Patchwork

Release 3.1.3.alpha.0

Patchwork Developers

Jan 25, 2023

USAGE DOCUMENTATION

Patchwork is a patch tracking system for community-based projects. It is intended to make the patch management process easier for both the project's contributors and maintainers, leaving time for the more important (and more interesting) stuff.

Patches that have been sent to a mailing list are 'caught' by the system, and appear on a web page. Any comments posted that reference the patch are appended to the patch page too. The project's maintainer can then scan through the list of patches, marking each with a certain state, such as Accepted, Rejected or Under Review. Old patches can be sent to the archive or deleted.

Currently, Patchwork is being used for a number of open-source projects, mostly subsystems of the Linux kernel. Although Patchwork has been developed with the kernel workflow in mind, the aim is to be flexible enough to suit the majority of community projects.

CHAPTER

ONE

OVERVIEW

The key concepts or models of Patchwork are outlined below.

1.1 Projects

Projects typically represent a software project or sub-project. A Patchwork server can host multiple projects. Each project can have multiple maintainers. Projects usually have a 1:1 mapping with a mailing list, though it's also possible to have multiple projects in the same list using the subject as filter. Patches, cover letters, and series are all associated with a single project.

1.2 People

People are anyone who has submitted a patch, cover letter, or comment to a Patchwork instance.

1.3 Users

Users are anyone who has created an account on a given Patchwork instance.

1.3.1 Standard Users

A standard user can associate multiple email addresses with their user account, create bundles and store TODO lists.

1.3.2 Maintainers

Maintainers are a special type of user that with permissions to do certain operations that regular Patchwork users can't. Patchwork maintainers usually have a 1:1 mapping with a project's code maintainers though this is not necessary.

The operations that a maintainer can invoke include:

- Change the state of a patch
- Archive a patch
- Delegate a patch, or be delegated a patch

1.4 Submissions

Patchwork captures three types of mail to mailing lists: patches, cover letters, and replies to either patches or cover letters, a.k.a. comments. Any mail that does not fit one of these categories is ignored.

1.4.1 Patches

Patches are the central object in Patchwork structure. A patch contains both a diff and some metadata, such as the name, the description, the author, the version of the patch etc. Patchwork stores not only the patch itself but also various metadata associated with the email that the patch was parsed from, such as the message headers or the date the message itself was received.

1.4.2 Cover Letters

Cover letters provide a way to offer a "big picture" overview of a series of patches. When using Git, these mails can be recognised by way of their `0/N` subject prefix, e.g. `[00/11] A sample series`. Like patches, Patchwork stores not only the various aspects of the cover letter itself, such as the name and body of the cover letter, but also various metadata associated with the email that the cover letter was parsed from.

1.5 Comments

Comments are replies to a submission - either a patch or a cover letter. Unlike a Mail User Agent (MUA) like Gmail, Patchwork does not thread comments. Instead, every comment is associated with either a patch or a cover letter, and organized by date.

1.6 Patch Metadata

Patchwork allows users to store various metadata against patches. This metadata is only configurable by a maintainer.

1.6.1 States

States track the state of patch in its lifecycle. States vary from project to project, but generally a minimum subset of "new", "rejected" and "accepted" will exist.

1.6.2 Delegates

Delegates are Patchwork users who are responsible for both reviewing a patch and setting its eventual state in Patchwork. This makes them akin to reviewers in other tools. Delegation works particularly well for larger projects where various subsystems, each with their own maintainer(s), can be identified. Only one delegate can be assigned to a patch.

Note: Patchwork supports automatic delegation of patches. Refer to *Autodelegation* for more information.

1.6.3 Tags

Tags are specially formatted metadata appended to the foot the body of a patch or a comment on a patch. Patchwork extracts these tags at parse time and associates them with the patch. You add extra tags to an email by replying to the email. The following tags are available on a standard Patchwork install:

Acked-by:
For example:

```
Acked-by: Stephen Finucane <stephen@that.guru>
```

Tested-by:
For example:

```
Tested-by: Stephen Finucane <stephen@that.guru>
```

Reviewed-by:
For example:

```
Reviewed-by: Stephen Finucane <stephen@that.guru>
```

The available tags, along with the significance of said tags, varies from project to project and Patchwork instance to Patchwork instance. The kernel project documentation provides an overview of the supported tags for the Linux kernel project.

1.6.4 Checks

Checks store the results of any tests executed (or executing) for a given patch. This is useful, for example, when using a continuous integration (CI) system to test patches. Checks have a number of fields associated with them:

Context
A label to discern check from the checks of other testing systems

Description
A brief, optional description of the check

Target URL
A target URL where a user can find information related to this check, such as test logs.

State
The state of the check. One of: `pending`, `success`, `warning`, `fail`

User
The user creating the check

Note: Checks can only be created through the Patchwork APIs. Refer to *../api* for more information.

1.7 Comment Metadata

Like patches, Patchwork allows users to store various bits of metadata against comments.

1.7.1 Action required

New in version 3.1.0.

Patchwork allows users to set an "action required" flag on patch and cover letter comments. This flag can be set by maintainers or by the users submitting the cover letters. Once the submitter has provided the required information, either the submitter or a maintainer can mark the comment as "addressed". This provides a more granular way of tracking work items than patch states.

Note: Users can indicate that a comment requires an action using a custom mail header. For more information, refer to *Hint Headers*.

1.8 Collections

Patchwork provides a number of ways to store groups of patches. Some of these are automatically generated, while others are user-defined.

1.8.1 Series

Series are groups of patches, along with an optional cover letter. Series are mostly dumb containers, though they also contain some metadata themselves such as a version (which is inherited by the patches and cover letter) and a count of the number of patches found in the series.

1.8.2 Bundles

Bundles are custom, user-defined groups of patches. Bundles can be used to keep patch lists, preserving order, for future inclusion in a tree. There's no restriction of number of patches and they don't even need to be in the same project. A single patch also can be part of multiple bundles at the same time. An example of Bundle usage would be keeping track of the Patches that are ready for merge to the tree.

1.8.3 To-do Lists

Patchwork users can store a to-do list of patches.

