Forge User and Developer Manual

for version 0.4.4

Jonas Bernoulli

Table of Contents

1 Introduction

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

Forge fetches issues, pull-requests and other data using the forge's API and stores the retrieved information in a local database. Additionally it fetches pull-request references using Git.

2 Initial Setup

Please first do the common setup below and then carefully follow the instructions for your forge instance. Once you have completed the setup, you can start tracking repositories (see Chapter 3 [Initial Pull], page 9).

If you run into difficulties during setup or the initial pull, then please also see Appendix A [How Forge Detection Works], page 24, and Section "Getting Started" in `ghub`.

Common Setup

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)
```

By default Forge adds some bindings to Magit keymaps and menus, and some sections to Magit buffers. If you would like to prevent that, you have to set `forge-add-default-bindings` and/or `forge-add-default-sections` to `nil`, before `magit` (not just `forge`) is loaded.

2.1 Setup for Github.com

Set your Username

First inform Forge about your `https://github.com` username:

```
git config --global github.user USERNAME
```

If you need to identify as another user in a particular repository, then you have to set that variable locally:

```
cd /path/to/repo
git config --local github.user USERNAME
```

Create and Store an Access Token

Visit `https://github.com/settings/tokens` in a browser to generate a new "classic" token using the `repo`, `user` and `read:org` scopes. Do not close the browser window just yet, because the token will only be shown once.

The built-in Auth-Source (`auth`) package is used to store the token generated in the previous step. The `auth-sources` variable controls how and where Auth-Source keeps its secrets. The default value is a list of three files: `("~/.authinfo" "~/.authinfo.gpg" "~/.netrc")`, but that can lead to confusing behavior, so you should make sure that only one of these files exists, and then you should also adjust the value of the variable to only ever use that file, for example:

```
(setq auth-sources '("~/.authinfo"))
```

In `~/.authinfo` secrets are stored in plain text. If you don't want that, then you should use the encrypted `~/.authinfo.gpg` instead:

```
(setq auth-sources '("~/.authinfo.gpg"))
```

Make sure you put one of these forms in your init file **and** to evaluate it in the current Emacs instance as well, by placing the cursor after the final closing parenthesis and typing `C-x C-e` (`eval-last-sexp`).

Next add a line like the following to the chosen file:

```
machine api.github.com login USERNAME^forge password TOKEN
```

- The value of `machine` must be `api.github.com`. Variations of this won't work.
- USERNAME must be the same as the value used for the `github.user` Git variable above. You **must** append `^forge` to that, without any space in between.
- TOKEN is the token you generated earlier.

Finish by typing `M-x auth-source-forget-all-cached RET`. If you don't do this, then Auth-Source may fail to look up the token.

2.2 Setup for Another Github Instance

Before you setup a Github instance that is not `https://github.com`, please set that up first. The setup for `https://github.com` is easier and if that works, but the setup for the other Github instance fails, then we can tentatively narrow the issue down to the parts that differ between `https://github.com` and other instances.

Tell Forge about the Instance

While Forge knows about `https://github.com`, it does not know about your other Github instances. Forge instances are configured using the option `forge-alist` (also see its docstring). The entry for `https://github.com` in that variable looks like this:

```
("github.com"                       ; GITHOST
 "api.github.com"                   ; APIHOST
 "github.com"                       ; WEBHOST and INSTANCE-ID
  forge-github-repository)          ; CLASS
```

You have to add an entry for your instance. For example, assuming you company uses `https://example.com`, this might be correct:

```
(push '("example.com"               ; GITHOST
        "api.example.com"           ; APIHOST
        "example.com"               ; WEBHOST and INSTANCE-ID
        forge-github-repository)    ; CLASS
      forge-alist)
```

Your company may use hostnames that follow a different format. You should be able to easily determine and verify GITHOST and WEBHOST, but determining APIHOST is more difficult; you might have to ask a coworker. APIHOST could be something like `api.example.com`, but it could also be something like `example.com/api`.

If the REST API's end point is `/v3` and the GraphQL API's end point is `/graphql`, then use something like `example.com/v3` as APIHOST. This is a historic accident. See `https://github.com/magit/forge/issues/174`.

We will use INSTANCE-ID (aka WEBHOST) and APIHOST below.

Set your Username

Inform Forge about your username for the Github instance in question:

