{cookiecutter.app_name} Documentation

Release 0.0.1

{cookiecutter.author_name}

Aug 22, 2018

Contents:

If you are new to `repomate`, the *Fundamentals* and *repomate User Guide* sections are must-reads. Developers looking to modify or utilize the core functionality in ways the CLI does not allow will be best served by looking at the modindex.

Please open an issue and tag it with the `docs` tag for any bugs or missing information.

CHAPTER 1

Fundamentals

repomate is an opinionated tool for managing large amounts of GitHub repositories for higher education courses. It was created as a result of the old teachers_pet tool not fulfilling our every desire, as well as GitHub's migration over to the browser based GitHub Classroom, which did not quite strike our fancy. repomate is essentially a newer (and hopefully better) version of teachers_pet, written in Python (which is commonly used at KTH) instead of Ruby (which is barely used at all).

1.1 Philosophy and Goals

The primary goal of repomate is to lower the entry level for incorporating git and GitHub into higher education coursework, hopefully opening up the wonderful world of version control to teachers who may not be subject experts (and to their students). As such, repomate is firmly seated in the convention over configuration camp, favoring highly opinionated workflows that are easy to get started with, rather than highly configurable ones. The target audience is primarily teachers seeking to incorporate git and GitHub into their courses, but lack the time or expertise to build their own automation system from scratch.

Note: Is there a difference between git and GitHub? Yes! git is the version control system, and GitHub is a company that hosts git servers on github.com, and provides enterprise software for hosting your own GitHub instances. The terms are often intermixed, but the distinction is important.

1.2 Terminology

Some terms occur frequently in repomate and are best defined up front. Some of the descriptions may not click entirely before reading the *repomate User Guide* section, so quickly browsing through these definitions and re-visiting them when needed is probably the best course of action.

- *Target organization*: The GitHub Organization related to the current course round.
- *Master repository*: Or *master repo*, is a template repository upon which student repositories are based.

- *Student repository*: Or *student repo*, refers to a *copy* of a master repo for some specific student.
- *GitHub instance*: A hosted GitHub service. This can be for example `https://github.com` or any Enterprise host.

1.3 Conventions

The following conventions are fundamental to working with `repomate`.

- For each course and course round, use one Organization.
- Any user of `repomate` has unrestricted access to the target organization (i.e. is an owner).
- Master repositories should be available as private repositories in the target organization (using local repos on the current machine is also *ok* and generally works well).
- Master repositories are added to the `master_repos` team.
- Student repositories are copies of the default branches of the master repositories (i.e. `--single-branch` cloning is used by default). That is, until students make modifications.
- Student repositories are named `<username>-<master_repo_name>` to guarantee unique repo names.
- Each student is assigned to a team with the same name as the student's username. It is the team that is granted access to the repositories, not the student's actual user.
- Student teams have `push` access to the repositories, but not administrative access (i.e. students can't delete their own repos).

Note: Few of these conventions are actually enforced, and there are ways around almost every single one. However, with the exception of the *one organization per course round* convention, which must be ensured manually, `repomate` will automatically adhere to the other conventions. Although `repomate` does adhere to the conventions, there is no way to stop users from breaking them using e.g. the GitHub web interface, manually performing master repo migrations etc. Straying form the conventions may cause `repomate` to behave unexpectedly.

CHAPTER 2

Install

2.1 Requirements

`repomate` requires Python 3.5+ and a somewhat up-to-date version of `git`. Officially supported platforms are `Ubuntu 17.04+` and `OSX`, but `repomate` should run fine on any Linux distribution and also on WSL on Windows 10. Please report any issues with operating systems and/or `git` versions on the issue tracker.

2.2 Option 1: Install from PyPi with *pip*

Important: Not yet available on PyPi, go with *clone repo* instead!

The latest release of `repomate` is on PyPi, and can thus be installed as usual with `pip`. I strongly discourage system-wide `pip` installs (i.e. `sudo pip install <package>`), as this may land you with incompatible packages in a very short amount of time. A per-user install can be done like this:

1. Execute `pip install --user repomate` to install the package.
2. Further steps to be added ...

2.3 Option 2: Clone the repo and the install with *pip*

If you want the dev version, you will need to clone the repo, as only release versions are uploaded to PyPi. Unless you are planning to work on this yourself, I suggest going with the release version.