1.9 Events

Events are raised whenever patches are created or modified.

All events have a number of common properties, along with some event-specific properties:

`category`
: The type of event

`project`
: The project this event belongs to

`date`
: When this event was created

`actor`
: The user, if any, that caused/created this event

`payload`
: Additional information

1.9.1 Cover Letter Created

Category
: `cover-created`

Sent when a cover letter is created.

`cover`
: Created cover letter

1.9.2 Patch Created

Category
: `patch-created`

Sent when a patch is created.

`patch`
: Created patch

1.9.3 Patch Completed

Category
: `patch-completed`

Sent when a patch in a series has its dependencies met, or when a patch that is not in a series is created (since that patch has no dependencies).

`patch`
: Completed patch

`series`
: Series from which patch dependencies were extracted, if any

1.9.4 Patch Delegated

Category
`patch-delegated`

Sent when a patch's delegate is changed.

patch
Updated patch

previous
Previous delegate, if any

current
Current delegate, if any

1.9.5 Patch State Changed

Category
`patch-state-changed`

Sent when a patch's state is changed.

patch
Updated patch

previous
Previous state

current
Current state

1.9.6 Check Created

Category
`check-created`

Sent when a patch check is created.

check
Created check

1.9.7 Series Created

Category
`series-created`

Sent when a series is created.

series
Created series

1.9.8 Series Completed

Category
`series-completed`

Sent when a series is completed.

`series`
Completed series

1.9.9 What's Not Exposed

- Bundles

  We don't expose an "added to bundle" event as it's unlikely that this will be useful to either users or CI setters.

- Comments

  Like Bundles, there likely isn't much value in exposing these via the API.

CHAPTER

TWO

DESIGN

Patchwork should supplement mailing lists, not replace them

Patchwork isn't intended to replace a community mailing list; that's why you can't comment on a patch in Patchwork. If this were the case, then there would be two forums of discussion on patches, which fragments the patch review process. Developers who don't use Patchwork would get left out of the discussion.

Don't pollute the project's changelogs with Patchwork poop

A project's changelogs are valuable - we don't want to add Patchwork-specific metadata.

Patchwork users shouldn't require a specific version control system

Not everyone uses git for kernel development, and not everyone uses git for Patchwork-tracked projects.

It's still possible to hook other programs into Patchwork, using various *clients* or the APIs directly.

CHAPTER

THREE

AUTODELEGATION

Autodelegation allows patches to be automatically delegated to a user based on the files modified by the patch. To do this, a number of rules can be configured in the project administration page. This can usually be found at:

/admin/patchwork/project/<project_id>/change

Note: Autodelegation can only be configured by Patchwork administrators, i.e. those that can access the 'admin' panel. If you require configuration of autodelegation rules on a local instance, contact your Patchwork administrator.

In this section there are the following fields:

User
The patchwork user that should be autodelegated to the patch

Priority
The priority of the rule relative to other patches. Higher values indicate higher priority. If two rules have the same priority, ordering will be based on the path.

Path
A path in fnmatch format. The fnmatch library allows for limited, Unix shell-style wildcarding. Filenames are extracted from patch lines beginning with `---` or `+++`.

You can simply use a bare path:

```
patchwork/views/about.py
```

Or it is also possible to use relative paths, such as:

```
*/manage.py
```

Rules are configured by setting the above fields and saving the rules. These rules will be applied at patch parse time.

CHAPTER

FOUR

HINT HEADERS

Patchwork provides a number of special email headers to control how a patch is handled when it is received. The examples provided below use *git-send-email*, but custom headers can also be set when using tools like *mutt*.

X-Patchwork-Hint
: Valid values: ignore

  When set, this header will ensure the provided email is not parsed by Patchwork. For example:

```
$ git send-email --add-header="X-Patchwork-Hint: ignore" master
```

X-Patchwork-Delegate
: Valid values: An email address associated with a Patchwork user

  If set and valid, the user corresponding to the provided email address will be assigned as the delegate of any patch parsed. If invalid, it will be ignored. For example:

```
$ git send-email --add-header="X-Patchwork-Delegate: a@example.com" master
```

X-Patchwork-State
: Valid values: Varies between deployments. This can usually be one of "Accepted", "Rejected", "RFC" or "Awaiting Upstream", among others.

  If set and valid, the state provided will be assigned as the state of any patch parsed. If invalid, it will be ignored. For example:

```
$ git send-email --add-header="X-Patchwork-State: RFC" master
```

X-Patchwork-Action-Required
: Valid values: <none, value is ignored>

  When set on a reply to an existing cover letter or patch, this header will mark the comment as "unaddressed". The comment can then be addressed using the API or web UI. For more details, refer to *Comment Metadata*.

CHAPTER

FIVE

CLIENTS

A number of clients are available for interacting with Patchwork's various APIs.

5.1 pwclient

Changed in version 2.2: **pwclient** was previously provided with Patchwork. It has been packaged as a separate application since Patchwork v2.2.0.

The **pwclient** application can be used to interact with Patchwork from the command line. Functionality provided by **pwclient** includes:

- Listing patches, projects, and checks
- Downloading and applying patches to a local code base
- Modifying the status of patches
- Creating new checks

More information on **pwclient**, including installation and usage instructions, can be found in the documentation and the GitHub repo.

5.2 git-pw

The **git-pw** application can be used to integrate Git with Patchwork. The **git-pw** application relies on the REST API and can be used to interact to list, download and apply series, bundles and individual patches.

More information on **git-pw**, including installation and usage instructions, can be found in the documentation and the GitHub repo.

5.3 snowpatch

The **snowpatch** application is a bridge between Patchwork and the Jenkins continuous integration automation server. It monitors the REST API for incoming patches, applies them on top of an existing git tree, triggers appropriate builds and test suites, and reports the results back to Patchwork.

Find out more about **snowpatch** at its GitHub repo.

CHAPTER

SIX

INSTALLATION

This document describes the necessary steps to configure Patchwork in a production environment. This requires a significantly "harder" deployment than the one used for development. If you are interested in developing Patchwork, refer to the *development guide* instead.