```
git config --global github.INSTANCE-ID.user USERNAME
```

So if INSTANCE-ID is `example.com` and USERNAME is `tarsius` then use:

```
git config --global github.example.com.user tarsius
```

Create and Store an Access Token

Visit your forge in a browser. Follow a link to "Settings", from there to "Developer settings", from there to "Personal access tokens", and finally to "Tokens (classic)". On that page generate a new token using the `repo`, `user` and `read:org` scopes. Do not close the browser window just yet, because the token will only be shown once.

The built-in Auth-Source (`auth`) package is used to store the token generated in the previous step. The `auth-sources` variable controls how and where Auth-Source keeps its secrets. The default value is a list of three files: `("~/.authinfo" "~/.authinfo.gpg" "~/.netrc")`, but that can lead to confusing behavior, so you should make sure that only one of these files exists, and then you should also adjust the value of the variable to only ever use that file, for example:

```
(setq auth-sources '("~/.authinfo"))
```

In `~/.authinfo` secrets are stored in plain text. If you don't want that, then you should use the encrypted `~/.authinfo.gpg` instead:

```
(setq auth-sources '("~/.authinfo.gpg"))
```

Make sure you put one of these forms in your init file **and** to evaluate it in the current Emacs instance as well, by placing the cursor after the final closing parenthesis and typing `C-x C-e` (`eval-last-sexp`).

Next add a line like the following to the chosen file:

```
machine APIHOST login USERNAME^forge password TOKEN
```

- APIHOST must be the same as the second element of the entry we added to `forge-alist`. In the above example that would be `api.example.com`. Do **not** instead use GITHOST or INSTANCE-ID (aka WEBHOST).
- USERNAME must be the same username you used above as the value of the Git variable. You **must** append `^forge` to that, without any space in between.
- TOKEN is the token you generated earlier.

Finish by typing `M-x auth-source-forget-all-cached RET`. If you don't do this, then Auth-Source may fail to look up the token.

2.3 Setup for Gitlab.com

Set your Username

First inform Forge about your `https://gitlab.com` username:

```
git config --global gitlab.user USERNAME
```

If you need to identify as another user in a particular repository, then you have to set that variable locally:

```
cd /path/to/repo
```

```
git config --local gitlab.user USERNAME
```

Create and Store an Access Token

Visit `https://gitlab.com/-/user_settings/personal_access_tokens` in a browser to generate a new token using the `api`, `read_api` and `read_user` scopes. Do not close the browser window just yet, because the token will only be shown once.

The built-in Auth-Source (`auth`) package is used to store the token generated in the previous step. The `auth-sources` variable controls how and where Auth-Source keeps its secrets. The default value is a list of three files: `("~/.authinfo" "~/.authinfo.gpg" "~/.netrc")`, but that can lead to confusing behavior, so you should make sure that only one of these files exists, and then you should also adjust the value of the variable to only ever use that file, for example:

```
(setq auth-sources '("~/.authinfo"))
```

In `~/.authinfo` secrets are stored in plain text. If you don't want that, then you should use the encrypted `~/.authinfo.gpg` instead:

```
(setq auth-sources '("~/.authinfo.gpg"))
```

Make sure you put one of these forms in your init file **and** to evaluate it in the current Emacs instance as well, by placing the cursor after the final closing parenthesis and typing `C-x C-e` (`eval-last-sexp`).

Next add a line like the following to the chosen file:

