
octonode

octonode is a library for nodejs to access the github v3 api

Installation

```
1 npm install octonode
```

Usage

```
1 var github = require('octonode');
2
3 // Then we instantiate a client with or without a token (as show in a
  later section)
4
5 var ghme = client.me();
6 var ghuser = client.user('pksunkara');
7 var ghrepo = client.repo('pksunkara/hub');
8 var ghorg = client.org('flatiron');
9 var ghissue = client.issue('pksunkara/hub', 37);
10 var ghmilestone = client.milestone('pksunkara/hub', 37);
11 var ghlabel = client.label('pksunkara/hub', 'todo');
12 var ghpr = client.pr('pksunkara/hub', 37);
13 var ghrelease = client.release('pksunkara/hub', 37);
14 var ghgist = client.gist();
15 var ghteam = client.team(37);
16 var ghproject = client.project(37);
17 var ghnotification = client.notification(37);
18
19 var ghsearch = client.search();
```

Build a client which accesses any public information

```
1 var client = github.client();
2
3 client.get('/users/pksunkara', {}, function (err, status, body, headers
  ) {
4   console.log(body); //json object
5 });
```

Build a client from an access token

```
1 var client = github.client('some_access_token');
2
3 client.get('/user', {}, function (err, status, body, headers) {
4   console.log(body); //json object
5 });
```

Build a client from a different host You can configure the `protocol`, `hostname` and `port` to use. For example to connect to a GitHub Enterprise instance.

```
1 var client = github.client('some_access_token', {
2   hostname: 'mydomain.com/api/v3'
3 });
4
5 client.get('/user', {}, function (err, status, body, headers) {
6   console.log(body); //json object
7 });
```

Request Options

Request options can be set by setting defaults on the client. (e.g. Proxies)

```
1 var client = github.client();
2
3 client.requestDefaults['proxy'] = 'https://myproxy.com:1085'
```

These options are passed through to `request`, see their API here: <https://github.com/request/request#requestoptions> callback

Proxies You can set proxies dynamically by using the example above, but Octonode will respect environment proxies by default. Just set this using: `export HTTP_PROXY='https://myproxy.com:1085'` if you are using the command line

Many of the below use cases use parts of the above code

Conditional requests

The client supports conditional requests and helps you respecting rate limits by caching information that hasn't changed. You can add the `If-None-Match` header to each request calling the `conditional` method.

```
1 var client = github.client();
2
3 // This add If-None-Match header to the request
4 github.repo().conditional('ETAG').issues();
```

More info about conditional requests can be founded here.

Authentication

Authenticate to github in cli mode (desktop application) **Note:** Ensure that the scopes argument is an object containing the required `note` property. For two-factor authentication add the One Time Password `otp` key with its corresponding code to the configuration object.

```
1 var scopes = {
2   'scopes': ['user', 'repo', 'gist'],
3   'note': 'admin script'
4 };
5
6 github.auth.config({
7   username: 'pksunkara',
8   password: 'password'
9 }).login(scopes, function (err, id, token, headers) {
10   console.log(id, token);
11 });
```

Authenticate to github in web mode (web application)

```
1 // Web application which authenticates to github
2 var http = require('http')
3   , url = require('url')
4   , qs = require('querystring')
5   , github = require('octonode');
6
7 // Build the authorization config and url
8 var auth_url = github.auth.config({
9   id: 'mygithubclientid',
10  secret: 'mygithubclientsecret',
11  apiUrl: 'https://optional-internal-github-enterprise/api/v3',
12  webUrl: 'https://optional-internal-github-enterprise'
13 }).login(['user', 'repo', 'gist']);
14
15 // Store info to verify against CSRF
16 var state = auth_url.match(/&state=([0-9a-z]{32})/i);
17
18 // Web server
19 http.createServer(function (req, res) {
20   uri = url.parse(req.url);
21   // Redirect to github login
22   if (uri.pathname== '/login') {
23     res.writeHead(302, {'Content-Type': 'text/plain', 'Location':
24       auth_url});
25     res.end('Redirecting to ' + auth_url);
26   }
27   // Callback url from github login
28   else if (uri.pathname== '/auth') {
29     var values = qs.parse(uri.query);
30     // Check against CSRF attacks
```

```
30     if (!state || state[1] !== values.state) {
31         res.writeHead(403, {'Content-Type': 'text/plain'});
32         res.end('');
33     } else {
34         github.auth.login(values.code, function (err, token, headers) {
35             res.writeHead(200, {'Content-Type': 'text/plain'});
36             res.end(token);
37         });
38     }
39 } else {
40     res.writeHead(200, {'Content-Type': 'text/plain'})
41     res.end('');
42 }
43 }).listen(3000);
44
45 console.log('Server started on 3000');
```

Rate Limiting

You can also check your rate limit status by calling the following.

```
1 client.limit(function (err, left, max, reset) {
2     console.log(left); // 4999
3     console.log(max);  // 5000
4     console.log(reset); // 1372700873 (UTC epoch seconds)
5 });
```

API Callback Structure

All the callbacks for the following will take first an error argument, then a data argument, like this:

```
1 ghme.info(function(err, data, headers) {
2     console.log("error: " + err);
3     console.log("data: " + data);
4     console.log("headers:" + headers);
5 });
```

Async / Promises

If you would like to work with promises rather than callbacks, you can call the promise based version of any of the api calls by appending `Async` to the function call.

