# Forge User and Developer Manual

for version 0.1.0

**Jonas Bernoulli**

# Table of Contents

# 1 Introduction

Forge allows you to work with Git forges, such as Github and Gitlab, from the comfort of Magit and the rest of Emacs.

Forge fetches issues, pull-requests and other data using the forge's API and stores that in a local database. Additionally it fetches the pull-request references using Git. Forge implements various features that use this data but the database and pull-request refs can also be used by third-party packages.

# 2 Supported Forges and Hosts

Currently Forge supports two forges and three more forges partially. Additionally it supports four semi-forges. Support for more forges and semi-forges can and will be added.

Both forges and semi-forges provide web interfaces for Git repositories. Forges additionally support pull-requests and issues and make those and other information available using an API.

When a forge is only partially supported, then that means that only the functionality that does not require the API is implemented, or in other words, that the forge is only supported as a semi-forge.

A host is a particular instance of a forge. For example the hosts `https://gitlab.com` and `https://salsa.debian.org` are both instances of the Gitlab forge. Forge supports some well known hosts out of the box and additional hosts can easily be supported by adding entries to the option `forge-alist`.

For more details about the caveats mentioned below (and some others) also see Chapter 3 [Getting Started], page 5.

## 2.1 Supported Forges

- Github

  Forge's support for Github can be considered the "reference implementation". Support for other forges can lag behind a bit.

  Github Caveats

  - Forge uses the Github GraphQl API when possible but has to fall back to use the REST API in many cases because the former is still rather incomplete.
  - Forge depends on the `updated_at` field being updated when appropriate. For Github pull-requests at least, that is not always done.

  Github Hosts

  - `https://github.com`

- Gitlab

  Gitlab Caveats

  - Forge cannot use Ghub's setup wizard to create the API token because the Gitlab API does not support doing that.
  - Forge cannot provide notifications because the Gitlab API does not expose those.

  Gitlab Hosts

  - `https://gitlab.com`
  - `https://salsa.debian.org`

## 2.2 Partially Supported Forges

- Gitea `https://gitea.io/en-us`

  This is the next forge whose API will to be supported.

Gitea Hosts

- `https://teahub.io` Note that this host is in pre-alpha and that it is not clear whether the operators are trustworthy. I have only added this to `forge-alist` because I couldn't find any other more trustworthy **and** persistent public instances but needed something for testing purposes. The Gitea maintainers plan to create their own instance, see `https://github.com/go-gitea/gitea/issues/1029`. Once that is available, I will probably remove teahub.io again.

- Gogs `https://gogs.io`

  Once Gitea is supported it should be fairly simple to support Gogs too, because the former is a fork of the latter and the APIs seem to still be very similar.

  Gogs Hosts

  - `https://code.orgmode.org`

- Bitbucket

  I don't plan to support Bitbucket's API any time soon, and it gets less likely that I will every do it every time I look at it.

  Bitbucket Caveats

  - The API documentation is poor and initial tests indicated that the implementation is buggy.
  - Atlassian's offering contains two very distinct implementations that are both called "Bitbucket". Forge only supports the implementation whose only instance is available at `https://bitbucket.org`, because I only have access to that.
  - Unlike all other forges, Bitbucket does not expose pull-requests as references in the upstream repository. For that reason Forge actually treats it as a semi-forge, not as forge whose API is not supported yet. This means that you cannot checkout pull-requests locally. There is little hope that this will ever get fixed; the respective issue was opened six years ago and there has been no progress since: `https://bitbucket.org/site/master/issues/5814`.

  Bitbucket Hosts

  - `https://bitbucket.org`

## 2.3 Supported Semi-Forges

- Gitweb `https://git-scm.com/docs/gitweb`

  Gitweb Caveats

  - I could find only one public installation (`https://git.savannah.gnu.org`), which gives users the choice between Gitweb and Cgit. The latter seems more popular (not just on this site).

- Cgit `https://git.zx2c4.com/cgit/about`

  Cgit Caveats

  - Different sites use different URL schemata and some of the bigger sites use a fork. For this reason Forge has to provide several classes to support different variations of Cgit and you have to look at their definitions to figure out which on is the correct one for a particular installation.

Cgit Hosts

- `https://git.savannah.gnu.org/cgit`
- `https://git.kernel.org`
- `https://repo.or.cz`

- Stgit `https://codemadness.org/git/stagit/file/README.html`

  Stgit Caveats

  - Stgit cannot show logs for branches beside "master". For that reason Forge takes users to a page listing the branches when they request a log a particular branch (even for "master" whose log is just one click away from there).

  Stgit Hosts

  - `https://git.suckless.org`

- Stht `https://meta.sr.ht`

  Stht Caveats

  - Stht cannot show logs for branches beside "master". For that reason Forge takes users to a page listing the branches when they request a log a particular branch (even for "master" whose log is just one click away from there).

