh.gists.get(1)

the_first_gist_comments = gh.gists.comments.list(1)
```

Gist

class pygithub3.services.gists.**Gist**(***config*)
Consume Gists API

comments
Comments

create(*data*)
Create a gist

Parameters **data** (*dict*) – Input. See github gists doc

```
gist_service.create(dict(description='some gist', public=True,
    files={'xample.py': {'content': 'import code'}}))
```

delete(*id*)
Delete a gist

Parameters **id** (*int*) – Gist id

Warning: You must be authenticated

fork(*id*)
Fork a gist

Parameters **id** (*int*) – Gist id

Warning: You must be authenticated

get(*id*)
Get a single gist

Parameters **id** (*int*) – Gist id

is_starred(*id*)
Check if a gist is starred

Parameters **id** (*int*) – Gist id

Warning: You must be authenticated

list(*user=None*)
Get user's gists

Parameters **user** (*str*) – Username

Returns A Result

If you call it without user and you are authenticated, get the authenticated user's gists. but if you aren't authenticated get the public gists

```
gist_service.list('copitux')
gist_service.list()
```

public()
Get public gists

Returns A Result

Note: Be careful iterating the result

star(*id*)
Star a gist

Parameters **id** (*int*) – Gist id

> **Warning:** You must be authenticated

starred()
Get authenticated user's starred gists

Returns A Result

> **Warning:** You must be authenticated

unstar(*id*)
Unstar a gist

Parameters **id** (*int*) – Gist id

> **Warning:** You must be authenticated

update(*id*, *data*)
Update a single gist

Parameters

- **id** (*int*) – Gist id
- **data** (*dict*) – Input. See github gists doc

> **Warning:** You must be authenticated

```
gist_service.update(dict(description='edited',
    files={'xample.py': {
        'filename': 'new_xample.py',
        'content': 'import new_code'}}))
```

Comments

class pygithub3.services.gists.**Comments**(***config*)
Consume Comments API

> **Note:** This service support *MimeTypes* configuration

create(*gist_id*, *message*)
Create a comment

Parameters

- **gist_id** (*int*) – Gist id
- **message** (*str*) – Comment's message

> **Warning:** You must be authenticated

```
comment_service.create(1, 'comment')
```

delete (*id*)
Delete a comment

Parameters **id** (*int*) – Comment id

> **Warning:** You must be authenticated

get (*id*)
Get a single comment

Parameters **id** (*int*) – Comment id

list (*gist_id*)
Get gist's comments

Parameters **gist_id** (*int*) – Gist id

Returns A Result

update (*id*, *message*)
Update a comment

Parameters

- **id** (*int*) – Comment id
- **message** (*str*) – Comment's message

> **Warning:** You must be authenticated

Git Data services

Example:

```
from pygithub3 import Github

gh = Github(user='someone', repo='some_repo')

a_blob = gh.git_data.blobs.get('a long sha')