For example `prs()` becomes `prsAsync()` like this:

```
1 async function getPullRequests () {
2   const client = github.client(config.githubAccessToken)
3   const repo = client.repo('pksunkara/octonode')
4
5   const result = await repo.prsAsync({ per_page: 100 })
6   return result[0]
7 }
```

Pagination

If a function is said to be supporting pagination, then that function can be used in many ways as shown below. Results from the function are arranged in pages.

The page argument is optional and is used to specify which page of issues to retrieve. The perPage argument is also optional and is used to specify how many issues per page.

```
1 // Normal usage of function
2 ghrepo.issues(callback); //array of first 30 issues
3
4 // Using pagination parameters
5 ghrepo.issues(2, 100, callback); //array of second 100 issues
6 ghrepo.issues(10, callback); //array of 30 issues from page 10
7
8 // Pagination parameters can be set with query object too
9 ghrepo.issues({
10   page: 2,
11   per_page: 100, //maximum is 100
12   state: 'closed'
13 }, callback); //array of second 100 issues which are closed
```

Github authenticated user api

Token/Credentials required for the following:

Get information about the user (GET /user)

```
1 ghme.info(callback); //json
```

Update user profile (PATCH /user)

```
1 ghme.update({
2   "name": "monalisa octocat",
3   "email": "octocat@github.com",
4 }, callback);
```

Get emails of the user (GET /user/emails)

```
1 ghme.emails(callback); //array of emails
```

Set emails of the user (POST /user/emails)

```
1 ghme.emails(['new1@ma.il', 'new2@ma.il'], callback); //array of emails
2 ghme.emails('new@ma.il', callback); //array of emails
```

Delete emails of the user (DELETE /user/emails)

```
1 ghme.emails(['new1@ma.il', 'new2@ma.il']);
2 ghme.emails('new@ma.il');
```

Get the followers of the user (GET /user/followers)

```
1 ghme.followers(callback); //array of github users
```

Get users whom the user is following (GET /user/following) This query supports pagination.

```
1 ghme.following(callback); //array of github users
```

Check if the user is following a user (GET /user/following/marak)

```
1 ghme.following('marak', callback); //boolean
```

Follow a user (PUT /user/following/marak)

```
1 ghme.follow('marak');
```

Unfollow a user (DELETE /user/following/marak)

```
1 ghme.unfollow('marak');
```

Get public keys of a user (GET /user/keys)

```
1 ghme.keys(callback); //array of keys
```

Get a single public key (GET /user/keys/1)

```
1 ghme.keys(1, callback); //key
```

Create a public key (POST /user/keys)

```
1 ghme.keys({"title":"laptop", "key":"ssh-rsa AAA..."}, callback); //key
```

Update a public key (PATCH /user/keys/1)

```
1 ghme.keys(1, {"title":"desktop", "key":"ssh-rsa AAA..."}, callback); //
  key
```

Delete a public key (DELETE /user/keys/1)

```
1 ghme.keys(1);
```

Get the starred repos for the user (GET /user/starred) This query supports pagination.

```
1 ghme.starred(callback); //array of repos
```

Check if you have starred a repository (GET /user/starred/pksunkara/octonode)

```
1 ghme.checkStarred('flatiron/flatiron', callback); //boolean
```

Star a repository (PUT /user/starred/pksunkara/octonode)

```
1 ghme.star('flatiron/flatiron');
```

Unstar a repository (DELETE /user/starred/pksunkara/octonode)

```
1 ghme.unstar('flatiron/flatiron');
```

Get the subscriptions of the user (GET /user/subscriptions) This query supports pagination.

```
1 ghme.watched(callback); //array of repos
```

List your public and private organizations (GET /user/orgs) This query supports pagination.

```
1 ghme.orgs(callback); //array of orgs
```

List your repositories (GET /user/repos) This query supports pagination.

```
1 ghme.repos(callback); //array of repos
```

Create a repository (POST /user/repos)

```
1 ghme.repo({
2   "name": "Hello-World",
3   "description": "This is your first repo",
4 }, callback); //repo
```

Fork a repository (POST /repos/pksunkara/hub/forks)

```
1 ghme.fork('pksunkara/hub', callback); //forked repo
```

List all issues across owned and member repositories (GET /user/issues) This query supports pagination.

```
1 ghme.issues({
2   page: 2,
3   per_page: 100,
4   filter: 'assigned',
5   state: 'open',
6   sort: 'created'
7 }, callback); //array of issues
```

