Forge User and Developer Manual

for version 0.3.2.50-git

Jonas Bernoulli

Table of Contents

1 Introduction 1

2 Supported Forges and Hosts 2

2.1 Supported Forges 2
2.2 Partially Supported Forges 3
2.3 Supported Semi-Forges 3

3 Getting Started 5

3.1 Loading Forge 5
3.2 Token Creation 5
3.3 Initial Pull 6
3.4 Repository Detection 6
3.5 Caveats 8

4 Usage 9

4.1 Pulling 9
4.2 Branching 10
4.3 Working with Topics 12
4.3.1 Visiting Topics 12
4.3.2 Listing Topics and Notifications 13
4.3.3 Creating Topics 15
4.3.4 Editing Topics and Posts 15
4.4 Miscellaneous 17

Appendix A FAQ 19

A.1 Is it possible to create a single pull-request without pulling everything? 19
A.2 `error in process filter: HTTP Error: 502, "Bad gateway"` 19

Appendix B Keystroke Index 20

Appendix C Function and Command Index 21

Appendix D Variable Index 22

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` (see Section 3.4 [Repository Detection], page 6).

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.
- The Github GraphQL API has a hard-coded timeout on queries. The only solution is to reduce the number of entities we query at once, which can be done by adjusting either the `forge.graphqlItemLimit` git variable or the field "GQL entity limit" in a status buffer.
- 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 provide notifications because the Gitlab API does not expose those.

Gitlab Hosts

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

2.2 Partially Supported Forges

Gitea https://gitea.io

This is the next forge whose API will be supported.

Gitea Hosts

- `https://codeberg.org`

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 https://bitbucket.org

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 one 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 the log for a particular branch (even for "master" whose log is just one click away from there).

Stgit Hosts

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

Srht https://meta.sr.ht

Srht Caveats

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

Srht Hosts

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

3 Getting Started

Initial setup turned out to more challenging for many users than I had hoped. I am trying to counter that by providing detailed instructions that cover not just the happy path but also many of the issues users have ran into in the past. The recommended reading is longer than one might wish, but skipping it does not necessarily save time.

Forge uses the Ghub package to access the APIs of supported forges. How this works and how to create and store a token is documented in Section "Getting Started" in `ghub`. Please read that in full before coming back here and continuing with the subsections listed below.

`https://github.com/magit/forge/wiki` lists additional resources; including a link to a much shorter quick-start guide provided by a user.

3.1 Loading Forge

Loading Magit doesn't cause Forge to be loaded automatically. Adding something like this to your init file takes care of that:

```
(with-eval-after-load 'magit
  (require 'forge))
```

Or if you use `use-package`:

```
(use-package forge
  :after magit)
```

3.2 Token Creation

Forge uses the Ghub package to access the APIs of supported Git forges. How this works and how to create and store a token is documented in Section "Getting Started" in `ghub`.

Ghub used to provide a setup wizard, but that had to be removed for reasons given in the manual just mentioned. Nowadays there is no way around reading the documentation and doing this manually I am afraid.

Forge requires the following token scopes.

- For Github these scopes are required.
  - `repo` grants full read/write access to private and public repositories.
  - `user` grants access to profile information.
  - `read:org` grants read-only access to organization membership.

  More information about these and other scopes can be found at `https://docs.github.com/en/developers/apps/scopes-for-oauth-apps`.
- For Gitlab instances `api` is the only required scope. It gives read and write access to everything. The Gitlab API provides more fine-grained scopes for read-only access, but when any write access at all is required, then it is all or nothing.

3.3 Initial Pull

To start using Forge in a certain repository visit the Magit status buffer for that repository and type `f n` (`forge-pull`). Alternatively, you can use `M-x forge-add-repository`, which makes it possible to add a forge repository without pulling all topics and even without having to clone the respective Git repository.

You must set up a token **before** you can add the first repository. See Section 3.2 [Token Creation], page 5.

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.

3.4 Repository Detection

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 `*.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. To reduce the number of support requests, this is even the case if the Git variable `forge.remote` names another, non-existent, remote.

If several remotes exist, then a remote may be selected based on its name. Almost always we want to fetch the data associated with the upstream repository, so that is what the logic described here tries to achieve. The convention is to name the upstream remote `"origin"`, and if that convention were universally followed, then things would be trivial. However many people name the upstream remote `"upstream"`, which also makes sense.