```
machine gitlab.com/api/v4 login USERNAME^forge password TOKEN
```

- The value of `machine` must be `gitlab.com/api/v4`. Variations of this won't work.
- USERNAME must be the same as the value used for the `gitlab.user` Git variable above. You **must** append `^forge` to that, without any space in between.
- TOKEN is the token you generated earlier.

Finish by typing `M-x auth-source-forget-all-cached RET`. If you don't do this, then Auth-Source may fail to look up the token.

2.4 Setup for Another Gitlab Instance

Before you setup a Gitlab instance that is not `https://gitlab.com`, please set that up first. The setup for `https://gitlab.com` is easier and if that works, but the setup for the other Gitlab instance fails, then we can tentatively narrow the issue down to the parts that differ between `https://gitlab.com` and other instances.

Tell Forge about the Instance

While Forge knows about `https://gitlab.com` (and a few other well-known instances, see its value) it has to be taught about other Gitlab instances. Forge instances are configured using the option `forge-alist` (also see its docstring). The entry for `https://gitlab.com` in that variable looks like this:

```
("gitlab.com"                      ; GITHOST
 "gitlab.com/api/v4"               ; APIHOST
 "gitlab.com"                      ; WEBHOST and INSTANCE-ID
 forge-gitlab-repository)          ; CLASS
```

For historic reasons, APIHOST actually has to be a host followed by a path.

You have to add an entry for your instance. For example, assuming you company/organisation uses `https://example.com`, this might be correct:

```
(push '("example.com"                ; GITHOST
        "example.com/api/v4"         ; APIHOST
        "example.com"                ; WEBHOST and INSTANCE-ID
        forge-gitlab-repository)     ; CLASS
      forge-alist)
```

Your company may use hostnames that follow a different format. You should be able to easily determine and verify GITHOST and WEBHOST, but determining APIHOST is more difficult; you might have to ask a colleague.

We will use INSTANCE-ID (aka WEBHOST) and APIHOST below.

Set your Username

Inform Forge about your username for the Gitlab instance in question:

```
git config --global gitlab.INSTANCE-ID.user USERNAME
```

So if INSTANCE-ID is `example.com` and USERNAME is `tarsius` then use:

```
git config --global gitlab.example.com.user tarsius
```

Create and Store an Access Token

Visit your forge in a browser. Follow a link to "Preferences" and from there to "Access Tokens". On that page generate a new "Personal access token" using the `api`, `read_api` and `read_user` scopes. Do not close the browser window just yet, because the token will only be shown once.

The built-in Auth-Source (`auth`) package is used to store the token generated in the previous step. The `auth-sources` variable controls how and where Auth-Source keeps its secrets. The default value is a list of three files: `("~/.authinfo" "~/.authinfo.gpg" "~/.netrc")`, but that can lead to confusing behavior, so you should make sure that only one of these files exists, and then you should also adjust the value of the variable to only ever use that file, for example:

```
(setq auth-sources '("~/.authinfo"))
```

In `~/.authinfo` secrets are stored in plain text. If you don't want that, then you should use the encrypted `~/.authinfo.gpg` instead:

```
(setq auth-sources '("~/.authinfo.gpg"))
```

Make sure you put one of these forms in your init file **and** to evaluate it in the current Emacs instance as well, by placing the cursor after the final closing parenthesis and typing `C-x C-e` (`eval-last-sexp`).

Next add a line like the following to the chosen file:

```
machine APIHOST login USERNAME^forge password TOKEN
```

- APIHOST must be the same as the second element of the entry we added to `forge-alist`. In the above example that would be `example.com/api/v4`. Do **not** instead use GITHOST or INSTANCE-ID (aka WEBHOST).

- USERNAME must be the same username you used above as the value of the Git variable. You **must** append `^forge` to that, without any space in between.
- TOKEN is the token you generated earlier.

Finish by typing `M-x auth-source-forget-all-cached RET`. If you don't do this, then Auth-Source may fail to look up the token.

2.5 Setup a Partially Supported Host

Forge currently only supports the Github and Gitlab APIs.

It does however partially support a few additional forge types (see Section B.2 [Partially Supported Forges], page 27) and other lighter weight software used to host Git repositories, which also provide a web interfaces (see Section B.3 [Supported Semi-Forges], page 27). Forge doesn't use the APIs of such forges, but registering the host and adding repositories to the local database at least enables the use of commands such as `forge-browse`.

Tell Forge about the Instance

A few hosts, which use partially supported forge types, are available out-of-the-box, because they have an entry in the default value of option `forge-alist` (also see its docstring). For example, the entry for `https://github.com` in that variable looks like this:

```
("codeberg.org"                     ; GITHOST
 "codeberg.org/api/v1"              ; APIHOST
 "codeberg.org"                     ; WEBHOST and INSTANCE-ID
 forge-gitea-repository)            ; CLASS
```

To be able to add repositories from a, so far, unknown forge instance to your local database, you have to add an entry for that instance to `forge-alist`. For example, assuming you use another Gitea instance, hosted at `https://example.com`, this might be correct:

```
(push '("example.com"               ; GITHOST
        "example.com/api/v1"        ; APIHOST
        "example.com"               ; WEBHOST and INSTANCE-ID
        forge-gitea-repository)     ; CLASS
      forge-alist)
```

Look at `forge-alist` entries of other hosts using the same forge type as the instance you are configuring, to see what format **might** be appropriate. You should be able to easily determine and verify GITHOST and WEBHOST, but determining APIHOST is more difficult; you might have to ask a colleague. APIHOST could be something like `example.com/api/vi`, but it could also be something like `api.example.com`.

Add Support for Additional Forge Types

For each fully or partially supported forge type, Forge defines at least a class. The following example is taken from `forge-semi.el`:

```
(defclass forge-cgit-repository (forge-noapi-repository)
  ((commit-url-format :initform "https://%h/%p.git/commit/?id=%r")
   (branch-url-format :initform "https://%h/%p.git/log/?h=%r")
   (remote-url-format :initform "https://%h/%p.git/about"))
  "Cgit from https://git.zx2c4.com/cgit/about.
```