1. **Clone the repo with `git`:**
    - `git clone https://github.com/slarse/repomate`
2. `cd` into the project root directory with `cd repomate`.

3. **Install the requirements with `pip install -r requirements.txt`**
    - To be able to run the tests, you must install the `requirements.test.txt` file.
4. **Install locally with `pip`.**
    - `pip install --user .`, this will create a local install for the current user.
    - Or just `pip install .` if you use `virtualenv`.
    - For development, use `pip install -e .` in a `virtualenv`.

CHAPTER 3

repomate User Guide

3.1 Getting Started (the verify-settings, migrate and setup commands)

Important: This guide assumes that the user has access to a bash shell, or is tech-savvy enough to translate the instructions into some other shell environment.

The basic workflow of repomate is best described by example. In this section, I will walk you through how to set up an Organization with master and student repositories by showing every single step I would perform myself. The basic workflow can be summarized in the following steps:

1. Create an organization (the target organization).
2. Configure repomate for the target organization.
3. Verify settings.
4. Migrate master repositories into the target organization.
5. Create one copy of each master repo for each student.

There is more to repomate, such as opening/closing issues, updating student repos and cloning repos in batches, but here we will just look at the bare minimum to get started. Now, let's delve into these steps in greater detail.

3.1.1 Create an Organization

This is an absolutely necessary pre-requisite for using repomate. Create an organization with an appropriate name on the GitHub instance you intend to use. You can find the New organization button by going to Settings -> Organization. I will call my *target organization* repomate_demo, so whenever you see that, substitute in the name of your target organization.

Important: At KTH, we most often do not want our students to be able to see each others' repos. By default, however, members have read access to *all* repos. To change this, go to the organization dashboard and find your way to `Settings -> Member privileges`. At the very bottom, there should be a section called `Default repository permission`. Set this to `None` to disallow students from viewing each others' repos unless explicitly given permission by an organization owner (e.g. you).

3.1.2 Configure `repomate` For the Target Organization

For the tool to work at all, an environment variable called `REPOMATE_OAUTH` must contain an OAUTH2 token to whichever GitHub instance you intend to use. See the GitHub OAUTH docs for how to create a token. The token should have the `repo` and `admin:org` scopes. Setting the token is easy in `bash`. Just add the following line to your `bash` config file (`~/.bashrc` on most Linux distros, and `~/.bash_profile` on OSX).

```
export REPOMATE_OAUTH=<SUPER SECRET TOKEN>
```

When that's added, either source the file with `source path/to/bash/config` or simply start another `bash` shell, which will automatically read the file. Verify that the token is there by typing:

```
$ echo $REPOMATE_OAUTH
```

You should see your token in the output.

Note: Whenever you see a `$` sign preceeding a line in a code block, you are meant to type what's *after* the `$` sign into your shell. Here, you should type only `echo $REPOMATE_OAUTH`, for example.

With that out of the way, let's create a configuration file We can now use `repomate` to figure out where it should be located.

```
$ repomate -h
[INFO] no config file found. Expected config file location: /home/USERNAME/.config/
↪repomate/config.cnf

<HELP MESSAGE OMITTED>
```

At the very top, you will find the expected config file location. The exact path will vary depending on operating system and username. Let's add a configuration file with the following contents:

```
[DEFAULTS]
github_base_url = https://some-enterprise-host/api/v3
user = slarse
org_name = repomate-demo
```

Now, you need to substitute in some of your own values in place of mine.

- **Enter the correct url for your GitHub instance. There are two options:**
  - If you are working with an enterprise instance, simply replace `some-enterprise-host` with the appropriate hostname.
  - If you are working with `github.com`, replace the whole url with `https://api.github.com`.
- Replace `slarse` with your GitHub username.
- Replace `repomate-demo` with whatever you named your target organization.