List user teams (GET /user/teams) This query supports pagination.

```
1 ghme.teams({
2   page: 1,
3   per_page: 50
4 }, callback); //array of team memberships
```

List notifications Options based on <http://git.io/vYYOx>

```
1 ghme.notifications({}, callback); //array of notifications
```

Github users api

No token required for the following

Get information about a user (GET /users/pksunkara)

```
1 ghuser.info(callback); //json
```

Get user followers (GET /users/pksunkara/followers) This query supports pagination.

```
1 ghuser.followers(callback); //array of github users
```

Get user followings (GET /users/pksunkara/following) This query supports pagination.

```
1 ghuser.following(callback); //array of github users
```

Get user public repos (GET /users/pksunkara/repos) This query supports pagination.

```
1 ghuser.repos(callback); //array of public github repos
```

Get events performed by a user (GET /users/pksunkara/events) This query supports pagination.

Optionally, supply an array of Event Types to filter by.

```
1 ghuser.events(['PushEvent'], callback); //array of PushEvent events
```

Or leave it out to get all Event Types.

```
1 ghuser.events(callback); //array of events
```

Get user public organizations (GET /users/pksunkara/orgs) This query supports pagination.

```
1 ghuser.orgs(callback); //array of organizations
```

Github repositories api

Get information about a repository (GET /repos/pksunkara/hub)

```
1 ghrepo.info(callback); //json
```

Edit a repository (PATCH /repos/pksunkara/hub)

```
1 ghrepo.update({
2   private: false
3 }, callback); // repo
```

Get the collaborators for a repository (GET /repos/pksunkara/hub/collaborators)

```
1 ghrepo.collaborators(callback); //array of github users
```

Check if a user is collaborator for a repository (GET /repos/pksunkara/hub/collaborators/marak)

```
1 ghrepo.collaborators('marak', callback); //boolean
```

Get the commits for a repository (GET /repos/pksunkara/hub/commits)

```
1 ghrepo.commits(callback); //array of commits
```

Get a certain commit for a repository (GET /repos/pksunkara/hub/commits/18293abcd72)

```
1 ghrepo.commit('18293abcd72', callback); //commit
```

Get a comparison between branches for a repository (GET /repos/pksunkara/hub/compare/master...develop)

```
1 ghrepo.compare('master', 'develop', callback); //compare develop to
  master
```

Get the tags for a repository (GET /repos/pksunkara/hub/tags)

```
1 ghrepo.tags(callback); //array of tags
```

Get the releases for a repository (GET /repos/pksunkara/hub/releases)

```
1 ghrepo.releases(callback); //array of releases
```

Get the languages for a repository (GET /repos/pksunkara/hub/languages)

```
1 ghrepo.languages(callback); //array of languages
```

Get the contributors for a repository (GET /repos/pksunkara/hub/contributors)

```
1 ghrepo.contributors(callback); //array of github users
```

Get the branches for a repository (GET /repos/pksunkara/hub/branches) This query supports pagination.

```
1 ghrepo.branches(callback); //array of branches
```

Get a branch for a repository (GET /repos/pksunkara/hub/branches/master)

```
1 ghrepo.branch('master', callback); //branch
```

Create a Reference (POST /repos/pksunkara/hub/git/refs)

```
1 ghrepo.createReference('master', '18293abcd72', callback);
```

Get the issues for a repository (GET /repos/pksunkara/hub/issues) This query supports pagination.

```
1 ghrepo.issues(callback); //array of issues
```

Create an issue for a repository (POST /repos/pksunkara/hub/issues)

```
1 ghrepo.issue({
2   "title": "Found a bug",
3   "body": "I'm having a problem with this.",
4   "assignee": "octocat",
5   "milestone": 1,
6   "labels": ["Label1", "Label2"]
7 }, callback); //issue
```

Get the milestones for a repository (GET /repos/pksunkara/hub/milestones) This query supports pagination.

```
1 ghrepo.milestones(callback); //array of milestones
```

Create a milestone for a repository (POST /repos/pksunkara/hub/milestones)

```
1 ghrepo.milestone({
2   "title": "Sprint 345",
3   "description": "The sprint where we fix all the things!",
4   "due_on": new Date(2014, 7, 1)
5 }, callback); //milestone
```

Get the projects for a repository (GET /repos/pksunkara/hub/projects) This query supports pagination.

```
1 ghrepo.projects(callback); //array of projects
```

Create a project for a repository (POST /repos/pksunkara/hub/projects)

```
1 ghrepo.project({
2   "name": "Sprint 345",
3   "body": "The sprint where we fix all the things!"
4 }, callback); //project
```

Get the labels for a repository (GET /repos/pksunkara/hub/labels) This query supports pagination.

```
1 ghrepo.labels(callback); //array of labels
```

Create a label for a repository (POST /repos/pksunkara/hub/labels)

```
1 ghrepo.label({
2   "name": "Priority",
3   "color": "ff0000",
4 }, callback); //label
```