  Stht Hosts

  - `https://git.sr.ht`

# 3 Getting Started

Getting started using Forge should be fairly easy, but there are a few caveats you should be aware of:

- Forge uses the Ghub package to access forge APIs. That package comes with a setup wizard that should make it easy to create and store a Github API token. Unfortunately the same cannot be done for other forges and in the past it has failed for some users for Github too, in particular when using two-factor authentication. See Section 3.2 [Token Creation], page 6, for more information.
- Fetched information is stored in a database. The table schemata of that database have not been finalized yet. Until that has happened it will occasionally have to be discard. That isn't such a huge deal because for now the database does not contain any information that cannot simply be fetched again, see Section 3.1 [Initial Pull], page 5.
- Fetching is implemented under the assumption that the API can be asked to list the things that have changed since we last checked. Unfortunately the APIs are not bug-free, so this is not always the case. Issues such as `https://platform.github.community/t/7284` can take years to get addresses (in closed-source software), so I am no longer delaying the initial Forge release because of that. If in doubt, then re-fetch an individual pull-request to ensure it is up-to-date using the command `forge-pull-pullreq`.
- Some other, forge-specific, caveats are mentioned in Chapter 2 [Supported Forges and Hosts], page 2.

## 3.1 Initial Pull

To start using Forge in a certain repository visit the Magit status buffer for that repository and type `F y` (`forge-pull`).

The first time you do that for any repository from `https://github.com` you will be guided through the process of creating the API token. For other forges as well as for other Github instances some additional setup is required **before** you do so. See Section 3.2 [Token Creation], page 6.

The first time `forge-pull` is run in a repository, an entry for that repository is added to the database and a new value is added to the Git variable `remote.<remote>.fetch`, which fetches all pull-requests. (`+refs/pull/*/head:refs/pullreqs/*` for Github)

`forge-pull` then fetches topics and other information using the forge's API and pull-request references using Git.

The initial fetch can take a while but most of that is done asynchronously. Storing the information in the database is done synchronously though, so there can be a noticeable hang at the end. Subsequent fetches are much faster.

Fetching issues from Github is much faster than fetching from other forges because making a handful of GraphQl requests is much faster than making hundreds of REST requests.

## 3.2 Token Creation

Forge uses the Ghub package to access the APIs of supported Git forges. Ghub comes with a setup wizard that guides the user through the process of creating an API token for Github.com. When accessing a Github Enterprise instance, then some manual setup is required before the wizard can be used. Other forges don't support creating tokens using the API at all. Before accessing such a forge you have to create a token using the respective web interface.

Please consult Ghub's manual to learn more about token creation. See Section "Getting Started" in `ghub` in particular.

Ghub does **not** associate a given local repository with a repository on a forge. The Forge package itself takes care of this. In doing so it ignores the Git variable `ghub.host` and other `FORGE.host` variables used by Ghub. (But `github.user` and other variables used to specify the user are honored). Forge associates the local repository with a forge repository by first determining which remote is associated with the upstream repository and then looking that up in `forge-alist`.

If only one remote exists, then Forge uses that unconditionally. If several remotes exist, then a remote may be selected based on its name.

The convention is to name the upstream remote `origin`. If you follow this convention, then you have to do nothing else and the remote by that name is automatically used, provided it exists and regardless of whether other remotes exist. If it does not exist, then no other remotes are tried.

If you do not follow the naming convention, then you have to inform Forge about that by setting the Git variable `forge.remote` to the name that you instead use for upstream remotes. If this variable is set, then Forge uses the remote by that name, if it exists, the same way it may have used `origin` if the the variable were undefined. I.e. it does not fall through to try `origin` if no remote by your chosen name exists.

Once the upstream remote has been determined, Forge looks it up in `forge-alist`, using the host part of the url as the key. For example the key for `git@github.com:magit/forge.git` is `github.com`.

# 4 Usage

Once information has been fetched from a repository's forge, Forge adds two additional sections named "Pull requests" and "Issues" to Magit's status buffer. Some of Forge's commands are only bound when point is within one of these sections but other commands are also available elsewhere in Magit's status buffer and/or from Magit's popup commands.

`'` (`forge-dispatch-popup`)
: This prefix command is available in any Magit buffer and provides access to several of the available Forge commands. Most of these commands are also bound elsewhere, but some or not. See the following sections for information about the available commands.

## 4.1 Pulling

The commands that pull forge data are available from the same popup (`magit-pull-popup` on `F`) that is used to pull Git data.

*F y* (`forge-pull`)
: This command uses a forge's API to fetch topics and other information about the current repository and stores the fetched information in the database. It also fetches notifications for all repositories from the same forge host. (Currently this is limitted to Github.) Finally it fetches pull-request references using Git.