This document describes a single-node installation of Patchwork, which will handle the database, server, and application. It is possible to split this into multiple servers, which would provide additional scalability and availability, but this is is out of scope for this document.

6.1 Deployment Guides, Provisioning Tools and Platform-as-a-Service

Before continuing, it's worth noting that Patchwork is a Django application. With the exception of the handling of incoming mail (described below), it can be deployed like any other Django application. This means there are tens, if not hundreds, of existing articles and blogs detailing how to deploy an application like this. As such, if any of the below information is unclear then we'd suggest you go search for "Django deployment guide" or similar, deploy your application, and submit a patch for this guide to clear up that confusion for others.

You'll also find that the same search reveals a significant number of existing deployment tools aimed at Django. These tools, be they written in Ansible, Puppet, Chef or something else entirely, can be used to avoid much of the manual configuration described below. If possible, embrace these tools to make your life easier.

Finally, many Platform-as-a-Service (PaaS) providers and tools support deployment of Django applications with minimal effort. Should you wish to avoid much of the manual configuration, we suggest you investigate the many options available to find one that best suits your requirements. The only issue here will likely be the handling of incoming mail - something which many of these providers don't support. We address this in the appropriate section below.

6.2 Requirements

For the purpose of this guide, we will assume an **Ubuntu 20.04** host: commands, package names and/or package versions will likely change if using a different distro or release. Similarly, usage of different package versions to the ones suggested may require slightly different configuration.

Before beginning, you should update and restart this system:

```
$ sudo apt-get update -y
$ sudo apt-get upgrade -y
$ sudo reboot
```

Once rebooted, we need to configure some environment variables. These will be used to ease deployment:

DATABASE_NAME=patchwork
: Name of the database. We'll name this after the application itself.

DATABASE_USER=www-data
: Username that the Patchwork web application will access the database with. We will use `www-data`, for reasons described later in this guide.

DATABASE_PASSWORD=
: Password that the Patchwork web application will access the database with. As we're going to use *peer* authentication (more on this later), this will be unset.

DATABASE_HOST=
: IP or hostname of the database host. As we're hosting the application on the same host as the database and hoping to use *peer* authentication, this will be unset.

DATABASE_PORT=
: Port of the database host. As we're hosting the application on the same host as the database and using the default configuration, this will be unset.

Export each of these. For example:

```
$ export DATABASE_NAME=patchwork
```

The remainder of the requirements are listed as we install and configure the various components required.

6.3 Database

6.3.1 Install Requirements

We're going to rely on PostgreSQL, though MySQL is also supported:

```
$ sudo apt-get install -y postgresql postgresql-contrib
```

6.3.2 Configure Database

We need to create a database for the system using the database name above. In addition, we need to add database users for two system users, the web user (the user that the web server runs as) and the mail user (the user that the mail server runs as). On Ubuntu these are `www-data` and `nobody`, respectively. PostgreSQL supports peer authentication, which uses the standard UNIX authentication method as a backend. This means no database-specific passwords need to be configured.

PostgreSQL created a system user called `postgres`; you will need to run commands as this user.

```
$ sudo -u postgres createdb $DATABASE_NAME
$ sudo -u postgres createuser $DATABASE_USER
$ sudo -u postgres createuser nobody
```

We will also need to apply permissions to the tables in this database but seeing as the tables haven't actually been created yet this will have to be done later.

Note: As noted in the Django documentation, Django expects databases to be configured with an encoding of UTF-8 or UTF-16. If using MySQL, you may need to configure this this explicitly as older versions defaulted to *latin1* encoding. Refer to the MySQL documentation for more information.

6.4 Patchwork

6.4.1 Install Requirements

The first requirement is Patchwork itself. It can be downloaded like so:

```
$ wget https://github.com/getpatchwork/patchwork/archive/v3.0.0.tar.gz
```

We will install this under `/opt`, though this is only a suggestion:

```
$ tar -xvzf v3.0.0.tar.gz
$ sudo mv patchwork-3.0.0 /opt/patchwork
```

Important: Per the Django documentation, source code should not be placed in your web server's document root as this risks the possibility that people may be able to view your code over the Web. This is a security risk.

Next we require Python. If not already installed, then you should do so now. Patchwork supports Python 3.7+. Python 3 is installed by default, but you should validate this now:

```
$ sudo apt-get install -y python3
```

We also need to install the various requirements. Let's use system packages for this also:

```
$ sudo apt-get install -y python3-django python3-psycopg2 \
    python3-djangorestframework python3-django-filters
```

Tip: The pkgs.org website provides a great reference for identifying the name of these dependencies.

You can also install requirements using *pip*. If using this method, you can install requirements like so:

```
$ sudo pip install -r /opt/patchwork/requirements-prod.txt
```

6.4.2 Configure Patchwork

You will also need to configure a settings file for Django. A sample settings file is provided that defines default settings for Patchwork. You'll need to configure settings for your own setup and save this as `production.py`.

```
$ cd /opt/patchwork
$ cp patchwork/settings/production{.example,}.py
```

Alternatively, you can override the `DJANGO_SETTINGS_MODULE` environment variable and provide a completely custom settings file.

The provided `production.example.py` settings file is configured to read configuration from environment variables. This suits container-based deployments quite well but for the all-in-one deployment we're configuring here, hardcoded settings makes more sense. If you wish to use environment variables, you should export each setting using the appropriate name, such as `DJANGO_SECRET_KEY`, `DATABASE_NAME` or `EMAIL_HOST`, instead of modifying the `production.py` file as we've done below.

Databases

We already defined most of the configuration necessary for this in the intro. As a reminder, these were:

- `DATABASE_NAME`
- `DATABASE_USER`
- `DATABASE_PASSWORD`
- `DATABASE_HOST`
- `DATABASE_PORT`

Export these environment variables or configure the `DATABASE` setting in `production.py` accordingly.