Get the pull requests for a repository (GET /repos/pksunkara/hub/pulls) This query supports pagination.

```
1 ghrepo.prs(callback); //array of pull requests
```

Create a pull request (POST /repos/pksunkara/hub/pulls)

```
1 ghrepo.pr({
2   "title": "Amazing new feature",
3   "body": "Please pull this in!",
4   "head": "octocat:new-feature",
```

```
5   "base": "master"
6 }, callback); //pull request
```

Get the hooks for a repository (GET /repos/pksunkara/hub/hooks) This query supports pagination.

```
1 ghrepo.hooks(callback); //array of hooks
```

Create a hook (POST /repos/pksunkara/hub/hooks)

```
1 ghrepo.hook({
2   "name": "web",
3   "active": true,
4   "events": ["push", "pull_request"],
5   "config": {
6     "url": "http://myawesomesite.com/github/events"
7   }
8 }, callback); // hook
```

Delete a hook (DELETE /repos/pksunkara/hub/hooks/37)

```
1 ghrepo.deleteHook(37, callback);
```

Get the README for a repository (GET /repos/pksunkara/hub/readme)

```
1 ghrepo.readme(callback); //file
2 ghrepo.readme('v0.1.0', callback); //file
```

Get the root contents on a branch called "myBranch"

```
1 ghrepo.contents('', "myBranch", callback);
```

Get the contents of a path in repository

```
1 ghrepo.contents('lib/index.js', callback); //path
2 ghrepo.contents('lib/index.js', 'v0.1.0', callback); //path
```

Create a file at a path in repository

```
1 ghrepo.createContents('lib/index.js', 'commit message', 'content',
2   callback); //path
3 ghrepo.createContents('lib/index.js', 'commit message', 'content', 'v0
4   .1.0', callback); //path
```

Update a file at a path in repository

```
1 ghrepo.updateContents('lib/index.js', 'commit message', 'content', 'put
2   -sha-here', callback); //path
3 ghrepo.updateContents('lib/index.js', 'commit message', 'content', 'put
4   -sha-here', 'master', callback); //path
```

```
3 ghrepo.updateContents('lib/index.js', 'commit message', 'content', 'put
  -sha-here', 'v0.1.0', callback); //path
```

Delete a file at a path in repository

```
1 ghrepo.deleteContents('lib/index.js', 'commit message', 'put-sha-here',
  callback); //path
2 ghrepo.deleteContents('lib/index.js', 'commit message', 'put-sha-here',
  'v0.1.0', callback); //path
```

Get archive link for a repository

```
1 ghrepo.archive('tarball', callback); //link to archive
2 ghrepo.archive('zipball', 'v0.1.0', callback); //link to archive
```

Get the blob for a repository (GET /repos/pksunkara/hub/git/blobs/SHA)

```
1 ghrepo.blob('18293abcd72', callback); //blob
```

Get users who starred a repository (GET /repos/pksunkara/hub/stargazers)

```
1 ghrepo.stargazers(1, 100, callback); //array of users
2 ghrepo.stargazers(10, callback);      //array of users
3 ghrepo.stargazers(callback);          //array of users
```

Get the teams for a repository (GET /repos/pksunkara/hub/teams)

```
1 ghrepo.teams(callback); //array of teams
```

Get a git tree (GET /repos/pksunkara/hub/git/trees/18293abcd72)

```
1 ghrepo.tree('18293abcd72', callback); //tree
2 ghrepo.tree('18293abcd72', true, callback); //recursive tree
```

Delete the repository (DELETE /repos/pksunkara/hub)

```
1 ghrepo.destroy();
```

List statuses for a specific ref (GET /repos/pksunkara/hub/statuses/master)

```
1 ghrepo.statuses('master', callback); //array of statuses
```

List the combined status for a specific ref (GET /repos/pksunkara/hub/commits/master/statuses)

```
1 ghrepo.combinedStatus('master', callback); //array of statuses
```

Create status (POST /repos/pksunkara/hub/statuses/SHA)

```
1 ghrepo.status('18e129c213848c7f239b93fe5c67971a64f183ff', {
2   "state": "success",
3   "target_url": "http://ci.mycompany.com/job/hub/3",
```

```
4   "description": "Build success."
5 }, callback); // created status
```

GitHub notifications api

Mark a thread as read

```
1 ghnotification.markAsRead(callback);
```

Subscribe to a thread

```
1 ghnotification.subscribe(callback);
```

Unsubscribe from a thread

```
1 ghnotification.unsubscribe(callback);
```

Mute a thread

```
1 ghnotification.mute(callback);
```

Github organizations api

Get information about an organization (GET /orgs/flatiron)

```
1 ghorg.info(callback); //json
```

Update an organization (POST /orgs/flatiron)

```
1 ghorg.update({
2   blog: 'https://blog.com'
3 }, callback); // org
```