```
Different hosts use different url schemata, so we need multiple
classes.  See their definitions in \"forge-semi.el\".")
```

Once you add a host using that class to `forge-alist` and then a repository from that host to the local database, you will be able to use commands such as `forge-browse-branch` (but not much more).

If you want to add a repository from another host, which happens to use another software or another URL schemata, then you might have to define an additional class first. See `forge-semi.el` for simple examples and grep for `defclass forge-.*-repository` for more complex ones.

3 Initial Pull

To start using Forge in a certain repository, visit the Magit status buffer for that repository and type `N / a` (`forge-add-repository`). You are given a choice to pull all topics, all topics that were updated after a certain date, or only individual topics.

Beside adding the repository to the database, this also adds a new value to the Git variable `remote.<remote>.fetch`, which causes all pull-request refs (`+refs/pull/*/head:refs/pullreqs/*` for Github) to be fetched by Git.

Note that it is possible to use the same command to add any repository from a supported forge to the database, without cloning the Git repository first.

The initial fetch can take a while but most of the work 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.

4 Getting Started

Much like Git stores information in a local repository and does not require a constant internet connection, Forge retrieves additional information using a forge's API and stores that in a local database.

Forge's equivalent of `git clone` is `forge-add-repository`, which has to be run, before most of Forges features become available in the local clone of a Git repository.

N / a (`forge-add-repository`)

This command guides the user through the process of adding a repository to the local database.

Note that it is possible to add a repository to the local database, without pulling all the data, which is useful if you just want to create a single issue or pull-request in a repository, but are not interested in existing topics, e.g., because you do not regularly contribute to that repository.

Also note that you can add a repository to the local database, even if no local Git clone exists.

Like with Git, you have to explicitly pull remote changes, at your leisure, using `forge-pull`.

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.

If the current repository isn't being tracked in the local database yet, then this command pivots to behave like `forge-add-repository`.

Forge adds two additional sections to Magit's status buffer, which list open and/or pending issues and pull-requests. Typing `RET`, while the cursor is on a topic section, shows more information about that topic in a separate buffer. Typing `RET` on a topic list section, shows that list in a separate buffer, where you can apply different filters.

The other main entry point to the functionality provided by Forge is the `forge-dispatch` menu.

N (`forge-dispatch`)

This prefix command is available in all Magit buffers and provides access to most of the available Forge commands. See the following sections for information about the available commands.

5 Lists and Menus

Topics are listed in two sections in Magit's status buffer, but can also be listed in dedicated buffers. Likewise individual topics can be visited in separate buffers. In both cases this can be done by placing the cursor on the respective section in the status buffer and typing `RET`, or by invoking the appropriate command from Forge's main menu, on `N` (`forge-dispatch`).

List commands and corresponding menu commands exist for topics, notifications and repositories, but there isn't always an exclusive mapping from menu to buffer. The main menu (`forge-dispatch`), the configuration menu (`forge-configure`), the menu which controls the current topic or the topic at point (`forge-topic-menu`), and the menu which controls the topics listed in the current buffer (`forge-topics-menu`), are useful in more than one major mode.

All of these menus feature bindings to directly switch to the other appropriate menus. So it is enough to remember that `N` always brings up the dispatch menu; you can always navigate to another menu from there.

`C-c C-c` brings up the most appropriate menu for the current buffer. In Magit's status buffer the most appropriate menu is Magit's own dispatch menu (`magit-dispatch`), so here the quickest way to invoke Forge's dispatch menu is `N`. Even in Magit's status buffer, when the cursor is an individual topic or on a topic list section, `C-c C-c` opens the respective menu (`forge-topics-menu` or `forge-topic-menu`).

The following sections describe most of the available menu and list commands. For `forge-topic-menu`, see Chapter 8 [Editing Topics], page 17.

Dispatch and configuration menus

N (`forge-dispatch`)
: This prefix menu command is available in all Magit buffers and provides access to most of the available Forge commands. See the following sections for information about the available commands.

N m c (`forge-configure`)
: This command displays a menu used to configure the current repository and some global settings as well.

Topic menu and list commands

N m f (`forge-topics-menu`)
C-c C-c [in topics list buffer/section]
: This command displays a menu used to control the list of topics displayed in the current buffer.

  Note that this command can not only be used in buffers dedicated to listing topics, but also in Magit's status buffer.

N l t (`forge-list-topics`)