Static Files

While we have not yet configured our proxy server, we need to configure the location that these files will be stored in. We will install these under `/var/www/patchwork`, though this is only a suggestion and can be changed.

```
$ sudo mkdir -p /var/www/patchwork
```

Export the `STATIC_ROOT` environment variable or configure the `STATIC_ROOT` setting in `production.py`.

```
STATIC_ROOT = '/var/www/patchwork'
```

Secret Key

The `SECRET_KEY` setting is necessary for Django to generate signed data. This should be a random value and kept secret. You can generate and a value for `SECRET_KEY` with the following Python code:

```
import string
import secrets

chars = string.ascii_letters + string.digits + string.punctuation
print("".join([secrets.choice(chars) for i in range(50)]))
```

Export the `DJANGO_STATIC_KEY` environment variable or configure the `STATIC_KEY` setting in `production.py`.

Other Options

There are many other settings that may be configured, many of which are described in *Configuration*.

- `ADMINS`
- `TIME_ZONE`
- `LANGUAGE_CODE`
- `DEFAULT_FROM_EMAIL`
- `NOTIFICATION_FROM_EMAIL`

These are not configurable using environment variables and must be configured directly in the `production.py` settings file instead. For example, if you wish to enable the XML-RPC API, you should add the following:

```
ENABLE_XMLRPC = True
```

Similarly, should you wish to disable the REST API, you should add the following:

```
ENABLE_REST_API = False
```

For more information, refer to *Configuration*.

6.4.3 Final Steps

Once done, we should be able to check that all requirements are met using the `check` command of the `manage.py` executable. This must be run as the `www-data` user:

```
$ sudo -u www-data python3 manage.py check
```

Note: If you've been using environment variables to configure your deployment, you must pass the `--preserve-env` option for each attribute or pass the environments as part of the command:

```
$ sudo -u www-data \
    --preserve-env=DATABASE_NAME \
    --preserve-env=DATABASE_USER \
    --preserve-env=DATABASE_PASSWORD \
    --preserve-env=DATABASE_HOST \
    --preserve-env=DATABASE_PORT \
    --preserve-env=STATIC_ROOT \
    --preserve-env=DJANGO_SECRET_KEY \
python3 manage.py check
```

We should also take this opportunity to both configure the database and static files:

```
$ sudo -u www-data python3 manage.py migrate
$ sudo python3 manage.py collectstatic
$ sudo -u www-data python3 manage.py loaddata default_tags default_states
```

Note: The above `default_tags` and `default_states` fixtures above are just that: defaults. You can modify these to fit your own requirements.

Finally, it may be helpful to start the development server quickly to ensure you can see *something*. For this to function, you will need to add the `ALLOWED_HOSTS` and `DEBUG` settings to the `production.py` settings file:

```
ALLOWED_HOSTS = ['*']
DEBUG = True
```

Now, run the server.

```
$ sudo -u www-data python3 manage.py runserver 0.0.0.0:8000
```

Browse this instance at `http://[your_server_ip]:8000`. If everything is working, kill the development server using `Control-c` and remove `ALLOWED_HOSTS` and `DEBUG`.

6.5 Reverse Proxy and WSGI HTTP Servers

6.5.1 Install Packages

We will use *nginx* and *uWSGI* to deploy Patchwork, acting as reverse proxy server and WSGI HTTP server respectively. Other options are available, such as *Apache* with the *mod_wsgi* module, or *nginx* with the *Gunicorn* WSGI HTTP server. While we don't document these, sample configuration files for the former case are provided in `lib/apache2/`.

Let's start by installing *nginx* and *uWSGI*:

```
$ sudo apt-get install -y nginx-full uwsgi uwsgi-plugin-python3
```

6.5.2 Configure nginx and uWSGI

Configuration files for *nginx* and *uWSGI* are provided in the `lib` subdirectory of the Patchwork source code. These can be modified as necessary, but for now we will simply copy them.

First, let's load the provided configuration for *nginx* and disable the default configuration:

```
$ sudo cp /opt/patchwork/lib/nginx/patchwork.conf \
    /etc/nginx/sites-available/
$ sudo unlink /etc/nginx/sites-enabled/default
```

If you wish to modify this configuration, now is the time to do so. Once done, validate and enable your configuration:

```
$ sudo ln -s /etc/nginx/sites-available/patchwork.conf \
    /etc/nginx/sites-enabled/patchwork.conf
$ sudo nginx -t
```

Now, use the provided configuration for *uWSGI*:

```
$ sudo mkdir -p /etc/uwsgi/sites
$ sudo cp /opt/patchwork/lib/uwsgi/patchwork.ini \
    /etc/uwsgi/sites/patchwork.ini
```

Note: We created the `/etc/uwsgi` directory above because we're going to run *uWSGI* in emperor mode. This has benefits for multi-app deployments.

Note: If you're using environment variables for configuration, you will need to edit the `patchwork.ini` file created above to include these using the `env = VAR=VALUE` syntax.

6.5.3 Configure Patchwork

For security reasons, Django requires you to configure the `ALLOWED_HOSTS` setting, which is a "list of strings representing the host/domain names that this Django site can serve". To do this, configure the setting in the `production.py` setting file using the hostname(s) and/or IP address(es) from which you will be serving this domain. For example:

```
ALLOWED_HOSTS = ('.example.com', )
```

6.5.4 Create systemd Unit File

As things stand, *uWSGI* will need to be started manually every time the system boots, in addition to any time it may fail. We can automate this process using *systemd*. To this end a systemd unit file should be created to start *uWSGI* at boot:

```
$ sudo tee /etc/systemd/system/uwsgi.service > /dev/null << EOF
[Unit]
Description=uWSGI Emperor service

[Service]
ExecStartPre=/bin/bash -c 'mkdir -p /run/uwsgi; chown www-data:www-data /run/uwsgi'
ExecStart=/usr/bin/uwsgi --emperor /etc/uwsgi/sites
Restart=always
KillSignal=SIGQUIT
Type=notify
NotifyAccess=all

[Install]
WantedBy=multi-user.target
EOF
```

You should also delete the default service file found in `/etc/init.d` to ensure the unit file defined above is used.

```
sudo rm /etc/init.d/uwsgi
sudo systemctl daemon-reload
```

6.5.5 Final Steps

Start the *uWSGI* service we created above:

```
$ sudo systemctl restart uwsgi
$ sudo systemctl status uwsgi
$ sudo systemctl enable uwsgi
```

Next up, restart the *nginx* service:

```
$ sudo systemctl restart nginx
$ sudo systemctl status nginx
$ sudo systemctl enable nginx
```

Finally, browse to the instance using your browser of choice. You may wish to take this opportunity to setup your projects and configure your website address (in the Sites section of the admin console, found at `/admin`).