List organization repositories (GET /orgs/flatiron/repos) This query supports pagination.

```
1 ghorg.repos(callback); //array of repos
```

Create an organization repository (POST /orgs/flatiron/repos)

```
1 ghorg.repo({
2   name: 'Hello-world',
3   description: 'My first world program'
4 }, callback); //repo
```

Get an organization's teams (GET /orgs/flatiron/teams) This query supports pagination.

```
1 ghorg.teams(callback); //array of teams
```

Get an organization's members (GET /orgs/flatiron/members) This query supports pagination.

```
1 ghorg.members(callback); //array of github users
```

Check an organization member (GET /orgs/flatiron/members/pksunkara)

```
1 ghorg.member('pksunkara', callback); //boolean
```

Check a member's public membership in an org (GET /orgs/flatiron/public_members/pksunkara)

```
1 ghorg.publicMember('pksunkara', callback); //boolean
```

Publicize a user's membership (PUT /orgs/flatiron/public_members/pksunkara)

```
1 ghorg.publicizeMembership('pksunkara', callback);
```

Conceal a user's membership (DELETE /orgs/flatiron/public_members/pksunkara)

```
1 ghorg.concealMembership('pksunkara', callback);
```

Check a member's membership status (GET /orgs/flatiron/memberships/pksunkara)

```
1 ghorg.membership('pksunkara', callback); //membership status object
   indicating state, role, etc.
```

Create an organization team (POST /orgs/flatiron/teams)

```
1 ghorg.createTeam({
2   "name": "new team name",
3   "permission": "push",
4   "repo_names": [
5     "flatiron/utile"
6   ]
7 }, callback);
```

Get the hooks for a Organization (GET /orgs/flatiron/hooks) This query supports pagination.

```
1 ghorg.hooks(callback); //array of hooks
```

Create a hook (POST /orgs/flatiron/hooks)

```
1 ghorg.hook({
2   "name": "web",
3   "active": true,
4   "events": ["push", "pull_request"],
5   "config": {
6     "url": "http://myawesomesite.com/github/events"
7   }
8 })
```

```
8 }, callback); // hook
```

Delete a hook (DELETE /orgs/flatiron/hooks/37)

```
1 ghorg.deleteHook(37, callback);
```

Github issues api

Get a single issue (GET /repos/pksunkara/hub/issues/37)

```
1 ghissue.info(callback); //issue
```

Edit an issue for a repository (PATCH /repos/pksunkara/hub/issues/37)

```
1 ghissue.update({
2   "title": "Found a bug and I am serious",
3 }, callback); //issue
```

List comments on an issue (GET /repos/pksunkara/hub/issues/37/comments) This query supports pagination.

```
1 ghissue.comments(callback); //array of comments
```

Create a comment (POST /repos/pksunkara/hub/issues/37/comments)

```
1 ghissue.createComment({
2   body: 'Me too.'
3 }, callback);
```

Edit a comment (PATCH /repos/pksunkara/hub/issues/comments/3)

```
1 ghissue.updateComment(3, {
2   body: 'The updated body of the comment.'
3 }, callback);
```

Delete a comment (DELETE /repos/pksunkara/hub/issues/comments/3)

```
1 ghissue.deleteComment(3, callback);
```

List labels on an issue (GET /repos/pksunkara/hub/issues/37/labels)

```
1 ghissue.labels(callback);
```

Add label(s) (POST /repos/pksunkara/hub/issues/37/labels)

```
1 ghissue.addLabels(['label-name'], callback);
```

Replace all labels (PUT /repos/pksunkara/hub/issues/37/labels)

```
1 ghissue.replaceAllLabels(['label-name'], callback);
```

Remove a single label (DELETE /repos/pksunkara/hub/issues/37/labels/label-name)

```
1 ghissue.removeLabel('label-name', callback);
```

Remove all labels (DELETE /repos/pksunkara/hub/issues/37/labels)

```
1 ghissue.removeAllLabels(callback);
```

Github milestones api

Get a single milestone (GET /repos/pksunkara/hub/milestones/37)

```
1 ghmilestone.info(callback); //milestone
```

Edit a milestone for a repository (PATCH /repos/pksunkara/hub/milestones/37)

```
1 ghmilestone.update({
2   "title": "Updated milestone title",
3 }, callback); //milestone
```

Delete a milestone for a repository (DELETE /repos/pksunkara/hub/milestones/37)

```
1 ghmilestone.delete(callback); //milestone
```

Github projects api

Get a single project (GET /projects/37)

```
1 ghproject.info(callback); //project
```

Edit a project for a repository (PATCH /projects/37)

```
1 ghproject.update({
2   "name": "Updated project name",
3 }, callback); //project
```

Delete a project for a repository (DELETE /projects/37)

```
1 ghproject.delete(callback); //project
```

Github labels api

Get a single label (GET /repos/pksunkara/hub/labels/todo)

```
1 ghlabel.info(callback); //label
```