That's it for configuration, and we can check that the file is correctly found and parsed by running `repomate -h` again.

```
$ repomate -h
[INFO] config file defaults:

    github_base_url: https://some-enterprise-host/api/v3
    user: slarse
    org_name: repomate-demo

<HELP MESSAGE OMITTED>
```

The `[INFO] config file defaults:` message (along with the defaults) will pop up on every `repomate` command. I should note that the configuration file isn't strictly necessary, but it will save us the hassle of typing in the url, username and organization name on every single command to `repomate`.

3.1.3 Verify Settings

Now that everything is set up, it's time to verify all of the settings. Given that you have a configuration file that looks something like the one above, you can simply run the `verify-settings` command without any options.

```
$ repomate verify-settings
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] verifying settings ...
[INFO] trying to fetch user information ...
[INFO] SUCCESS: found user slarse, user exists and base url looks okay
[INFO] verifying oauth scopes ...
[INFO] SUCCESS: oauth scopes look okay
[INFO] trying to fetch organization ...
[INFO] SUCCESS: found organization test-tools
[INFO] verifying that user slarse is an owner of organization repomate-demo
[INFO] SUCCESS: user slarse is an owner of organization repomate-demo
[INFO] GREAT SUCCESS: All settings check out!
```

If any of the checks fail, you should be provided with a semi-helpful error message. When all checks pass and you get `GREAT SUCCESS`, move on to the next section!

3.1.4 Migrate Master Repositories Into the Target Organization

This step sounds complicated, but it's actually very easy, and can be performed with a single `repomate` command. There is however a pre-requisite that must be fulfilled. You must either

- Have local copies of your master repos.

or

- Have all master repos in the same GitHub instance as your target organization.

Assuming we have the repos `master-repo-1` and `master-repo-2` in the current working directory (i.e. local repos), all we have to do is this:

```
$ repomate migrate -mn master-repo-1 master-repo-2
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] created team master_repos
[INFO] cloning into file:///some/directory/path/master-repo-1
[INFO] cloning into file:///some/directory/path/master-repo-2
[INFO] created repomate-demo/master-repo-1
[INFO] created repomate-demo/master-repo-2
[INFO] pushing, attempt 1/3
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/master-repo-1 master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/master-repo-2 master
[INFO] done!
```

There are a few things to note here. First of all, the team `master_repos` is created. This only happens the first time `migrate` is run on a new organization. As the name suggests, this team houses all of the master repos. Each master repo that is migrated with the `migrate` command is added to this team, so they can easily be found at a later time. It may also be confusing that the local repos are being cloned (into a temporary directory). This is simply an implementation detail that does not need much thinking about. Finally, the local repos are pushed to the `master` branch of the remote repo. This command is perfectly safe to run several times, in case you think you missed something. Running the same thing again yields the following output:

```
$ repomate migrate -mn master-repo-1 master-repo-2
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] cloning into file:///some/directory/path/master-repo-1
[INFO] cloning into file:///some/directory/path/master-repo-2
[INFO] repomate-demo/master-repo-1 already exists
[INFO] repomate-demo/master-repo-2 already exists
[INFO] pushing, attempt 1/3
[INFO] https://some-enterprise-host/repomate-demo/master-repo-1 master is up-to-date
[INFO] https://some-enterprise-host/repomate-demo/master-repo-2 master is up-to-date
[INFO] done!
```

In fact, all `repomate` commands that deal with pushing to or cloning from repos in some way are safe to run over and over. This is mostly because of how `git` works, and has little to do with `repomate` itself. Now that our master repos are migrated, we can move on to setting up the student repos!