  After using this command for the first time in a given repository the status buffer for that repository always lists the pull-requests and issues. See Section 3.1 [Initial Pull], page 5.

*F Y* (`forge-pull-notifications`)
: This command uses a forge's API to fetch all notifications from that forge, including but not limited to the notifications for the current repository.

  Fetching all notifications fetches associated topics even if you have not started fetching **all** topics for the respective repositories (using `forge-pull`), but it does not cause the topics to be listed in the status buffer of such "uninitialized" repositories.

Note how pulling data from a forge's API works the same way as pulling Git data does; you do it explicitly when you want to see the work done by others.

This is less disruptive, more reliable and easier to understand than if Forge did the pulling by itself at random intervals. It might however mean that you occasionally invoke a command expecting the most recent data to be available and then having to abort to pull first. The same can happen with Git, e.g. you might attempt to merge a branch that you know exists but haven't actually pulled yet.

*M-x forge-pull-pullreq* (`forge-pull-pullreq`)
: This command uses a forge's API to fetch a single pull-request and stores it in the database.

  Normally you wouldn't want to pull a single pull-request by itself, but due to a bug in the Github API you might sometimes have to do so.

Fetching is implemented under the assumption that the API can be asked to list the things that have changed since we last checked. Unfortunately the APIs are not bug-free, so this is not always the case. Issues such as `https://platform.github.community/t/7284` can take years to get addresses (in closed-source software), so I am no longer delaying the initial Forge release because of that. If in doubt, then re-fetch an individual pull-request to ensure it is up-to-date using this command.

## 4.2 Branching

Forge provides commands for creating and checking out a new branch or worktree from a pull-request. These commands are available from the same popups as the commands used to create and check out branches and worktrees in a more generic fashion (`magit-branch-popup` on `b` and `magit-worktree-popup` on `%`).

*`b Y`* (`forge-branch-pullreq`)

This command creates and configures a new branch from a pull-request, creating and configuring a new remote if necessary.

The name of the local branch is the same as the name of the remote branch that you are being asked to merge, unless the contributor could not be bother to properly name the branch before opening the pull-request. The most likely such case is when you are being asked to merge something like "fork/master" into "origin/master". In such cases the local branch will be named "pr-N", where `N` is the pull-request number.

These variables are always set by this command:

- `branch.<name>.pullRequest` is set to the pull-request number.
- `branch.<name>.pullRequestRemote` is set to the remote on which the pull-request branch is located.
- `branch.<name>.pushRemote` is set to the same remote as `branch.<name>.pullRequestRemote` if that is possible, otherwise it is set to the upstream remote.
- `branch.<name>.description` is set to the pull-request title.
- `branch.<name>.rebase` is set to `true` because there should be no merge commits among the commits in a pull-request.

This command also configures the upstream and the push-remote of the local branch that it creates.

The branch against which the pull-request was opened, is always used as the upstream. This makes it easy to see what commits you are being asked to merge in the section titled something like "Unmerged into origin/master".

Like for other commands that create a branch it depends on the option `magit-branch-prefer-remote-upstream` whether the remote branch itself or the respective local branch is used as the upstream, so this section may also be titled e.g. "Unmerged into master".

When necessary and possible, then the remote pull-request branch is configured to be used as the push-target. This makes it easy to see what further changes

the contributor has made since you last reviewed their changes in the section titled something like "Unpulled from origin/new-feature" or "Unpulled from fork/new-feature".

- If the pull-request branch is located in the upstream repository, then you probably have set `remote.pushDefault` to that repository. However some users like to set that variable to their personal fork, even if they have push access to the upstream, so `branch.<name>.pushRemote` is set anyway.
- If the pull-request branch is located inside a fork, then you are usually able to push to that branch, because Github by default allows the recipient of a pull-request to push to the remote pull-request branch even if it is located in a fork. The contributor has to explicitly disable this.
  - If you are not allowed to push to the pull-request branch on the fork, then a branch by the same name located in the upstream repository is configured as the push-target.
  - A—sadly rather common—special case is when the contributor didn't bother to use a dedicated branch for the pull-request.

    The most likely such case is when you are being asked to merge something like "fork/master" into "origin/master". The special push permission mentioned above is never granted for the branch that is the repository's default branch, and that would almost certainly be the case in this scenario.

    To enable you to easily push somewhere anyway, the local branch is named "pr-N" (where `N` is the pull-request number) and the upstream repository is used as the push-remote.
  - Finally, if you are allowed to push to the pull-request branch and the contributor had the foresight to use a dedicated branch, then the fork is configured as the push-remote.

  The push-remote is configured using `branch.<name>.pushRemote`, even if the used value is identical to that of `remote.pushDefault`, just in case you change the value of the latter later on. Additionally the variable `branch.<name>.pullRequestRemote` is set to the remote on which the pull-request branch is located.