> Note, however, that even though a surprising number of people do just that, it does not make any sense to use the name `"origin"` to refer to a fork; not even to your own fork. A fork is a **copy** of the original, `"copy"` is an antonym for `"original"`, and the word `"origin"` is not only closely related to but even contained in the word `"original"`. Naming a fork the `"origin"` is at best extremely confusing.

copy	a thing made to be similar or identical to another.
original	the earliest form of something, from which copies may be made.
origin	the point or place where something begins, arises, or is derived.

If several remotes exist, then the following remote names are tried in order and the first remote thus named that exists in the repository is used.

1. The value of the Git variable `forge.remote`, if set. If the variable has a value but no remote by the specified name exists, then a warning is shown, but otherwise this conflict is ignored. This behavior is arguably odd, but due to historic and pragmatic reasons it is the least painful path forward.
2. The remote named `upstream`, if it exists.
3. The remote named `origin`, if it exists.

The remote named "upstream" is preferred over the remote named "origin" because the existence of the former strongly suggests that the latter is either not used in this repository (in which case the order does not matter) or else it is abused as the name of a fork (in which case "upstream" must be preferred).

`forge.remote` [Variable]

The value of this variable specifies the remote from which Forge fetches data. It is usually best to leave this unspecified and to rely on the behavior described above.

If the remote has to be specified explicitly, then this should be done locally, for a single repository.

Only ever set this globally, if you consistently use a certain name to refer to the upstream repository and it isn't one of "upstream" or "origin", and you **never** use that name to refer to a repository that does **not** refer to the upstream repository.

N r (`forge-forge.remote`)

This command changes the value of the `forge.remote` Git variable in the current repository.

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`.

`forge-alist` [User Option]

This option defines forge hosts known to Forge.

Each entry has the form `(GITHOST APIHOST ID CLASS)`.

- GITHOST is matched against the host part of Git remote urls using `forge--url-regexp` to identify the forge.
- APIHOST is the api endpoint of the forge's api.
- ID is used to identify the forge in the local database.
- CLASS is the class to be used for repository from the forge.

GITHOST and APIHOST can be changed, but ID and CLASS are final. If you change ID, then the identity of every repository from that forge changes. If you change CLASS, then things start falling apart.

There can be multiple elements that only differ in GITHOST. Among those, the canonical element should come first. Any elements that have the same APIHOST must also have the same ID, and vice-versa.

When adding a new entry, look at the default entries.

Complications:

- When connecting to a Github Enterprise edition whose REST API's end point is "<host>/v3" and whose GraphQL API's end point is "<host>/graphql", then use "<host>/v3" as APIHOST. This is a historic accident. See issue #174.
- This variable does not account for the possibility that the git repository does not use the same address as the web interface. That is another historic accident and if many users are affected by it, we might have to fix it properly. If you would like that, comment on issue #532.

  For now a workaround has to be used. Assuming the web interface is available at "example.com" and the repository is at "ssh.example.com", use "example.com" as GITHOST and add something like this to ".gitconfig":

  ```
  [url "git@ssh.example.com"]
      insteadOf = git@example.com"
  ```

3.5 Caveats

- 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 discarded. 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 3.3 [Initial Pull], page 6.
- 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. If in doubt, then re-fetch an individual topic to ensure it is up-to-date using the command `forge-pull-topic`.
- Some other, forge-specific, caveats are mentioned in Chapter 2 [Supported Forges and Hosts], page 2.

4 Usage

Once information has been pulled from a repository's forge for the first time, Forge adds two additional sections, named "Pull requests" and "Issues" to Magit's status buffer. It is also possible to add a repository to the local database without pulling all the data, which is useful if you want to create a single pull-request.

N a (`forge-add-repository`)
: This command adds a repository to the database.

  It offers to either pull topics (now and in the future) or to only pull individual topics when the user invokes `forge-pull-topic`.

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 transient commands.

N (`forge-dispatch`)
: 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 are not. See the following sections for information about the available commands.

  Throughout this manual you will find many bindings that begin with `N`, but if you prefer to continue to use `forge-dispatch`'s older binding you can substitute `'` for that.

4.1 Pulling

The commands that fetch forge data are available from the same transient prefix command (`magit-fetch` on `f`) that is used to fetch Git data. If option `magit-pull-or-fetch` is non-nil, then they are also available from the `magit-pull` transient (on `F`).

f n (`forge-pull`)
N f f
: 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 limited 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 3.3 [Initial Pull], page 6.

f N (`forge-pull-notifications`)
N f n
: 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 have 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.

N f t (`forge-pull-topic`)

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. If in doubt, then re-fetch an individual topic to ensure it is up-to-date using the command `forge-pull-topic`.

4.2 Branching

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

b F (`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 bothered 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, 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 f (`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.

Z n (`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.