If there are issues with the instance, you can check the logs for *nginx* and *uWSGI*. There are a couple of commands listed below which can help:

- `sudo systemctl status uwsgi`, `sudo systemctl status nginx`

  To ensure the services have correctly started

- `sudo cat /var/log/nginx/error.log`

  To check for issues with *nginx*

- `sudo cat /var/log/patchwork.log`

  To check for issues with *uWSGI*. This is the default log location set by the `daemonize` setting in the *uWSGI* configuration file.

6.6 Django administrative console

In order to access the administrative console at `/admin`, you need at least one user account to be registered and configured as a super user or staff account to access the Django administrative console. This can be achieved by doing the following:

```
$ python3 manage.py createsuperuser
```

Once the administrative console is accessible, you would want to configure your different sites and their corresponding domain names, which is required for the different emails sent by Patchwork (registration, password recovery) as well as the sample `pwclientrc` files provided by your project's page.

6.7 Incoming Email

Patchwork is designed to parse incoming mails which means you need an address to receive email at. This is a problem that has been solved for many web apps, thus there are many ways to go about this. Some of these ways are discussed below.

6.7.1 IMAP/POP3

The easiest option for getting mail into Patchwork is to use an existing email address in combination with a mail retriever like getmail, which will download mails from your inbox and pass them to Patchwork for processing. *getmail* is easy to set up and configure: to begin, you need to install it:

```
$ sudo apt-get install -y getmail
```

Once installed, you should configure it, substituting your own configuration details where required below:

```
$ sudo tee /etc/getmail/use@example.com/getmailrc > /dev/null << EOF
[retriever]
type = SimpleIMAPSSLRetriever
server = imap.example.com
port = 993
username = XXX
password = XXX
mailboxes = ALL
```

(continues on next page)

(continued from previous page)

```
[destination]
# we configure Patchwork as a "mail delivery agent", in that it will
# handle our mails
type = MDA_external
path = /opt/patchwork/patchwork/bin/parsemail.sh

[options]
# retrieve only new emails
read_all = false
# do not add a Delivered-To: header field
delivered_to = false
# do not add a Received: header field
received = false
EOF
```

Validate that this works as expected by starting *getmail*:

```
$ getmail --getmaildir=/etc/getmail/user@example.com --idle INBOX
```

If everything works as expected, you can create a *systemd* script to ensure this starts on boot:

```
$ sudo tee /etc/systemd/system/getmail.service > /dev/null << EOF
[Unit]
Description=Getmail for user@example.com

[Service]
User=nobody
ExecStart=/usr/bin/getmail --getmaildir=/etc/getmail/user@example.com --idle INBOX
Restart=always

[Install]
WantedBy=multi-user.target
EOF
```

And start the service:

```
$ sudo systemctl start getmail
$ sudo systemctl status getmail
$ sudo systemctl enable getmail
```

6.7.2 Mail Transfer Agent (MTA)

The most flexible option is to configure our own mail transfer agent (MTA) or "email server". There are many options, of which Postfix is one. While we don't cover setting up Postfix here (it's complicated and there are many guides already available), Patchwork does include a script to take received mails and create the relevant entries in Patchwork for you. To use this, you should configure your system to forward all emails to a given localpart (the bit before the `@`) to this script. Using the `patchwork` localpart (e.g. `patchwork@example.com`) you can do this like so:

```
$ sudo tee -a /etc/aliases > /dev/null << EOF
patchwork: "|/opt/patchwork/patchwork/bin/parsemail.sh"
EOF
```

You should ensure the appropriate user is created in PostgreSQL and that it has (minimal) access to the database. Patchwork provides scripts for the latter and they can be loaded as seen below:

```
$ sudo -u postgres psql -f \
    /opt/patchwork/lib/sql/grant-all.postgres.sql patchwork
```

Note: This assumes that you are using the aliases(5) file that is owned by root, and that Postfix's `default_privs` configuration is set as `nobody`. If this is not the case, you should change both the username in the `createuser` command above and substitute the username in the `grant-all.postgres.sql` script with the appropriate alternative.

6.7.3 Use a Email-as-a-Service Provider

Setting up an email server can be a difficult task and, in the case of deployment on PaaS provider, may not even be an option. In this case, there are a variety of web services available that offer "Email-as-as-Service". These services typically convert received emails into HTTP POST requests to your endpoint of choice, allowing you to sidestep configuration issues. We don't cover this here, but a simple wrapper script coupled with one of these services can be more than to get email into Patchwork.

You can also create such as service yourself using a PaaS provider that supports incoming mail and writing a little web app.

6.8 (Optional) Configure your VCS to Automatically Update Patches

The `tools` directory of the Patchwork distribution contains a file named `post-receive.hook` which is a sample Git hook that can be used to automatically update patches to the *Accepted* state when corresponding commits are pushed via Git.

To install this hook, simply copy it to the `.git/hooks` directory on your server, name it `post-receive`, and make it executable.

This sample hook has support to update patches to different states depending on which branch is being pushed to. See the `STATE_MAP` setting in that file.

If you are using a system other than Git, you can likely write a similar hook using the APIs or *API clients* to to update patch state. If you do write one, please contribute it.

6.9 (Optional) Configure the Patchwork Cron Job

Patchwork can send notifications of patch changes. Patchwork uses a cron management command - `manage.py cron` - to send these notifications and to clean up expired registrations. To enable this functionality, add the following to your crontab:

```
# m h  dom mon dow   command
*/10 * * * * cd patchwork; python3 ./manage.py cron
```

Note: The frequency should be the same as the `NOTIFICATION_DELAY_MINUTES` setting, which defaults to 10 minutes. Refer to the *configuration guide* for more information.

CHAPTER

SEVEN

CONFIGURATION

This document describes the various configuration options available in Patchwork. These options can be used for both *development* and *deployment* installations.