*b y* (`forge-checkout-pullreq`)
: This command creates and configures a new branch from a pull-request the same way `forge-branch-pullreq` does. Additionally it checks out the new branch.

*% y* (`forge-checkout-worktree`)
: This command creates and configures a new branch from a pull-request the same way `forge-branch-pullreq` does. Additionally it checks out the new branch using a new working tree.

When you delete a pull-request branch, which was created using one of the above three commands, then `magit-branch-delete` usually offers to also delete the corresponding remote. It does not offer to delete a remote if (1) the remote is the upstream remote, and/or (2) if other branches are being fetched from the remote.

Note that you have to delete the local branch (e.g. "feature") for this to work. If you delete the tracking branch (e.g "fork/feature"), then the remote is never removed.

## 4.3 Working with Topics

We call both issues and pull-requests "topics". The contributions to the conversation are called "posts".

### 4.3.1 Visiting Topics

Magit's status buffer contains lists of issues and pull-requests. Topics are ordered by last modification time. All open issues and some recently edited and closed topics are listed.

Forge provides some commands that act on the listed topics. These commands can also be used in other contexts, such as when point is on a commit or branch section.

*C-c C-w* (forge-browse-TYPE)
*C-c C-w* (forge-browse-dwim)
These commands visit the pull-request(s), issue(s), post, branch, commit or remote at point in a browser.

This is implemented using various commands named forge-browse-TYPE, and the key binding is defined by remapping magit-browse-thing (as defined in magit-mode-map). For commit sections this key is bound to forge-browse-dwim, which prefers a topic over a branch and a branch over a commit.

*RET* (forge-visit-TYPE)
*C-c C-v* (forge-visit-topic)
These commands visit the pull-request or issue at point in a separate buffer.

The RET binding is only available when point is on a issue or pull-request section because that key is already bound to something else for most of Magit's own sections. C-c C-v however is also available on regular commit (e.g. in a log) and branch sections.

This is implemented using various commands named forge-visit-TYPE and the key binding is defined by remapping magit-visit-thing (as defined in magit-mode-map).

### 4.3.2 Listing Topics and Notifications

Magit's status buffer contains lists of issues and pull-requests, but you can also list topics as well as notifications in separate buffers.

*' l* (forge-list-notifications)
This command lists all notifications for all forges in a separate buffer.

*' P* (forge-list-pull-requests)
This command lists the current repository's pull-requests in a separate buffer.

*' I* (forge-list-issues)
This command lists the current repository's issues in a separate buffer.

### 4.3.3 Creating Topics

*' p* (forge-create-pullreq)
*C-n C-n [on "Pull requests" section]* (forge-create-pullreq)
This command creates a new pull-request for the current repository.

*' i* (forge-create-issue)
*C-n C-n [on "Issues" section]* (forge-create-pullreq)
This command creates a new issue for the current repository.

### 4.3.4 Editing Topics and Posts

We call both issues and pull-requests "topics". The contributions to the conversation are called "posts". The post that initiated the conversation is also called a post.

These commands are available only from the topic buffer (i.e. from the buffer that shows the posts on a topic). Other commands that also work in other buffers are available here also. For example C-c C-w on a post visits that post in a browser.

*C-c C-n* (forge-create-post)
*C-c C-r* (forge-create-post)
This command allows users to create a new post on an existing topic. It opens a buffer in which the user can write the post. When the post is done, then the user has to sumbit using C-c C-c.

*C-c C-e [on a post section]* (forge-edit-post)
This command visits an existing post in a separate buffer. When the changes to the post are done, then the user has to sumbit using C-c C-c.

*C-c C-e [on "Title" section]* (forge-edit-topic-title)
This command reads a new title for an existing topic in the minibuffer.

*C-c C-e [on "State" section]* (forge-edit-topic-state)
This command toggles the state of an existing topic between "open" and "closed".

*C-c C-e [on "Labels" section]* (forge-edit-topic-labels)
This command reads a list of labels for an existing topic in the minibuffer.

*C-c C-e [on "Assignees" section]* (forge-edit-topic-assignees)
This command reads a list of assignees for an existing topic in the minibuffer.

Creating a new post and editing an existing post are similar to creating a new commit and editing the message of an existing commit. In both cases the message has to be written in a separate buffer. And then the process has to be finished or canceled using a separate command.

The following commands are available in buffers used to edit posts:

*C-c C-c* (forge-post-submit)
This command submits the post that is being edited in the current buffer.

*C-c C-k* (forge-post-cancel)
This command cancels the post that is being edited in the current buffer.

## 4.4 Miscellanous

*M-x forge-reset-database* (`forge-reset-database`)

This command moves the current database file to the trash and creates a new empty database.

This is useful after the database's table schemata have changed, which will happen a few times while the Forge functionality is still under heavy development.