Edit a label for a repository (PATCH /repos/pksunkara/hub/labels/todo)

```
1 ghlabel.update({
2   "color": "000000",
3 }, callback); //label
```

Delete a label for a repository (PATCH /repos/pksunkara/hub/labels/todo)

```
1 ghlabel.delete(callback); //label
```

Github Merge API**Merge a branch (POST /repos/pksunkara/hub/merges)**

```
1 ghrepo.merge({ base: baseBranch, head: destinationBranch }, callback);
```

Github pull requests api**Get a single pull request (GET /repos/pksunkara/hub/pulls/37)**

```
1 ghpr.info(callback); //pull request
```

Update a pull request (PATCH /repos/pksunkara/hub/pulls/37)

```
1 ghpr.update({
2   'title': 'Wow this pr'
3 }, callback); //pull request
```

Close a pull request

```
1 ghpr.close(callback); //pull request
```

Get if a pull request has been merged (GET /repos/pksunkara/hub/pulls/37/merged)

```
1 ghpr.merged(callback); //boolean
```

List commits on a pull request (GET /repos/pksunkara/hub/pulls/37/commits)

```
1 ghpr.commits(callback); //array of commits
```

List comments on a pull request (GET /repos/pksunkara/hub/pulls/37/comments)

```
1 ghpr.comments(callback); //array of comments
```

Add a comment on a pull request (POST /repos/pksunkara/hub/pulls/37/comments)

```
1 ghpr.createComment({
2   body: 'my comment',
3   commit_id: '8cde3b6c5be2c3067cd87ee4117c0f65e30f3e1f', // needed to
    comment on current time in PR
4   path: 'file.txt', // optional
5   position: 4 // optional
6 }, callback);
```

Remove a comment on a pull request (DELETE /repos/pksunkara/hub/pulls/37/comments/104)

```
1 ghpr.removecomment(104, callback);
```

List files in pull request (GET /repos/pksunkara/hub/pulls/37/files)

```
1 ghpr.files(callback); //array of files
```

List pull request reviews (GET /repos/pksunkara/hub/pulls/37/reviews)

```
1 ghpr.reviews(callback); //array of pull request reviews
```

Get a single pull request review (GET /repos/pksunkara/hub/pulls/37/reviews/104)

```
1 ghpr.review(104, callback); //pull request review
```

Delete a *pending* pull request review (DELETE /repos/pksunkara/hub/pulls/37/reviews/104)

```
1 ghpr.removeReview(104, callback); //pull request review
```

List comments for a pull request review (GET /repos/pksunkara/hub/pulls/37/reviews/104/comments)

```
1 ghpr.reviewComments(104, callback); //array of review comments
```

Create a pull request review (POST /repos/pksunkara/hub/pulls/37/reviews)

```
1 ghpr.createReview({
2   body: 'review message', // optional
3   comments: [ // optional
4     {
5       body: 'comment message', // required for each optional comment
6       path: 'file.txt', // required for each optional comment
7       position: 4 // required for each optional comment
8     }
9   ],
10  event: 'APPROVE || COMMENT || REQUEST_CHANGES' // optional
11 }, callback); //pull request review
```

Submit a pull request review (POST /repos/pksunkara/hub/pulls/37/reviews/104/events)

```
1 ghpr.submitReview(104, {
2   body: 'review message', // optional
3   event: 'APPROVE || COMMENT || REQUEST_CHANGES' // required
4 }, callback); //pull request review
```

Dismiss a pull request review (PUT /repos/pksunkara/hub/pulls/37/reviews/104/dismissals)

```
1 ghpr.dismissReview(104, 'dismissal message', callback); //pull request review
```

List review requests (GET /repos/pksunkara/hub/pulls/37/requested_reviewers)

```
1 ghpr.reviewRequests(callback); //array of review requests
```

Create review request(s) (POST /repos/pksunkara/hub/pulls/37/requested_reviewers)

```
1 ghpr.createReviewRequests(['user1', 'user2'], callback); //pull request
```

Delete review request(s) (DELETE /repos/pksunkara/hub/pulls/37/requested_reviewers)

```
1 ghpr.removeReviewRequests(['user1', 'user2'], callback); //pull request
```

Github releases api

Create release (POST /repos/pksunkara/releases)

```
1 ghrepo.release({
2   tag_name: 'v1.0.0',
3   draft: true
4 }, callback);
```

Upload assets in a release (POST /uploads.github.com/repos/pksunkara/hub/releases/37/assets?name=archive-v0.0.1.zip)

```
1 var archive = fs.readFileSync(__dirname, 'archive-v0.0.1.zip');
2 // Will upload a file with the default options
3 /*
4 {
5   name: 'archive.zip',
6   contentType: 'application/zip',
7   uploadHost: 'uploads.github.com'
8 }
9 */
10 ghrelease.uploadAssets(archive, callback);
11
12 // Will upload a file with your custom options
13 var image = fs.readFileSync(__dirname, 'octonode.png');
14 var options = {
15   name: 'octonode.png',
16   contentType: 'image/png',
17   uploadHost: 'uploads.github.com'
18 };
19 ghrelease.uploadAssets(image, options, callback)
```