Note: The `migrate` command can also be used to migrate repos from somewhere on the GitHub instance into the target organization. To do this, use the `-mu` option and provide the urls, instead of `-mn` with local paths. For example, given a repo at `https://some-enterprise-host/other-org/master-repo-1`, it can be migrated into `repomate-demo` by typing

```
$ repomate migrate -mu https://some-enterprise-host/other-org/master-repo-1
```

3.1.5 Setup Student Repositories

Now that the master repos have been added to the target organization, it's time to create the student repos. While student usernames *can* be specified on the command line, it's often convenient to have them written down in a file instead. Let's pretend I have three students with usernames `spam`, `ham` and `eggs`. I'll simply create a file called `students.txt` and type each username on a separate line.

```
spam
ham
eggs
```

I want to create one student repo for each student per master repo. The repo names will be on the form `<username>-<master-repo-name>`, guaranteeing their uniqueness. Each student will also be added to a team (which bears the same name as the student's user), and it is the team that is allowed access to the student's repos, and not the student's actual user. That all sounded fairly complex, but again, it's as simple as issuing a single command with `repomate`.

```
$ repomate setup -mn master-repo-1 master-repo-2 -sf students.txt
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] cloning into master repos ...
[INFO] cloning into file:///home/slarse/tmp/master-repo-1
[INFO] cloning into file:///home/slarse/tmp/master-repo-2
[INFO] created team eggs
[INFO] created team ham
[INFO] created team spam
[INFO] adding members eggs to team eggs
[WARNING] user eggs does not exist
[INFO] adding members ham to team ham
[INFO] adding members spam to team spam
[INFO] creating student repos ...
[INFO] created repomate-demo/eggs-master-repo-1
[INFO] created repomate-demo/ham-master-repo-1
[INFO] created repomate-demo/spam-master-repo-1
[INFO] created repomate-demo/eggs-master-repo-2
[INFO] created repomate-demo/ham-master-repo-2
[INFO] created repomate-demo/spam-master-repo-2
[INFO] pushing files to student repos ...
[INFO] pushing, attempt 1/3
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/ham-master-repo-2␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/ham-master-repo-1␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/spam-master-repo-1␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/eggs-master-repo-2␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/eggs-master-repo-1␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/spam-master-repo-2␣
↪master
```

Note that there was a `[WARNING]` message for the username `eggs`: the user does not exist. At KTH, this is common, as many (sometimes most) first-time students will not have created their GitHub accounts until sometime after the

course starts. These students will still have their repos created, but the users need to be added to their teams at a later time (for example with the `repomate add-to-teams` command). This is one reason for why we use teams for access privileges: it's easy to set everything up even when the students have yet to create their accounts (given that their usernames are pre-determined).

And that's it, the organization is primed and the students should have access to their repositories!

3.2 Updating Student Repositories (the `update` command)

Sometimes, we find ourselves in situations where it is necessary to push updates to student repositories after they have been published. As long as students have not started working on their repos, this is fairly simple: just push the new files to all of the related student repos. However, if students have started working on their repos, then we have a problem. Let's start out with the easy case where no students have worked on their repos.

3.2.1 Scenario 1: Repos are Unchanged

Let's say that we've updated `master-repo-1`, and that users `spam`, `ham` and `eggs` should get the updates. Then, we simply run `update` like this:

```
$ repomate update -mn master-repo-1 -s spam eggs ham
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] cloning into master repos ...
[INFO] cloning into https://some-enterprise-host/repomate-demo/master-repo-1
[INFO] pushing files to student repos ...
[INFO] pushing, attempt 1/3
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/spam-master-repo-1␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/eggs-master-repo-1␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/ham-master-repo-1␣
↪master
[INFO] done!
```

That's all there is to it for this super simple case. But what if `ham` had started working on `ham-master-repo-1`?

Note: Here, `-s spam eggs ham` was used to directly specify student usernames on the command line, instead of pointing to a students file with `-sf students.txt`. All commands that require you to specify student usernames can be used with either the `-s|--students` or the `-sf|--students-file` options.

3.2.2 Scenario 2: At Least 1 Repo Altered

Let's assume now that `ham` has started working on the repo. Since we do not `force` pushes (that would be irresponsible!) to the student repos, the push to `ham-master-repo-1` will be rejected. This is good, we don't want to overwrite a student's progress because we messed up with the original repository. There are a number of things one *could* do in this situation, but in `repomate`, we opted for a very simple solution: open an issue in the student's repo that explains the situation.

Important: If we don't specify an issue to `repomate update`, rejected pushes will simply be ignored.