: This command lists the current repository's issues in a separate buffer. If the list buffer already exists, this command only ensures that all types of topics are listed. If any other filters are in effect, they are left intact. TODO fix preserving type

RET [on "Issues" status section] (forge-list-issues)
: This command lists the current repository's issues in a separate buffer. If the list buffer already exists, this command limits the list to issues. If any other filters are in effect, they are left intact.

RET [on "Pull requests" status section] (forge-list-pullreqs)
: This command lists the current repository's pull-requests in a separate buffer. If the list buffer already exists, this command limits the list to pull-requests. If any other filters are in effect, they are left intact.

N l g (forge-list-global-topics)
: This command lists topics across all tracked repository. If the list buffer already exists, filters except for the type filter are left in effect.

forge-list-global-issues [Command]
: This command lists issues across all tracked repository. If the list buffer already exists, filters except for the type filter are left in effect.

forge-list-global-pullreqs [Command]
: This command lists pull-requests across all tracked repository. If the list buffer already exists, filters except for the type filter are in effect.

Notification menu and list commands

N m n (forge-notifications-menu)
C-c C-c This command displays a menu used to control the list of notifications displayed in the current buffer.

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

Repository menu and list commands

N m r (forge-repositories-menu)
C-c C-c This command displays a menu used to control the list of repositories displayed in the current buffer.

N l r (forge-list-repositories)
: This command lists all known repositories in a separate buffer. Here "known" means that an entry exists in the local database.

RET [on repository] (forge-visit-this-repository)
: This commands visits the repository at point in a separate buffer.

o [in forge-repositories-menu] (forge-list-owned-repositories)
: This command lists all known repositories that belong to the user in a separate buffer. Here "known" means that an entry exists in the local database. Only Github is supported for now.

The below options controls which repositories are considered to be owned by the user. They are additionally used by forge-fork.

`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.

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`.

Exiting menus and lists

To exit a menu, type `C-g`. If the menu was invoked from another menu and that menu is useful in the current buffer, then that menu becomes active again. If that happens and you actually want to quit all menus, then just type `C-g` again. You can also directly exit all menus by using `C-q`, instead of `C-g`.

Type `q` to quit not only the menu, but also the list or topic detail buffer. That binding is also available when no menu is active, in which case it will simply quit the buffer. When invoked from a menu, then this binding may return to another list buffer, in which case some menu may also remain active.

Default topic filters

`forge-status-buffer-default-topic-filters` [User Option]

This option specifies the filters initially used to limit topics listed in topic list buffers.

`forge-status-buffer-default-topic-filters` [User Option]

This option specifies the filters initially used to limit topics listed in Magit status buffers.

Also see [Topic sections in Magit status buffers], page 13.

Topic sections in Magit status buffers

Forge arranges for certain issues and pull-requests to be list in Magit status buffers, by adding the following functions to `magit-status-sections-hook`.

Which topics are listed initially is customizable using option `forge-status-buffer-default-topic-filters` and can be changed temporarily for the current buffer, using `N m f` ('forge-topics-menu').

`forge-insert-issues` [Function]

This function inserts a list of issues, by default a list of "active" issues.

`forge-insert-pullreqs` [Function]

This function inserts a list of pull-requests, by default a list of "active" pull-requests.

Forge used to provide additional functions to insert hard-coded topic subsets, but they were removed in favor of the more flexible approach described above. If you miss the removed sections, you can use the new `forge-insert-topics` helper function to define your own section inserter functions. See its docstring for more information.

If you don't want any topic list sections to be displayed in Magit status buffers, set `forge-add-default-sections` to `nil` before `magit` is loaded.

6 Visiting Topics

The commands, accessible from `forge-topic-menu` (on `C-return`), act on the topic at point; so this menu is useful in buffers dedicated to listing topics and notifications (which correspond to topics), but also in the status buffer (which also lists topics). In buffers dedicated to showing details about a single topic, these commands act on that topic; so this menu can be used there too.

To switch to this menu from another menu use `m s`. If the cursor is on a topic or the current buffer visits a topic.

To display details about a topic in a separate buffer and at the same time display the topic menu, invoke `forge-topic-menu` with a prefix argument, i.e., `C-u RET`.

RET [on topic] (`forge-visit-this-topic`)
: This commands visits the topic at point in a separate buffer. When invoked with a prefix argument then it not only visits the topic in a separate buffer, it at the same time displays