Github gists api

List authenticated user's gists (GET /gists) This query supports pagination.

```
1 ghgist.list(callback); //array of gists
```

List authenticated user's public gists (GET /gists/public) This query supports pagination.

```
1 ghgist.public(callback); //array of gists
```

List authenticated user's starred gists (GET /gists/starred) This query supports pagination.

```
1 ghgist.starred(callback); //array of gists
```

List a user's public gists (GET /users/pksunkara/gists) This query supports pagination.

```
1 ghgist.user('pksunkara', callback); //array of gists
```

Get a single gist (GET /gists/37)

```
1 ghgist.get(37, callback); //gist
```

Create a gist (POST /gists)

```
1 ghgist.create({  
2   description: "the description",  
3   files: { ... }  
4 }, callback); //gist
```

Edit a gist (PATCH /gists/37)

```
1 ghgist.edit(37, {  
2   description: "hello gist"  
3 }, callback); //gist
```

Delete a gist (DELETE /gists/37)

```
1 ghgist.delete(37);
```

Fork a gist (POST /gists/37/forks)

```
1 ghgist.fork(37, callback); //gist
```

Star a gist (PUT /gists/37/star)

```
1 ghgist.star(37);
```

Unstar a gist (DELETE /gists/37/unstar)

```
1 ghgist.unstar(37);
```

Check if a gist is starred (GET /gists/37/star)

```
1 ghgist.check(37); //boolean
```

List comments on a gist (GET /gists/37/comments)

```
1 ghgist.comments(37, callback); //array of comments
```

Create a comment (POST /gists/37/comments)

```
1 ghgist.comments(37, {  
2   body: "Just commenting"  
3 }, callback); //comment
```

Get a single comment (GET /gists/comments/1)

```
1 ghgist.comment(1, callback); //comment
```

Edit a comment (POST /gists/comments/1)

```
1 ghgist.comment(1, {  
2   body: "lol at commenting"  
3 }, callback); //comment
```

Delete a comment (DELETE /gists/comments/1)

```
1 ghgist.comment(1);
```

Github teams api

Get a team (GET /team/37)

```
1 ghteam.info(callback); //json
```

Get the team members (GET /team/37/members)

```
1 ghteam.members(callback); //array of github users
```

Check if a user is part of the team (GET /team/37/members/pksunkara)

```
1 ghteam.member('pksunkara', callback); //boolean
```

Add a user to a team (PUT /team/37/members/pksunkara)

```
1 ghteam.addUser("pksunkara", callback); //boolean
```

Remove a user from a team (DELETE /team/37/members/pksunkara)

```
1 ghteam.removeUser("pksunkara", callback); //boolean
```

Get team membership (GET /teams/37/memberships/pksunkara)

```
1 ghteam.membership("pksunkara", callback); //boolean
```

Add team membership (PUT /teams/37/memberships/pksunkara)

```
1 ghteam.addMembership("pksunkara", callback); //boolean
```

Remove team membership (DELETE /team/37/memberships/pksunkara/)

```
1 ghteam.removeMembership("pksunkara", callback); //boolean
```

List all repos of a team (GET /team/37/repos/)

```
1 ghteam.repos(callback); //array of repos
```

Remove a repo from a team (DELETE /team/37/repos/flatiron/hub/)

```
1 ghteam.removeRepo("flatiron/hub", callback);
```

Delete a team (DELETE /team/37/)

```
1 ghteam.destroy(callback);
```

Github search api

Search issues

```
1 ghsearch.issues({
2   q: 'windows+state:open+repo:pksunkara/hub',
3   sort: 'created',
4   order: 'asc'
5 }, callback); //array of search results
```

Search repositories

```
1 ghsearch.repos({
2   q: 'hub+language:go',
3   sort: 'created',
4   order: 'asc'
5 }, callback); //array of search results
```

Search users

```
1 ghsearch.users({
2   q: 'tom+followers:>100',
3   sort: 'created',
4   order: 'asc'
5 }, callback); //array of search results
```

Search code

```
1 ghsearch.code({
2   q: 'auth+in:file+repo:pksunkara/hub',
3   sort: 'created',
4   order: 'asc'
5 }, callback); //array of search results
```

Get all Organizations / Users

This query supports pagination.

Note: For listing all Organizations / Users pagination is powered exclusively by the `since` parameter (Github APIs) . `since` denotes the `login ID` of last org / user you encountered. Also, note that it's important to pass `true` as third param when using pagination.

Organizations

```
1 var client = github.client();
2 client.get('/organizations', callback);
3 // OR
4 client.get('/organizations', since, per_page, true, callback);
```

Users

```
1 var client = github.client();
2 client.get('/users', callback);
3 // OR
4 client.get('/users', since, per_page, true, callback);
```

Testing

Before run test copy the `config.example.json` file in `test` folder to `config.json` and populate it with your github information.