So, let's first create that issue. It should be a Markdown-formatted file, and the **first line in the file will be used as the title**. Here's an example file called `issue.md`.

```
This is a nice title

### Sorry, we messed up!
There are some grave issues with your repo, and since you've pushed to the
repo, you need to apply these patches yourself.

<EXPLAIN CHANGES>
```

Something like that. If the students have used `git` for a while, it may be enough to include the ouptut from `git diff`, but for less experienced students, plain text is more helpful. Now it's just a matter of using `repomate update` and including `issue.md` with the `-i|--issue` argument.

```
$ repomate update -mn master-repo-1 -s spam eggs ham -i issue.md
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] cloning into master repos ...
[INFO] cloning into https://some-enterprise-host/repomate-demo/master-repo-1
[INFO] pushing files to student repos ...
[INFO] pushing, attempt 1/3
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/spam-master-repo-1␣
↪master
[INFO] Pushed files to https://some-enterprise-host/repomate-demo/eggs-master-repo-1␣
↪master
[ERROR] Failed to push to https://some-enterprise-host/repomate-demo/ham-master-repo-1
return code: 128
fatal: repository 'https://some-enterprise-host/repomate-demo/ham-master-repo-1/' not␣
↪found
[WARNING] 1 pushes failed ...
[INFO] pushing, attempt 2/3
[ERROR] Failed to push to https://some-enterprise-host/repomate-demo/ham-master-repo-1
return code: 128
fatal: repository 'https://some-enterprise-host/repomate-demo/ham-master-repo-1/' not␣
↪found
[WARNING] 1 pushes failed ...
[INFO] pushing, attempt 3/3
[ERROR] Failed to push to https://some-enterprise-host/repomate-demo/ham-master-repo-1
return code: 128
fatal: repository 'https://some-enterprise-host/repomate-demo/ham-master-repo-1/' not␣
↪found
[WARNING] 1 pushes failed ...
[INFO] Opening issue in repos to which push failed
[INFO] Opened issue ham-master-repo-1/#1-'Nice title'
[INFO] done!
```

Note that `repomate` tries to push 3 times before finally giving up and opening an issue. This is because pushes can fail for other reasons than rejections, such as timeouts and other network errors.

Note: If you forget to specify the `-i|--issue` argument and get a rejection, you may simply rerun `update` and add it. All updated repos will simply be listed as `up-to-date`, and the rejecting repos will still reject the push! However, be careful not to run `update` with `-i` multiple times, as it will then open the same issue multiple times.

3.3 Opening and Closing issues (the `open-issue` and `close-issue` commands)

Sometimes, the best way to handle an error in a repo is to simply notify affected students about it. This is especially true if the due date for the assignment is rapidly approaching, and most students have already started modifying their repositories. Therefore, `repomate` provides the `open-issue` command, which can open issues in bulk. When the time is right (perhaps after the deadline has passed), issues can be closed with the `close-issue` command.

3.3.1 Opening Issues

The `open-issue` command is very simple. Before we use it, however, we need to write a Markdown-formatted issue. Just like with the `update` command, the **first line of the file is the title**. Here is `issue.md`:

```
An important announcement

### Dear students
I have this important announcement to make.

Regards,
_The Announcer_
```

Awesome, that's an excellent issue. Let's open it in the `master-repo-2` repo for our dear students `spam`, `eggs` and `ham`, who are listed in the `students.txt` file (see *Setup Student Repositories*).

```
$ repomate open-issue -mn master-repo-2 -sf students.txt -i issue.md
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] Opened issue spam-master-repo-2/#1-'An important announcement'
[INFO] Opened issue eggs-master-repo-2/#1-'An important announcement'
[INFO] Opened issue ham-master-repo-2/#1-'An important announcement'
```

From the output, we can read that in each of the repos, an issue with the title `An important announcement` was opened as issue nr 1 (`#1`). The number isn't that important, it's mostly good to note that the title was fetched correctly. And that's it! Neat, right?