dev_branch = gh.git_data.references.get('heads/add_a_thing')
```

GitData

class pygithub3.services.git_data.**GitData**(***config*)
Consume Git Data API

blobs
Blobs

commits
Commits

references
References

tags
Tags

trees
Trees

Blobs

class pygithub3.services.git_data.**Blobs**(**config)
Consume Blobs API

create(*data*, *user=None*, *repo=None*)
Create a blob

Parameters

- **data** (*dict*) – Data describing the blob to create
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get(*sha*, *user=None*, *repo=None*)
Get a particular blob

Parameters

- **sha** (*str*) – The sha of the blob to get
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Commits

class pygithub3.services.git_data.**Commits**(**config)
Consume Commits API

create(*data*, *user=None*, *repo=None*)
create a commit on a repo

Parameters

- **data** (*dict*) – Input. See github commits doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get(*sha*, *user=None*, *repo=None*)
get a commit from the current repo

Parameters

- **sha** (*str*) – SHA of the Commit that you want
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

References

class pygithub3.services.git_data.**References**(**config)
Consume References API

create(*data*, *user=None*, *repo=None*)
Create a reference

Parameters

- **data** (*dict*) – Input. See github refs doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

delete(*ref*, *user=None*, *repo=None*)
Delete a reference

Parameters

- **ref** (*str*) – The SHA of the reference to delete
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get(*ref*, *user=None*, *repo=None*)
Get a reference

Parameters

- **ref** (*str*) – The name of the reference to get
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Note: Remember that branch references look like “heads/<branch_name>”

list (*namespace=''*, *user=None*, *repo=None*)
List all the references

Parameters

- **namespace** (*str*) – Limit the request to a particular type of reference. For example, `heads` or `tags`.
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

update (*ref*, *data*, *user=None*, *repo=None*)
Update an existing reference

Parameters

- **ref** (*str*) – The SHA of the reference to update
- **data** (*dict*) – Input. See github refs doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Tags

class pygithub3.services.git_data.**Tags** (***config*)
Consume Tags API

create (*data*, *user=None*, *repo=None*)
Create a tag

Parameters

- **data** (*dict*) – Input. See github tags doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get (*sha*, *user=None*, *repo=None*)
Get a tag

Parameters

- **sha** (*str*) – The sha of the tag to get.
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Trees

class pygithub3.services.git_data.**Trees**(**config*)
Consume Trees API

create(*data*, *user=None*, *repo=None*)
Create a tree object

Parameters

- **data** (*dict*) – Input. See github trees doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get(*sha*, *recursive=False*, *user=None*, *repo=None*)
Get a tree object

Parameters

- **sha** (*str*) – The SHA of the tree you want.
- **recursive** (*bool*) – Whether to resolve each sub-tree belonging to this tree
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Pull Requests service

Example:

```
from pygithub3 import Github

gh = Github(user='octocat', repo='sample')

pull_requests = gh.pull_requests.list().all()
pull_request_commits = gh.pull_requests.list_commits(2512).all()
```

Pull Requests

class pygithub3.services.pull_requests.**PullRequests**(**config*)
Consume Pull Request API

comments
Pull Request Comments

create (*data*, *user=None*, *repo=None*)
Create a pull request

Parameters

- **data** (*dict*) – Input. See github pullrequests doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get (*number*, *user=None*, *repo=None*)
Get a single pull request

Parameters

- **number** (*str*) – The number of the pull request to get
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

is_merged (*number*, *user=None*, *repo=None*)
Gets whether a pull request has been merged or not.

Parameters

- **number** (*str*) – The pull request to check
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list (*state='open'*, *user=None*, *repo=None*)
List all of the pull requests for a repo

Parameters

- **state** (*str*) – Pull requests state ('open' or 'closed')
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_commits (*number*, *user=None*, *repo=None*)
List the commits for a pull request

Parameters

- **number** (*str*) – The number of the pull request to list commits for
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_files(*number*, *user=None*, *repo=None*)
List the files for a pull request

Parameters

- **number** (*str*) – The number of the pull request to list files for
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

merge(*number*, *message=''*, *user=None*, *repo=None*)
Merge a pull request.

Parameters

- **number** (*str*) – The pull request to merge
- **message** (*str*) – Message of pull request
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

This currently raises an HTTP 405 error if the request is not mergable.

update(*number*, *data*, *user=None*, *repo=None*)
Update a pull request

Parameters

- **number** (*str*) – The number of the the pull request to update
- **data** (*dict*) – Input. See github pullrequests doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Pull Request Comments

class pygithub3.services.pull_requests.**Comments**(***config*)
Consume Review Comments API

create(*number*, *data*, *user=None*, *repo=None*)
Create a comment

Parameters

- **number** (*str*) – the pull request to comment on
- **data** (*dict*) – Input. See github pullrequests comments doc
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

delete(*number*, *user=None*, *repo=None*)
Delete a comment

Parameters

- **number** (*str*) – The comment to delete
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

get(*number*, *user=None*, *repo=None*)
Get a single comment

Parameters

- **number** (*str*) – The comment to get
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

list(*number*, *user=None*, *repo=None*)
List all the comments for a pull request

Parameters

- **number** (*str*) – The number of the pull request
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

update (*number*, *message*, *user=None*, *repo=None*)
Update a comment

Parameters

- **number** (*str*) – The id of the comment to update
- **message** (*str*) – Comment message
- **user** (*str*) – Username
- **repo** (*str*) – Repository

Note: Remember *Config precedence*

Orgs services

Fast sample:

```
from pygithub3 import Github

gh = Github(token='abc123')

auth_orgs = gh.orgs.list().all()
members = gh.orgs.members.list('github')
```

Org

class pygithub3.services.orgs.**Org** (***config*)
Consume Orgs API

members
Members

teams
Teams

get (*org*)
Get a single org

Parameters **org** (*str*) – Org name

list (*user=None*)
Get user's orgs

Parameters **user** (*str*) – Username

Returns A Result

If you call it without user and you are authenticated, get the authenticated user's orgs, public and private.

If you call it with a user, get the user's public orgs.

```
org_service.list('copitux')
org_service.list()
```

update(*org*, *data*)
Update a single org

Parameters

- **org** (*str*) – Org name
- **data** (*dict*) – Input. See github orgs doc

Warning: You must be authenticated

```
org_service.update(dict(company='ACME Development',
                        location='Timbuctoo'))
```

Members

class pygithub3.services.orgs.members.**Members**(***config*)
Consume Members API

conceal_membership(*org*, *user*)
Conceal user's membership in org

Parameters

- **org** (*str*) – Organisation name
- **user** (*str*) – User name

Warning: You must be authenticated and the user, or an owner of the org

is_member(*org*, *user*)
Determine if user is a member of org

Parameters

- **org** (*str*) – Organisation name
- **user** (*str*) – User name

is_public_member(*org*, *user*)
Determine if user is a public member of org

Parameters

- **org** (*str*) – Organisation name
- **user** (*str*) – User name

list(*org*)
Get org's members

Parameters **org** (*str*) – Organisation name

Returns A Result

If you call it authenticated, and are a member of the org, public and private members will be visible.

If not, only public members will be visible.

list_public(*org*)
Get org's public members

Parameters **org** (*str*) – Organisation name

Returns A Result

publicize_membership(*org*, *user*)
Publicize user's membership in org

Parameters

- **org** (*str*) – Organisation name
- **user** (*str*) – User name

> **Warning:** You must be authenticated and the user, or an owner of the org

remove_member(*org*, *user*)
Remove user from all teams in org

Parameters

- **org** (*str*) – Organisation name
- **user** (*str*) – User name

> **Warning:** You must be authenticated and an owner of org

Teams

class pygithub3.services.orgs.teams.**Teams**(***config*)
Consume Teams API

> **Warning:** You must be authenticated as an owner of the org

add_member(*id*, *user*)
Add a user to a team

Parameters

- **id** (*int*) – The team id
- **user** (*str*) – User name

add_repo(*id*, *user*, *repo*)
Give team members access to a repo

Parameters

- **id** (*int*) – The team id
- **user** (*str*) – User name
- **repo** (*str*) – Repo name

contains_repo(*id*, *user*, *repo*)
Determine if user is a member of a team

Parameters

- **id** (*int*) – The team id
- **user** (*str*) – User name

- **repo** (*str*) – Repo name

create(*org*, *data*)
Create a new team

Parameters

- **org** (*str*) – Organisation name
- **data** (*dict*) – Input. See github orgs teams doc

delete(*id*)
Delete a team

Parameters **id** (*int*) – The team id

get(*id*)
Get a team

Parameters **id** (*int*) – The team id

Returns A Result

is_member(*id*, *user*)
Determine if user is a member of a team

Parameters

- **id** (*int*) – The team id
- **user** (*str*) – User name

list(*org*)
Get org's teams

Parameters **org** (*str*) – Organisation name

Returns A Result

list_members(*id*)
List the members of a team

Parameters **id** (*int*) – The team id

Returns A Result

list_repos(*id*)
List the repos that a team's members get access to

Parameters **id** (*int*) – The team id

Returns A Result

remove_member(*id*, *user*)
Remove a member from a team

Parameters

- **id** (*int*) – The team id
- **user** (*str*) – User name

remove_repo(*id*, *user*, *repo*)
Remove a repo from the a team

Parameters

- **id** (*int*) – The team id

- **user** (*str*) – User name
- **repo** (*str*) – Repo name

update(*id*, *data*)
Update a team

Parameters

- **id** (*int*) – The team id
- **data** (*dict*) – Input. See github orgs teams doc

Issues services

Fast sample:

```
from pygithub3 import Github

auth = dict(login='octocat', password='pass')
gh = Github(**auth)

octocat_issues = gh.issues.list()
octocat_repo_issues = gh.issues.list_by_repo('octocat', 'Hello-World')
```

Issues

class pygithub3.services.issues.**Issue**(***config*)
Consume Issues API

comments
Comments

events
Events

labels
Labels

milestones
Milestones

create(*data*, *user=None*, *repo=None*)
Create an issue

Parameters

- **data** (*dict*) – Input. See github issues doc
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

```
        issues_service.create(dict(title='My test issue',
            body='This needs to be fixed ASAP.',
            assignee='copitux'))
```

get (*number*, *user=None*, *repo=None*)
Get a single issue

Parameters

- **number** (*int*) – Issue number
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Note: Remember *Config precedence*

list (***params*)
List your issues

Parameters

- **filter** (*str*) – 'assigned', 'created', 'mentioned' or 'subscribed'
- **state** (*str*) – 'open' or 'closed'
- **labels** (*str*) – List of comma separated Label names. e.g: bug,ui, @high
- **sort** (*str*) – 'created', 'updated' or 'comments'
- **direction** (*str*) – 'asc' or 'desc'
- **since** (*datetime*) – Date filter (datetime or str in ISO 8601)

Returns A Result

Warning: You must be authenticated

list_by_repo (*user=None*, *repo=None*, ***params*)
List issues for a repo

Parameters

- **milestone** (*str*) – Milestone ID, 'none' or '*'
- **state** (*str*) – 'open' or 'closed'
- **assignee** (*str*) – Username, 'none' or '*'
- **mentioned** (*str*) – Username
- **labels** (*str*) – List of comma separated Label names. e.g: bug,ui, @high
- **sort** (*str*) – 'created', 'updated' or 'comments'
- **direction** (*str*) – 'asc' or 'desc'
- **since** (*datetime*) – Date filter (datetime or str in ISO 8601)

Returns A Result

Note: Remember *Config precedence*

update (*number*, *data*, *user=None*, *repo=None*)
Update an issue

Parameters

- **number** (*int*) – Issue number
- **data** (*dict*) – Input. See github issues doc
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

Comments

class pygithub3.services.issues.**Comments**(***config*)
Consume Comments API

create (*number*, *message*, *user=None*, *repo=None*)
Create a comment on an issue

Parameters

- **number** (*int*) – Issue number
- **message** (*str*) – Comment message
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

delete (*id*, *user=None*, *repo=None*)
Delete a single comment

Parameters

- **id** (*int*) – Comment id
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

get (*id*, *user=None*, *repo=None*)
Get a single comment

Parameters

- **id** (*int*) – Comment id
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Note: Remember *Config precedence*

list (*number*, *user=None*, *repo=None*)
List comments for an issue

Parameters

- **number** (*int*) – Issue number
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

Note: Remember *Config precedence*

update (*id*, *message*, *user=None*, *repo=None*)
Update a comment on an issue

Parameters

- **id** (*int*) – Issue id
- **message** (*str*) – Comment message
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

Events

class pygithub3.services.issues.**Events** (***config*)
Consume Issues Events API

get (*id*, *user=None*, *repo=None*)
Get a single event

Parameters

- **id** (*int*) – Comment id
- **user** (*str*) – Username

- **repo** (*str*) – Repo name

list_by_issue(*number*, *user=None*, *repo=None*)
List events for an issue

Parameters

- **number** (*int*) – Issue number
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

list_by_repo(*user=None*, *repo=None*)
List events for a repository

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

Labels

class pygithub3.services.issues.**Labels**(***config*)
Consume Labels API

add_to_issue(*number*, *labels*, *user=None*, *repo=None*)
Add labels to issue

Parameters

- **number** (*int*) – Issue number
- **user** (*str*) – Username
- **repo** (*str*) – Repo name
- **labels** (*list*) – Label names

Returns A Result

Note: Remember *Config precedence*