`forge-checkout-worktree-read-directory-function` [User Option]
: This function is used by `forge-checkout-worktree` to read read the new worktree directory where it checks out to pull-request. It takes the pull-request as the only argument and must return a directory.

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`)
N b r (`forge-browse-remote`)
N b I (`forge-browse-issues`)
N b P (`forge-browse-pullreqs`)
N b t (`forge-browse-topic`)
N b i (`forge-browse-issue`)
N b p (`forge-browse-pullreq`)
: These commands visit the topic, issue(s), pull-request(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`)
N v t (`forge-visit-topic`)
N v i (`forge-visit-issue`)
N v p (`forge-visit-pullreq`)

These commands visit the pull-request(s), issue(s), or repository 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

By default Forge lists a subset of topics directly in the Magit status buffer. It also provides commands to list topics and notifications in separate buffers.

Forge adds the following functions to `magit-status-sections-hook`:

`forge-insert-pullreqs` [Function]
This function inserts a list of the most recent and/or open pull-requests.

`forge-insert-issues` [Function]
This function inserts a list of the most recent and/or open issues.

`forge-topic-list-limit` [User Option]
This option limits the number of topics listed by the above functions.

All unread topics are always shown. If the value of this option has the form `(OPEN . CLOSED)`, then the integer `OPEN` specifies the maximal number of topics and `CLOSED` specifies the maximal number of closed topics. IF `CLOSED` is negative then show no closed topics until the command `forge-toggle-closed-visibility` changes the sign.

`forge-toggle-closed-visibility` [Command]
This command toggles whether the above two functions list recently closed issues in the current buffer.

The following functions are also suitable for `magit-status-sections-hook`:

`forge-insert-assigned-pullreqs` [Function]
This function inserts a list of open pull-requests that are assigned to you.

`forge-insert-requested-reviews` [Function]
This function inserts a list of open pull-requests that are awaiting your review.

`forge-insert-authored-pullreqs` [Function]
This function inserts a list of open pull-requests that are authored by you.

`forge-insert-assigned-issues` [Function]
This function inserts a list of open issues that are assigned to you.

`forge-insert-authored-issues` [Function]
This function inserts a list of open issues that are authored by you.

The following commands list repositories, notifications and topics in dedicated buffers:

N l r (`forge-list-repositories`)
This command lists all known repositories in a separate buffer.

N l n (`forge-list-notifications`)
This command lists all notifications for all forges in a separate buffer.

N l p (`forge-list-pullreqs`)
This command lists the current repository's pull-requests in a separate buffer.

N l i (`forge-list-issues`)
This command lists the current repository's issues in a separate buffer.

`forge-list-labeled-pullreqs` [Command]
This command lists the current repository's open pull-requests that are labeled with a label read from the user.

`forge-list-labeled-issues` [Command]
This command lists the current repository's open issues that are labeled with a label read from the user.

`forge-list-assigned-pullreqs` [Command]
This command lists the current repository's open pull-requests that are assigned to you in a separate buffer.

`forge-list-assigned-issues` [Command]
This command lists the current repository's open issues that are assigned to you in a separate buffer.

`forge-list-requested-reviews` [Command]
This command lists pull-requests of the current repository that are awaiting your review in a separate buffer.

`forge-list-authored-pullreqs` [Command]
This command lists the current repository's open pull-requests that are authored by you in a separate buffer.

`forge-list-authored-issues` [Command]
This command lists the current repository's open issues that are authored by you in a separate buffer.

`forge-list-owned-pullreqs` [Command]
This command lists open pull-requests from all the repositories that you own. Options `forge-owned-accounts` and `forge-owned-ignored` controls which repositories are considered to be owned by you. Only Github is supported for now.

`forge-list-owned-issues` [Command]
: This command lists open issues from all the repositories that you own. Options `forge-owned-accounts` and `forge-owned-ignored` controls which repositories are considered to be owned by you. Only Github is supported for now.

`forge-owned-accounts` [User Option]
: This is an alist of accounts that are owned by you. This should include your username as well as any organization that you own. Used by the commands `forge-list-owned-issues`, `forge-list-owned-pullreqs` and `forge-fork`.

  Each element has the form `(ACCOUNT . PLIST)`. The following properties are currently being used:

  - `remote-name` The default name suggested by `forge-fork` for a fork created within this account. If unspecified, then the name of the account is used.

  Example: `(("tarsius") ("emacsmirror" remote-name "mirror"))`.

`forge-owned-ignored` [User Option]
: This is a list of repository names that are considered to not be owned by you even though they would have been considered to be owned by you based on `forge-owned-accounts`.