Is suggested to fork <https://github.com/octocat/Spoon-Knife> repo and run test on your copy.

```
1 npm test
```

If you like this project, please watch this and follow me.

Contributors

Here is a list of Contributors

TODO

The following method names use underscore as an example. The library contains camel cased method names.

```

1
2 // public repos for unauthenticated, private and public for
  authenticated
3 me.get_watched_repositories(callback);
4 me.is_watching('repo', callback);
5 me.start_watching('repo', callback);
6 me.stop_watching('repo', callback);
7
8 // organization data
9 var org = octonode.Organization('bulletjs');
10
11 org.update(dict_with_update_properties, callback);
12 org.get_public_members(callback);
13 org.is_public_member('user', callback);
14 org.make_member_public('user', callback);
15 org.conceal_member('user', callback);
16
17 org.get_team('team', callback);
18 org.create_team({name: '', repo_names: '', permission: ''}, callback);
19 org.edit_team({name: '', permission: ''}, callback);
20 org.delete_team('name', callback);
21 org.get_team_members('team', callback);
22 org.get_team_member('team', 'user', callback);
23 org.remove_member_from_team('user', 'team', callback);
24 org.get_repositories(callback);
25 org.create_repository({name: ''}, callback);
26 org.get_team_repositories('team', callback);
27 org.get_team_repository('team', 'name', callback);
28 org.add_team_repository('team', 'name', callback);
29 org.remove_team_repository('team', 'name', callback);
30
31 var repo = octonode.Repository('pksunkara/octonode');
32
33 repo.update({name: ''}, callback);
34
35 // collaborator information
36 repo.add_collaborator('name', callback);
37 repo.remove_collaborator('name', callback);
38
39 // commit data
40 repo.get_commit('sha-id', callback);
41 repo.get_all_comments(callback);
42 repo.get_commit_comments('SHA ID', callback);
43 repo.comment_on_commit({body: '', commit_id: '', line: '', path: '',
  position: ''}, callback);
44 repo.get_single_comment('comment id', callback);
45 repo.edit_single_comment('comment id', callback);
46 repo.delete_single_comment('comment id', callback);
47
48 // downloads

```

```
49 repo.get_downloads(callback);
50 repo.get_download(callback);
51 repo.create_download({name: ''}, 'filepath', callback);
52 repo.delete_download(callback);
53
54 // keys
55 repo.get_deploy_keys(callback);
56 repo.get_deploy_key('id', callback);
57 repo.create_deploy_key({title: '', key: ''}, callback);
58 repo.edit_deploy_key({title: '', key: ''}, callback);
59 repo.delete_deploy_key('id', callback);
60
61 // watcher data
62 repo.get_watchers(callback);
63
64 // pull requests
65 repo.get_all_pull_request_comments(callback);
66 repo.get_pull_request_comment('id', callback);
67 repo.reply_to_pull_request_comment('id', 'body', callback);
68 repo.edit_pull_request_comment('id', 'body', callback);
69 repo.get_issues(params, callback);
70 repo.get_issue('id', callback);
71 repo.create_issue({title: ''}, callback);
72 repo.edit_issue({title: ''}, callback);
73 repo.get_issue_comments('issue', callback);
74 repo.get_issue_comment('id', callback);
75 repo.create_issue_comment('id', 'comment', callback);
76 repo.edit_issue_comment('id', 'comment', callback);
77 repo.delete_issue_comment('id', callback);
78 repo.get_issue_events('id', callback);
79 repo.get_events(callback);
80 repo.get_event('id', callback);
81 repo.get_labels(callback);
82 repo.get_label('id', callback);
83 repo.create_label('name', 'color', callback);
84 repo.edit_label('name', 'color', callback);
85 repo.delete_label('id', callback);
86 repo.get_labels_for_milestone_issues('milestone', callback);
87 repo.get_milestones(callback);
88 repo.get_milestone('id', callback);
89 repo.create_milestone('title', callback);
90 repo.edit_milestone('title', callback);
91 repo.delete_milestone('id', callback);
92
93 // raw git access
94 repo.create_blob('content', 'encoding', callback);
95 repo.get_commit('sha-id', callback);
96 repo.create_commit('message', 'tree', [parents], callback);
97 repo.get_reference('ref', callback);
98 repo.get_all_references(callback);
99 repo.update_reference('ref', 'sha', force, callback);
```

I accept pull requests

License

MIT/X11

octonode 644375ea24

FOSSA

✔ No Issues Found

LICENSE SCAN

DEEP IMPACT STATS

+ 7 Deep Dependencies

+ 4 Obligations from 3 Licenses

View More Details on FOSSA

Bug Reports

Report here.

Contact

Pavan Kumar Sunkara (pavan.sss1991@gmail.com)

Follow me on github, twitter