N v t (`forge-visit-topic`)
N v i (`forge-visit-issue`)
N v p (`forge-visit-pullreq`)
: These commands read a topic, issue or pull-request and visit it in a separate buffer.

C-c C-o (`forge-browse`)
o [on topic in topic list] (`forge-browse-this-topic`)
o [on repository in repository list] (`forge-browse-this-repository`)
: These commands visit the topic, issue(s), pull-request(s), post, branch, commit, remote or repository at point in a browser.

`forge-browse-commit` [Command]

`forge-browse-branch` [Command]

`forge-browse-repository` [Command]

N b t (`forge-browse-topic`)
N b i (`forge-browse-issue`)
N b p (`forge-browse-pullreq`)
N b r (`forge-browse-remote`)
N b I (`forge-browse-issues`)
N b P (`forge-browse-pullreqs`)
: These commands read a topic, issue(s), pull-request(s), branch, commit, remote or repository, and open it in a browser.

7 Creating Topics and Posts

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

Creating a new topic or post and editing an existing post work similarly to now creating a new commit or editing the message of an existing commit works in Magit. 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 drop you into such a buffer.

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.

C-c C-n (forge-create-post)
C-c C-r
: This command creates a new post on an existing topic. It is only available in buffers that visit an existing topic.

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

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.

8 Editing Topics

Many details about a topic can be changed from the buffer that visits that topic, but also from topic lists, if the cursor is placed on the topic to be edited. However, to edit the posts on a topic, the topic has to be visited in its own buffer.

C-c C-e [on a post section] (`forge-edit-post`)
: This command visits an existing post in a separate buffer, it can only be invoked from a topic buffer, when the cursor is on the post to be edited.

  Editing an existing post is similar to creating a new post, as described in the previous section.

C-c C-k [on a post section] (`forge-delete-comment`)
: This command deletes the post the cursor is on. The initial message that was written when the topic was created, cannot be deleted, only replies to that.

N m s (`forge-topic-menu`)
C-<return> [on a topic section]
: This command displays a menu used to edit details about the topic the cursor is on or that is being visited in the current buffer. E.g., it can be used to change the status of the topic or to apply labels to it. Additionally it features a few commands that act on that topic.

Details about a topic, such as its status and labels, can alternatively be edited by visiting the topic in its own buffer, navigating to the header that displays the detail and then typing `C-c C-e`. This older approach is still available, but it is usually much faster to use the menu.

9 Pulling

The commands that fetch forge data are available the Forge's main menu (`forge-dispatch` on `N`) and from the same menu (`magit-fetch` on `f`) that is used to fetch Git data. If `magit-pull-or-fetch` is non-nil, then they are also available from the `magit-pull` menu (on `F`).

With Git you have to explicitly pull Git data to make it available in the local repository. Forge works the same; you have to explicitly pull to pull data using the forge's API and storing in the local database. This is less disruptive, more reliable, familiar and easier to understand than if Forge pulled 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 and 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.

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.

  If the current repository is still untracked locally, or the current repository cannot be determined, this command instead behaves like `forge-add-repository`, i.e., it adds the repository to the database and then performs the initial pull.

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 notifications fetches associated topics even for repositories that you have not yet explicitly added to the local database.

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. This is useful if you chose to not fetch all topics when you added the repository using `forge-add-repository`.

10 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 the new worktree directory where it checks out the 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.

11 Miscellaneous Commands

N M (`forge-merge`)
m 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.

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 - H (`forge-toggle-topic-legend`)
: This command toggle whether to show a legend for faces used in topic menus and lists.

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

C-c C-w (`forge-copy-url-at-point-as-kill`)
: This command copies the url for the topic, issue(s), pull-request(s), post, branch, commit, remote or repository to the kill-ring.

  This determines the url the same way as `forge-browse` does, but then adds it to the kill-ring, instead of visiting it in a browser.

M b r (`forge-rename-default-branch`)
: This command rename the default branch to a new name read from the user.

  This changes the name on the upstream remotely and locally, and update the upstream remotes of local branches accordingly.

`forge-add-pullreq-refspec` [Command]
: This command configures Git to fetch all pull-requests.

  This is done by adding `+refs/pull/*/head:refs/pullreqs/*` to the value of `remote.REMOTE.fetch`, where REMOTE is the upstream remote.