7.1 The `settings.py` File

Patchwork is a Django application and, as such, relies on Python-based settings files. Refer to the Django documentation for more information on the general format.

Patchwork provides three settings files:

`base.py`
: A base settings file that should not be used directly.

`dev.py`
: A settings file for development use. **This file is horribly insecure and must not be used in production**.

`production.example.py`
: A sample settings file for production use. This will likely require some heavy customization. The *deployment guide* provides more information.

7.2 Patchwork-specific Settings

Patchwork utilizes a number of Patchwork-only settings in addition to the Django and Django REST Framework settings.

7.2.1 `ADMINS_HIDE`

If True, the details in ADMINS will be hidden from the *About* page (`/about`).

New in version 2.2.

7.2.2 COMPAT_REDIR

Enable redirections of URLs from previous versions of Patchwork.

7.2.3 CONFIRMATION_VALIDITY_DAYS

The number of days to consider an account confirmation request valid. After this interval, the *cron management command* will delete the request.

7.2.4 DEFAULT_ITEMS_PER_PAGE

The default number of items to display in the list pages for a project (`/project/{projectID}/list`) or bundle (`/bundle/{userID}/{bundleName}`).

This is customizable on a per-user basis from the user configuration page.

Changed in version 2.0: This option was previously named `DEFAULT_PATCHES_PER_PAGE`. It was renamed as cover letters are now supported also.

7.2.5 ENABLE_REST_API

Enable the *REST API*.

New in version 2.0.

7.2.6 ENABLE_XMLRPC

Enable the *XML-RPC API*.

7.2.7 FORCE_HTTPS_LINKS

Force use of `https://` links instead of guessing the scheme based on current access. This is useful if SSL protocol is terminated upstream of the server (e.g. at the load balancer)

7.2.8 MAX_REST_RESULTS_PER_PAGE

The maximum number of items that can be requested in a REST API request using the `per_page` parameter.

New in version 2.2.

7.2.9 NOTIFICATION_DELAY_MINUTES

The number of minutes to wait before sending any notifications to a user. An notification generated during this time are gathered into a single digest email, ensuring users are not spammed with emails from Patchwork.

7.2.10 NOTIFICATION_FROM_EMAIL

The email address that notification emails should be sent from.

7.2.11 REST_RESULTS_PER_PAGE

The number of items to include in REST API responses by default. This can be overridden by the `per_page` parameter for some endpoints.

New in version 2.0.

CHAPTER

EIGHT

MANAGEMENT

This document describes the myriad administrative commands available with Patchwork. Many of these commands are referenced in the *development* and *deployment* installation guides.

8.1 The `manage.py` Script

Django provides the `django-admin` command-line utility for interacting with Django applications and projects, as described in the Django documentation. Patchwork, being a Django application, provides a wrapper for this command - `manage.py` - that exposes not only the management commands of Django and its default applications, but also a number of custom, Patchwork-only management commands.

An overview of the Patchwork-specific commands is provided below. For information on the commands provided by Django itself, refer to the Django documentation. Information on any command can also be found by passing the `--help` parameter:

```
./manage.py cron --help
```

8.2 Available Commands

8.2.1 cron

```
./manage.py cron
```

Run periodic Patchwork functions: send notifications and expire unused users.

This is required to ensure notifications emails are actually sent to users that request them and is helpful to expire unused users created by spambots. For more information on integration of this script, refer to the *deployment installation guide*.

8.2.2 dumparchive

Export Patchwork projects as tarball of mbox files.

```
./manage.py dumparchive [-c | --compress] [PROJECT [PROJECT...]]
```

This is mostly useful for exporting the patch dataset of a Patchwork project for use with other programs.

`-c, --compress`

compress generated archive.

PROJECT
list ID of project(s) to export. Export all projects if none specified.

8.2.3 parsearchive

Parse an mbox archive file and store any patches/comments found.

```
./manage.py parsearchive [--list-id <list-id>] <infile>
```

This is mostly useful for development or for adding message that were missed due to, for example, an outage.

--list-id <list-id>
mailing list ID. If not supplied, this will be extracted from the mail headers.

infile
input mbox filename

8.2.4 parsemail

Parse an mbox file and store any patch/comment found.

```
./manage.py parsemail [--list-id <list-id>] <infile>
```

This is the main script used to get mails (and therefore patches) into Patchwork. It is generally used by the `parsemail.sh` script in combination with a mail transfer agent (MTA) like Postfix. For more information, refer to the *deployment installation guide*.

--list-id <list-id>
mailing list ID. If not supplied, this will be extracted from the mail headers.

infile
input mbox filename. If not supplied, a patch will be read from `stdin`.

8.2.5 replacerelations

Parse a patch groups file and store any relation found

```
./manage.py replacerelations <infile>
```

This is a script used to ingest relations into Patchwork.

A patch groups file contains on each line a list of space separated patch IDs of patches that form a relation.

For example, consider the contents of a sample patch groups file:

```
1 3 5
2
7 10 11 12
```

In this case the script will identify 2 relations, (1, 3, 5) and (7, 10, 11, 12). The single patch ID "2" on the second line is ignored as a relation always consists of more than 1 patch.

Further, if a patch ID in the patch groups file does not exist in the database of the Patchwork instance, that patch ID will be silently ignored while forming the relations.

Running this script will remove all existing relations and replace them with the relations found in the file.

infile

input patch groups file.

8.2.6 rehash

Update the hashes on existing patches.

```
./manage.py rehash [<patch_id>, ...]
```

Patchwork stores hashes for each patch it receives. These hashes can be used to uniquely identify a patch for things like *automatically changing the state of the patch in Patchwork when it merges*. If you change your hashing algorithm, you may wish to rehash the patches.

patch_id

a patch ID number. If not supplied, all patches will be updated.

8.2.7 retag

Update the tag (Ack/Review/Test) counts on existing patches.

```
./manage.py retag [<patch_id>...]
```

Patchwork extracts *tags* from each patch it receives. By default, three tags are extracted, but it's possible to change this on a per-instance basis. Should you add additional tags, you may wish to scan older patches for these new tags.

patch_id

a patch ID number. If not supplied, all patches will be updated.