4.3.3 Creating Topics

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

N c i (`forge-create-issue`)
C-c C-n [on "Issues" section]
: 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
: 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 submit using `C-c C-c`.

  If the region is active and marks part of an existing post, then that part of the post is quoted. Otherwise, or if a prefix argument is used, then the complete post that point is currently on is quoted.

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 submit 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 "Draft" section] (forge-edit-topic-draft)
: This command toggles whether an existing topic is a draft or not.

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 "Marks" section] (forge-edit-topic-marks)
: This command reads a list of marks for an existing topic in the minibuffer.

  Marks are like labels except that they are not shared with anyone else. To create a mark that topics can subsequently be marked with use the command forge-create-mark. Existing marks can be edited using the command forge-edit-mark.

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.

C-c C-e [on "Review-Requests" section] (forge-edit-topic-review-requests)
: This command reads a list of people who you would like to review an existing topic in the minibuffer.

C-c C-e [on "Note" section]
M-x forge-edit-topic-note
: This lets you edit your private note about a topic.

C-c C-k [on a comment section] (forge-delete-comment)
: This command deletes the comment at point.

m M [if enabled] (forge-merge)
N M [if enabled]
: This command merges the current pull-request using the forge's API. If there is no current pull-request or with a prefix argument, then it reads a pull-request to visit instead.

  The "merge method" to be used is read from the user.

  Use of this command is discouraged. Unless the remote repository is configured to disallow that, you should instead merge locally and then push the target branch. Forges detect that you have done that and respond by automatically marking the pull-request as merged.

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.

`C-c C-e` (`forge-post-dispatch`)
: This prefix command features the above two commands as suffixes, and when creating a pull-request also the following command. More suffix commands will likely be added in the future.

`C-c C-e d` (`forge-post-toggle-draft`)
: This command toggles whether the pull-request being created is a draft.

4.4 Miscellaneous

`N c f` (`forge-fork`)
: This command adds an additional remote to the current repository. The remote can either point at an existing repository or one that has to be created first by forking it to an account the user has access to.

  Currently this only supports Github and Gitlab.

`N a` (`forge-add-repository`)
: This command reads a repository from the user and adds it to the database. The repository can be provided as a URL, a name, or in the form OWNER/NAME. This is subject to `magit-clone-name-alist`.

  This command offers to either pull topics (now and in the future) or to only pull individual topics when the user invokes `forge-pull-topic`.

`N t t` (`forge-toggle-display-in-status-buffer`)
: This command toggles whether any topics are displayed in the current Magit status buffer.

`N t c` (`forge-toggle-closed-visibility`)
: This command toggles whether closed topics are shown in the Magit status buffer.

`forge-add-user-repositories` [Command]
: This command reads a host and a username from the user and adds all of that user's repositories on that host to the local database.

  This may take a while. Only Github is supported at the moment.

`forge-add-organization-repositories` [Command]
: This command reads a host and an organization from the user and adds all the organization's repositories on that host to the local database.

  This may take a while. Only Github is supported at the moment.

`forge-remove-repository` [Command]
: This command reads a repository and removes it from the local database.

`forge-remove-topic-locally` [Command]
: This command reads a topic and removes it from the local database. The topic is not removed from the forge and, if it is later modified, then it will be added to the database again.

Due to how the supported APIs work, it would be too expensive to automatically remove topics from the local database that were removed from the forge. The only purpose of this command is to allow you to manually clean up the local database.

`forge-reset-database` [Command]

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.

Appendix A FAQ

This section lists some frequently asked questions. Please also see `https://github.com/magit/forge/wiki/FAQ` for an extended list of common issues.

A.1 Is it possible to create a single pull-request without pulling everything?

Yes. `M-x forge-add-repository` offers to add a repository to the database without also fetching all pull-requests and issues.

A.2 `error in process filter: HTTP Error: 502, "Bad gateway"`

This is a frequently occurring error. Adding some formatting the full error is:

```
error in process filter: ghub--signal-error: HTTP Error: 502,
  "Bad gateway", "/graphql",
  ((data . "null")
   (errors ((message . "Something went wrong while executing your query.
      This may be the result of a timeout, or it could be a GitHub bug.
      Please include ‘CC2C:4FEA:A1771C1:CBF40CE:5C33F7E5‘
      when reporting this issue."))))
```

This indicates that something went wrong within Github's network. See `https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#5xx_server_errors`. The appended error message also says as much: "This may be the result of a timeout, or it could be a GitHub bug."

It appears that this happens more often in big repositories, especially during the initial pull, but this may be an illusion; it is known to also happens for smaller, incremental requests.