```
labels_service.add_to_issue(2, user='github', repo='github',
    'label1', 'label2', 'label3')
```

create(*data*, *user=None*, *repo=None*)
Create a label on an repo

Parameters

- **data** (*dict*) – Input. See github labels doc
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

delete (*name*, *user=None*, *repo=None*)
Delete a label on an repo

Parameters

- **name** (*str*) – Label name
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

get (*name*, *user=None*, *repo=None*)
Get a single label

Parameters

- **name** (*str*) – Label name
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Note: Remember *Config precedence*

list (*user=None*, *repo=None*)
Get repository's labels

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

list_by_issue (*number*, *user=None*, *repo=None*)
List labels for an issue

Parameters

- **number** (*int*) – Issue number
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

Note: Remember *Config precedence*

list_by_milestone(*number*, *user=None*, *repo=None*)
Get labels for every issue in a milestone

Parameters

- **number** (*int*) – Milestone ID
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

Note: Remember *Config precedence*

remove_all(*number*, *user=None*, *repo=None*)
Remove all labels from a issue

Parameters

- **number** (*int*) – Issue number
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

Note: Remember *Config precedence*

remove_from_issue(*number*, *label*, *user=None*, *repo=None*)
Remove a label from an issue

Parameters

- **number** (*int*) – Issue number
- **label** (*str*) – Label name
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

Note: Remember *Config precedence*

replace_all(*number*, *labels*, *user=None*, *repo=None*)
Replace all labels for a issue

Parameters

- **number** (*int*) – Issue number
- **labels** (*list*) – New labels

- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Returns A Result

Note: If labels weren't especified, it'd remove all labels from the issue

Note: Remember *Config precedence*

update(*name*, *data*, *user=None*, *repo=None*)
Update a label on an repo

Parameters

- **name** (*str*) – Label name
- **data** (*dict*) – Input. See github labels doc
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

Milestones

class pygithub3.services.issues.**Milestones**(***config*)
Consume Milestones API

create(*data*, *user=None*, *repo=None*)
Create a milestone

Parameters

- **data** (*dict*) – Input. See github milestones doc
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

delete(*number*, *user=None*, *repo=None*)
Delete a milestone

Parameters

- **number** (*int*) – Milestone number

- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

get (*number*, *user=None*, *repo=None*)
Get a single milestone

Parameters

- **number** (*int*) – Milestone number
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Note: Remember *Config precedence*

list (*user=None*, *repo=None*, ***params*)
List milestones for a repo

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repo name
- **state** (*str*) – 'open' or 'closed'
- **sort** (*str*) – 'due_date' or 'completeness'
- **direction** (*str*) – 'asc' or 'desc'

Returns A Result

Note: Remember *Config precedence*

update (*number*, *data*, *user=None*, *repo=None*)
Update a milestone

Parameters

- **number** (*int*) – Milestone number
- **data** (*dict*) – Input. See github milestones doc
- **user** (*str*) – Username
- **repo** (*str*) – Repo name

Warning: You must be authenticated

Note: Remember *Config precedence*

Events service

This service exposes the Events API. Much of this API is read-only, and while pagination is supported, there is a fixed page size of 30 with a limit of 10 page requests.

Many events have an *actor* which denotes the user that performed an event. Additionally, there may be *org* or *repo* attributes for events related to Organizations and Repos. Finally, each event object contains a *payload* attribute containing more detailed information about the event.

Public Events

Yields the most recent public events from Github.

```
from pygithub3 import Github

gh = Github()

events = gh.events.list().all()
print events[0].payload
```

Event

class pygithub3.services.events.**Event**(***config*)

Consume Events API

The events API supports pagination, but with a fixed page size of 30; In addition, fetching up to ten pages is supported, for a total of 300 events.

issues

Issues events

networks

Network

orgs

Org

repos

Repo

users

User

list()

List all public events.

Returns A Result

Note: Hits the API fetching maximum number of events (300 = 30/page * 10)

Repo Events

These are events for a specific repo, including issue and network events. The Issues events are proxied to the *Issues services*.