3.3.2 Closing Issues

Now that the deadline has passed for `master-repo-2`, we want to close the issues opened in *open*. The `close-issue` command takes a *regex* that runs against titles. All issues with matching titles are closed. While you *can* make this really difficult, closing all issues with the title `An important announcement` is simple: we provide the regex `\AAn important announcement\Z`.

```
$ repomate close-issue -mn master-repo-2 -sf students.txt -r '\AAn important␣
↪announcement\Z'
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] closed issue spam-master-repo-2/#1-'An important announcement'
[INFO] closed issue eggs-master-repo-2/#1-'An important announcement'
[INFO] closed issue ham-master-repo-2/#1-'An important announcement'
```

And there we go, easy as pie!

Note: Enclosing a regex expression in `\A` and `\Z` means that it must match from the start of the string to the end of the string. So, the regex used here *will* match the title `An important announcement`, but it will *not* match e.g. `An important anouncement and lunch` or `Hey An important announcement`. In other words, it matches exactly the title `An important announcement`, and nothing else. Not even an extra space or linebreak is allowed.

3.4 Cloning Repos in Bulk (the `clone` command)

It can at times be beneficial to be able to clone a bunch of student repos at the same time. It could for example be prudent to do this slighly after a deadline, as timestamps in a `git` commit can easily be altered (and are therefore not particularly trustworthy). Whatever your reason may be, it's very simple using the `clone` command. Again, assume that we have the `students.txt` file from *Setup Student Repositories*, and that we want to clone all student repos based on `master-repo-1` and `master-repo-2`.

```
$ repomate clone -mn master-repo-1 master-repo-2 -sf students.txt
[INFO] config file defaults:

   github_base_url: https://some-enterprise-host/api/v3
   user: slarse
   org_name: repomate-demo

[INFO] cloning into student repos ...
[INFO] Cloned into https://some-enterprise-host/repomate-demo/spam-master-repo-1
[INFO] Cloned into https://some-enterprise-host/repomate-demo/ham-master-repo-1
[INFO] Cloned into https://some-enterprise-host/repomate-demo/ham-master-repo-2
[INFO] Cloned into https://some-enterprise-host/repomate-demo/eggs-master-repo-1
[INFO] Cloned into https://some-enterprise-host/repomate-demo/spam-master-repo-2
[INFO] Cloned into https://some-enterprise-host/repomate-demo/eggs-master-repo-2
```

Splendid! That's really all there is to it, the repos should now be in your current working directory.

CHAPTER 4

Configuration

`repomate` *must* be configured with a mandatory environment variable (see *oauth*). Additionally, some of the command line parameters can be pre-configured with e.g. the GitHub instances' API url and the target organization's name (see *config*).

4.1 REPOMATE_OAUTH Environment Variable

For the tool to work at all, an environment variable called `REPOMATE_OAUTH` must contain an OAUTH2 token to whichever GitHub instance you intend to use. See the GitHub OAUTH docs for how to create a token. The token should have the `repo` and `admin:org` scopes. Once you have the token, you should set the environment variable. In a `bash` terminal, this can be done with the command `export REPOMATE_OAUTH=<YOUR TOKEN>`, where `<YOUR TOKEN>` is replaced with the token.

4.2 Configuration File

An optional configuration file can be added, which specifies default values for the *–github_base_url*, *–org_name*, *–user* and *–students-list* command line options. This is especially useful for teachers who are managing repos for a single course (and, as a consequence, a single organization).

```
[DEFAULTS]
github_base_url = https://some-api-v3-url
user = YOUR_USERNAME
org_name = ORGANIZATION_NAME
students_file = STUDENTS_FILE_ABSOLUTE_PATH
```

To find out where to place the configuration file (and what to name it), run *repomate -h*. At the very top, there should be a line looking something like this:

```
[INFO] no config file found. Expected config file location: /home/USERNAME/.config/
↪repomate/config.cnf
```

The filepath at the end is where you should put your config file.

Important: Do note that the configuration file contains only default values. Specifying any of the parameters on the command line will override the configuration file's values.

Note: You can run `repomate verify-settings` to verify the basic configuration. This will check all settings but the students file.

CHAPTER 5

`repomate` Module Reference

CHAPTER 6

Indices and tables

- genindex
- modindex
- search