`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.

12 Miscellaneous Options

`forge-database-file` [User Option]
This option specifies the file used to store the forge database.

`forge-topic-repository-slug-width` [User Option]
This option specifies the width of repository slugs (i.e., "OWNER/NAME").

`forge-buffer-draft-p` [User Option]
This option controls whether new pull-requests start out as drafts by default.

The buffer-local value of this variable is used to keep track of the draft status of the current pull-request.

`forge-repository-list-columns` [User Option]
This option specifies the list of columns displayed when listing repositories.

Each element has the form `(HEADER SOURCE WIDTH SORT PROPS)`.

HEADER is the string displayed in the header. WIDTH is the width of the column. SOURCE is used to get the value, it has to be the name of a slot of `forge-repository` or a function that takes such an object as argument. SORT is a boolean or a function used to sort by this column. Supported PROPS include `:right-align` and `:pad-right`.

`forge-limit-topic-choices` [User Option]
This option controls whether to initially limit completion candidates to active topics.

`forge-post-heading-format` [User Option]
This option specifies the format for post headings in topic view.

The following `%`-sequences are supported:

- `%a` The forge nickname of the author.
- `%c` The absolute creation date.
- `%C` The relative creation date.

`forge-post-fill-region` [User Option]
This option controls whether to call `fill-region` before displaying forge posts.

`forge-bug-reference-hooks` [User Option]
This option lists the hooks to which `forge-bug-reference-setup` is added. It has to be customized before `forge` is loaded, or it won't take effect.

Appendix A How Forge Detection Works

Forge uses the Ghub package to communicate with forge APIs. For more information about Ghub, see `ghub`.

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 is 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 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 WEBHOST CLASS)`.

- GITHOST is the host used to access repositories on the forge using Git.
- APIHOST is the host used to access the forge's API. For some forges the isn't just a host, but a host followed by the path to the API's endpoint.
- WEBHOST is the host used to access repositories on this forge using a browser. The IDs used to identify repositories from the forge in the local database also derives from this value.
- CLASS is the class to be used for repositories from the forge.

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.
- WEBHOST and CLASS cannot be changed once you have added one or more repositories from a forge. Changing GITHOST and/or APIHOST may be possible, but should seldom be necessary.

Appendix B 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 Appendix A [How Forge Detection Works], page 24).

For more details about the caveats mentioned below (and some others) see also Chapter 4 [Getting Started], page 10.

B.1 Supported Forges

B.1.1 Github

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

B.1.1.1 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.

B.1.1.2 Github Hosts

- `https://github.com`

B.1.2 Gitlab

B.1.2.1 Gitlab Caveats

- Forge cannot provide notifications because the Gitlab API does not expose those.

B.1.2.2 Gitlab Hosts

- `https://gitlab.com`

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

B.2 Partially Supported Forges

B.2.1 Gitea https://gitea.io

This is the next forge whose API will be supported.

B.2.1.1 Gitea Hosts

- `https://codeberg.org`

B.2.2 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.

B.2.2.1 Gogs Hosts

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

B.2.3 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.

B.2.3.1 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`.

B.2.3.2 Bitbucket Hosts

- `https://bitbucket.org`

B.3 Supported Semi-Forges

B.3.1 Gitweb https://git-scm.com/docs/gitweb

B.3.1.1 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).

B.3.2 Cgit https://git.zx2c4.com/cgit/about

B.3.2.1 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.

B.3.2.2 Cgit Hosts

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

B.3.3 Stgit https://codemadness.org/git/stagit/file/README.html

B.3.3.1 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).

B.3.3.2 Stgit Hosts

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

B.3.4 Srht https://meta.sr.ht

B.3.4.1 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).

B.3.4.2 Srht Hosts

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

Appendix C FAQ

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

C.1 `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. Unfortunately the reason given is rather vague, but I believe this usually happens when there are topics with one or two magnitudes more posts than usual, which can cause GraphQL responses to become huge.

This can be countered in the affected repository by setting the Git variable `forge.graphqlItemLimit`:

```
git config --local forge.graphqlItemLimit 20
```

The default is specified using the `ghub-graphql-items-per-request`, which defaults to 50 (down from Github's default and maximum of 100).

Fetching less items per request results in more requests, which slows down the process, which is why the default should not be too small, but for some repositories a more aggressive limit is needed.

Appendix D Keystroke Index

Appendix E Function and Command Index

Appendix F Variable Index