CHAPTER

NINE

UPGRADING

This document provides some general tips and tricks that one can use when upgrading an existing, production installation of Patchwork. If you are interested in the specific changes between each release, refer to /releases/index instead. If this is your first time installing Patchwork, refer to the *Installation* instead.

9.1 Before You Start

Before doing anything, always **backup your data**. This generally means backing up your database, but it might also be a good idea to backup your environment in case you encounter issues during the upgrade process.

While Patchwork won't explicitly prevent it, it's generally wise to avoid upgrades spanning multiple releases in one go. An iterative upgrade approach will provide an easier, if slower, upgrade process.

9.2 Identify Changed Scripts, Requirements, etc.

/releases/index provides a comprehensive listing of all backwards-incompatible changes that occur between releases of Patchwork. Examples of such changes include:

- Moved/removed scripts and files
- Changes to the requirements, e.g. supported Django versions
- Changes to API that may affect, for example, third-party tools

It is important that you understand these changes and ensure any scripts you may have, such as systemd scripts, are modified accordingly.

9.3 Understand What Requirements Have Changed

New versions of Patchwork can often require additional or updated version of dependencies, e.g. newer versions of Django. It is important that you understand these requirements and can fulfil them. This is particularly true for users relying on distro-provided packages, who may have to deal with older versions of a package or may be missing a package altogether (though we try to avoid this). Such changes are usually listed in the /releases/index, but you can also diff the *requirements.txt* files in each release for comparison.

9.4 Collect Static Files

New versions of Patchwork generally contain changes to the additional files like images, CSS and JavaScript. To do this, run the *collectstatic* management commands:

```
$ ./manage.py collectstatic
```

9.5 Upgrade Your Database

New versions of Patchwork may provide a number of schema and/or data migrations which must be applied before starting the instance. To do this, run the *migrate* management command:

```
$ ./manage.py migrate
```

For more information on migrations, refer to the Django documentation.

CHAPTER

TEN

CONTRIBUTING

10.1 Coding Standards

Follow PEP8. All code is currently PEP 8 compliant and it should stay this way.

All code must be licensed using GPL v2.0 or later and must have a SPDX License Identifier stating this. A copyright line should be included on new files and may be added for significant changes to existing files.

```
# Patchwork - automated patch tracking system
# Copyright (C) 2000 Jane Doe <jane.doe@example.com>
# Copyright (C) 2001 Joe Bloggs <joebloggs@example.com>
#
# SPDX-License-Identifier: GPL-2.0-or-later
```

Changes that fix semantic issues will be happily received, but please keep such changes separate from functional changes.

Patchwork uses the pre-commit framework to allow automated style checks when committing code. This is opt-in but avoids the need to manually run style checks on commits. Pre-commit can be installed and enabled like so:

```
$ pip install --user pre-commit
$ pre-commit install --allow-missing-config
```

Once installed, the various checks listed in `.pre-commit-config.yaml` will be run on changed files when committing. It is also possible to run the checks on all files manually:

```
$ pre-commit run --all-files
```

In addition to *pre-commit*, we provide *tox* targets for style checks. These are used by CI and can be useful if checking all files manually. Refer to the *Testing* section below for more information on usage of this tool.

10.2 Testing

Patchwork includes a tox script to automate testing. This requires a functional database and some Python requirements like *tox*. Refer to *Installation* for information on how to configure these.

You may also need to install *tox*. If so, do this now:

```
$ pip install --user tox
```

Tip: If you're using Docker, you may not need to install *tox* locally. Instead, it will already be installed inside the container. For Docker, you can run *tox* like so:

```
$ docker-compose run --rm web tox [ARGS...]
```

For more information, refer to *Docker-Based Installation*.

Assuming these requirements are met, actually testing Patchwork is quite easy to do. To start, you can show the default targets like so:

```
$ tox -l
```

You'll see that this includes a number of targets to run unit tests against the different versions of Django supported, along with some other targets related to code coverage and code quality. To run one of these, use the `-e` parameter:

```
$ tox -e py27-django18
```

In the case of the unit tests targets, you can also run specific tests by passing the fully qualified test name as an additional argument to this command:

```
$ tox -e py27-django18 patchwork.tests.SubjectCleanUpTest
```

Because Patchwork support multiple versions of Django, it's very important that you test against all supported versions. When run without argument, tox will do this:

```
$ tox
```

10.3 Release Notes

Patchwork uses reno for release note management. To use *reno*, you must first install it:

```
$ pip install --user reno
```

Once installed, a new release note can be created using the `reno new` command:

```
$ reno new <slugified-summary-of-change>
```

Modify the created file, removing any irrelevant sections, and include the modified file in your change.

10.4 API

As discussed in *Release Process*, the API is versioned differently from Patchwork itself. Should you make changes to the API, you need to ensure these only affect newer versions of the API. Refer to previous changes in the `patchwork/api` directory and to the Django REST Framework documentation for more information.

Important: All API changes should be called out in *release notes* using the `api` section.

10.5 Reporting Issues

You can report issues to the *mailing list* or the GitHub issue tracker.

10.6 Submitting Changes

All patches should be sent to the *mailing list*. You must be subscribed to the list in order to submit patches. Please abide by the QEMU guidelines on contributing or submitting patches. This covers both the initial submission and any follow up to the patches. In particular, ensure:

- *All tests pass*
- Documentation has been updated with new requirements, new script names etc.
- *A release note is included*

Patches should ideally be submitted using the *git send-email* tool.

10.7 Mailing Lists

Patchwork uses a single mailing list for development, questions and announcements.

patchwork@lists.ozlabs.org

Further information about the Patchwork mailing list is available can be found on lists.ozlabs.org.

CHAPTER

ELEVEN

INSTALLATION

This document describes the necessary steps to configure Patchwork in a development environment. If you are interested in deploying Patchwork in a production environment, refer to *the deployment guide* instead.