```
events = gh.events.repos.list(user="copitux", repo="python-github3")
for e in events.next():
    print("{t}".format(t=e.type))

# Get the issue Events
events = gh.events.issues.list_by_repo(user="copitux",
                                       repo="python-github3")

# Get the Public Events for a Repo's Network
events = gh.events.networks.list(user="copitux", repo="python-github3")
```

Network

class pygithub3.services.events.networks.**Network**(***config*)

Consume a Repo's public Network Events API

list(*user=None*, *repo=None*)

List the events for a repository's Network

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

Repo

class pygithub3.services.events.repos.**Repo**(***config*)

Consume Repo Events API

list(*user=None*, *repo=None*)

Get repository's events

Parameters

- **user** (*str*) – Username
- **repo** (*str*) – Repository

Returns A Result

Note: Remember *Config precedence*

Note: Remember that the Events API give 10 pages with 30 entries, up to 300 events, so you'll only get the last 300 Repo events

Organization Events

These are the public events for an Organization

```
events = gh.events.orgs.list(org="Acme")
```

You may also get a user's feed of events for an Organization, but you *must* be authenticated as the provided user, and you must be a member of the given organization.

```
events = gh.events.users.orgs(user="copitux", org="acme")
```

Org

class pygithub3.services.events.orgs.**Org**(***config*)

Consume a Org Events API

list(*org=None*)

List the events for an Organization

Parameters **org** (*str*) – Organization name

Returns A Result

User Events

You can retrieve the public events performed by a user and the public events that a user receives. If you're authenticated, you may also receive private events.

```
received_events = gh.events.users.list_received_public(user="copitux")
performed_events = gh.events.users.list_performed_public(user="copitux")
```

If authenticated as *copitux*, you could get private events with the following, otherwise you'll just get the public events as above:

```
received_events = gh.events.users.list_received(user="copitux")
performed_events = gh.events.users.list_performed(user="copitux")
```

User

class pygithub3.services.events.users.**User**(***config*)

Consume User Events API

list_performed(*user=None*)

List the events performed by a given user. If authenticated as the given user, you'll see private events as well as public events.

Parameters **user** (*str*) – Username

Returns A Result

Note: Remember *Config precedence*

---

list_performed_public(*user=None*)

List only the public events performed by the given user.

Parameters **user** (*str*) – Username

Returns A Result

Note: Remember *Config precedence*

---

list_received(*user=None*)
List the events received by a given user. If authenticated as the given user, you'll see private events as well as public events.

Parameters **user** (*str*) – Username

Returns A Result

Note: Remember *Config precedence*

list_received_public(*user=None*)
List only the public events received by the given user.

Parameters **user** (*str*) – Username

Returns A Result

Note: Remember *Config precedence*

orgs(*user=None*, *org=None*)
List the events for the User's organization dashboard.

Parameters **user** (*str*) – Username

Returns A Result

Warning: You must be authenticated as the given user.

Note: Remember *Config precedence*

2.4 Result

Some requests returns multiple Resources, for that reason the `Github API` paginate it and **pygithub3** too

2.4.1 Smart Result

class pygithub3.core.result.smart.**Result**(*method*)
It's a very **lazy** paginator beacuse only do a real request when is needed, besides it's **cached**, so never repeats a request.

You have several ways to consume it

1. Iterating over the result:

```
result = some_request()
for page in result:
    for resource in page:
        print resource
```

2. With a generator:

```
result = some_request()
for resource in result.iterator():
    print resource
```

3.As a list:

```
result = some_request()
print result.all()
```

4.Also you can request some page manually

Each `Page` is an iterator and contains resources:

```
result = some_request()
assert result.pages > 3
page3 = result.get_page(3)
page3_resources = list(page3)
```

get_page(*page*)
Get `Page` of resources

Parameters **page** (*int*) – Page number

pages
Total number of pages in request

2.4.2 Normal Result

class pygithub3.core.result.normal.**Result**(*method*)
It's a middle-lazy iterator, because to get a new page it needs make a real request, besides it's **cached**, so never repeats a request.

You have several ways to consume it

1.Iterating over the result:

```
result = some_request()
for page in result:
    for resource in page:
        print resource
```

2.With a generator:

```
result = some_request()
for resource in result.iterator():
    print resource
```

3.As a list:

```
result = some_request()
print result.all()
```

2.5 Resources

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W