I believe that more data just means more requests and thus more opportunities for things to go wrong. It seems unlikely that this is due to us requesting too much data at once (in that case we would get a different error from GraphQL, not HTTP). When fetching lots of data, then we do not request one large response but make many requests and we then collect the many responses (pagination forces us to do that).

The timeout may be due to many requests from other people arriving at some Github-internal bottleneck at the same time, or it may be due to cold caches and overly aggressive timeouts. We just don't know; it's their infrastructure.

The second problem is that we currently simply error out if we get this error. This could be changed and eventually it will be, but for the time being your only option is to just try again, possibly repeatedly and possibly after a delay to give whatever congestion may exist on the other end a chance to clear or to give their caches a chance to warm up.

This was also discussed in `https://github.com/magit/forge/issues/20` and `https://github.com/magit/ghub/issues/83`.

Appendix B Keystroke Index

B

b f 11
b F 10

C

C-c C-c 16
C-c C-e 17
C-c C-e [on "Assignees" section] 16
C-c C-e [on "Draft" section] 16
C-c C-e [on "Labels" section] 16
C-c C-e [on "Marks" section] 16
C-c C-e [on "Note" section] 16
C-c C-e [on "Review-Requests" section] 16
C-c C-e [on "State" section] 16
C-c C-e [on "Title" section] 16
C-c C-e [on a post section] 15
C-c C-e d 17
C-c C-k 17
C-c C-k [on a comment section] 16
C-c C-n 15
C-c C-n [on "Issues" section] 15
C-c C-n [on "Pull requests" section] 15
C-c C-r 15
C-c C-v 13
C-c C-w 12

F

f n 9
f N 9

M

m M [if enabled] 16

N

N 9
N a 9, 17
N b i 12
N b I 12
N b p 12
N b P 12
N b r 12
N b t 12
N c f 17
N c i 15
N c p 15
N f f 9
N f n 9
N f t 10
N l i 14
N l n 14
N l p 14
N l r 14
N M [if enabled] 16
N r 7
N t c 17
N t t 17
N v i 13
N v p 13
N v t 13

R

RET 13

Z

Z n 12

Appendix C Function and Command Index

forge-add-organization-repositories 17
forge-add-repository 9, 17
forge-add-user-repositories 17
forge-branch-pullreq 10
forge-browse-dwim 12
forge-browse-issue 12
forge-browse-issues 12
forge-browse-pullreq 12
forge-browse-pullreqs 12
forge-browse-remote 12
forge-browse-topic 12
forge-browse-TYPE 12
forge-checkout-pullreq 11
forge-checkout-worktree 12
forge-create-issue 15
forge-create-post 15
forge-create-pullreq 15
forge-delete-comment 16
forge-dispatch 9
forge-edit-post 15
forge-edit-topic-assignees 16
forge-edit-topic-draft 16
forge-edit-topic-labels 16
forge-edit-topic-marks 16
forge-edit-topic-note 16
forge-edit-topic-review-requests 16
forge-edit-topic-state 16
forge-edit-topic-title 16
forge-forge.remote 7
forge-fork 17
forge-insert-assigned-issues 14
forge-insert-assigned-pullreqs 13
forge-insert-authored-issues 14
forge-insert-authored-pullreqs 13
forge-insert-issues 13
forge-insert-pullreqs 13
forge-insert-requested-reviews 13
forge-list-assigned-issues 14
forge-list-assigned-pullreqs 14
forge-list-authored-issues 14
forge-list-authored-pullreqs 14
forge-list-issues 14
forge-list-labeled-issues 14
forge-list-labeled-pullreqs 14
forge-list-notifications 14
forge-list-owned-issues 15
forge-list-owned-pullreqs 14
forge-list-pullreqs 14
forge-list-repositories 14
forge-list-requested-reviews 14
forge-merge 16
forge-post-cancel 17
forge-post-dispatch 17
forge-post-submit 16
forge-post-toggle-draft 17
forge-pull 9
forge-pull-notifications 9
forge-pull-topic 10
forge-remove-repository 17
forge-remove-topic-locally 17
forge-reset-database 18
forge-toggle-closed-visibility 13, 17
forge-toggle-display-in-status-buffer 17
forge-visit-issue 13
forge-visit-pullreq 13
forge-visit-topic 13
forge-visit-TYPE 13

Appendix D Variable Index

forge-alist 7
forge-checkout-worktree-read-directory-function 12
forge-owned-accounts 15
forge-owned-ignored 15
forge-topic-list-limit 13
forge.remote 7