To begin, you should clone Patchwork:

```
$ git clone git://github.com/getpatchwork/patchwork.git
```

11.1 Docker-Based Installation

Patchwork provides a Docker-based environment for quick configuration of a development environment. This is the preferred installation method. To configure Patchwork using Docker:

1. Install docker and docker-compose.[1] Patchwork assumes that you have Docker configured to allow a non-root user to manage Docker, as outlined in the Docker post-install instructions.

1. Create a `.env` file in the root directory of the project and store your `UID` and `GID` attribute there.

   ```
   $ echo "UID=$UID" > .env
   $ echo "GID=`id -g`" >> .env
   ```

2. Build the images. This will download over 200MB from the internet:

   ```
   $ docker-compose build
   ```

   To use Postgres instead of MySQL, give the `-f docker-compose-pg.yml` argument to `docker-compose`. This is required on non-x86 architectures as the MySQL Docker images do not have multiarch support.

3. Run `docker-compose up`:

   ```
   $ docker-compose up
   ```

   This will be visible at http://localhost:8000/.

To run a shell within this environment, run:

```
$ docker-compose run --rm web /bin/bash
```

To run `django-manage` commands, such as `createsuperuser` or `migrate`, run:

[1] Depending on your distro, *docker-compose* may also be available as a package.

```
$ docker-compose run --rm web python manage.py createsuperuser
```

To access the SQL command-line client, run:

```
$ docker-compose run --rm web python manage.py dbshell
```

To backup the database, run:

```
$ docker-compose run --rm web python manage.py dbbackup
```

Likewise, to restore an older version of the database, run:

```
$ docker-compose run --rm web python manage.py dbrestore
```

To run unit tests against the system Python packages, run:

```
# For MySQL database:
$ docker-compose exec -T -- db sh -c \
      "exec mysql -uroot -p\"\${MYSQL_ROOT_PASSWORD}\" -e \"GRANT ALL ON \\\`test\\_\$
↪{MYSQL_DATABASE}%\\\`.* to '\${MYSQL_USER}'@'%'; FLUSH PRIVILEGES;\""

$ docker-compose run --rm web python manage.py test
```

To run unit tests for multiple versions using `tox`, run:

```
$ docker-compose run --rm web tox
```

To reset the database, stop the db container and purge the database files:

```
$ docker-compose stop db
$ sudo rm -rf tools/docker/db
```

Any local edits to the project files made locally are immediately visible to the Docker container, and so should be picked up by the Django auto-reloader.

For more information on Docker itself, please refer to the docker and docker-compose documentation.

Note: If using SELinux, you will need to create a custom SELinux rule to allow the Docker process to access your working directory. Run:

```
$ chcon -R -t svirt_sandbox_file_t $PATCHWORK_DIR
```

where `$PATCHWORK_DIR` is the absolute patch to the `patchwork` folder created when you cloned the repo. For more information, see `man docker run`.

Note: If you see an error like the below:

```
ERROR: Couldn't connect to the Docker daemon at http+docker://localunixsocket - is it␣
↪running?
```

ensure you have correctly installed Docker, and have followed the Docker post-install instructions.

Note: If you see an error like the below:

```
You must define UID in .env
```

Ensure you have created a `.env` file in the root of your project directory and stored the `UID` attribute there. For more information on why this is necessary, refer to this docker-compose issue.

11.2 Manual Installation

Manual installation can be used where use of Docker is not possible or desired.

11.2.1 Install Required Packages

There are a number of different requirements for developing Patchwork:

- Python and libraries
- A supported database (RDBMS)

These are detailed below.

Python Requirements

To develop Python-based software you first need Python. Patchwork supports Python 3.7+. Python 3 will be installed by default on many installations, though a suitable version can usually be installed manually using the `python3` package.

It's a good idea to use virtual environments to develop Python software. Virtual environments are "instances" of your system Python without any of the additional Python packages installed. They are useful to develop and possibly deploy Patchwork against a "well known" set of dependencies, but they can also be used to test Patchwork against several versions of Django.

If you do not have `virtualenv` installed then you should install it now. This can be installed using the `python3-virtualenv` package. Alternatively you can install these using `pip`.

It is also helpful to install `tox` which is used for running tests in Patchwork. This can be installed using the `python3-tox` package, or via `pip`.

Database Requirements

If not already installed, you may need to install an RDBMS. You can use either MariaDB/MySQL or PostgreSQL for this purpose. You should also install the development headers, known as `libmysqlclient-dev` or `libpq-dev` respectively on Debian-based Debian-based distros like Ubuntu and `mysql-devel` or `postgresql-devel` on RHEL-based distros.

Note: While Django provides support for multiple database backends, Patchwork itself is only tested against MySQL/MariaDB and PostgreSQL. Should you wish to use a different backend, ensure you validate this first (and perhaps upstream any changes you may find necessary).

Note: You may be tempted to use SQLite to develop Patchwork. We'd advise against doing this. SQLite supports a subset of the functionality of "full" RDBMS like MySQL: for example, case-sensitive matching of Unicode is not supported. You will find some tests provided by Patchwork fail and some patches you develop may fail in production due to these differences.

Example Installation

An example for installing all these packages and the MySQL RDBMS on Ubuntu 20.04 is given below:

```
$ sudo apt-get install python3 python3-pip python3-dev python3-virtualenv \
    python3-tox mysql-server libmysqlclient-dev
```

If you have an existing MariaDB/MySQL installation then you can install all packages using `pip`:

```
$ sudo pip install virtualenv tox
```

11.2.2 Configure Virtual Environment

Note: If you are interested in simply *testing Patchwork*, many of the below steps are not required. tox will automatically install dependencies and use virtual environments when testing.

Once these requirements are installed, you should create and activate a new virtual environment. This can be done like so:

```
$ virtualenv .venv
$ source .venv/bin/activate
(.venv)$
```

Note: If you wish to use a specific Python version, you can provide the `--python` argument to use this, e.g. `--python=python3.7`.

Now install the packages. Patchwork provides three requirements files.

requirements-dev.txt
: Packages required to configure a development environment

requirements-prod.txt
: Packages required for deploying Patchwork in production

requirements-test.txt
: Packages required to run tests

We're going to install the first of these, which can be done like so:

```
(.venv)$ cd patchwork
(.venv)$ pip install -r requirements-dev.txt
```

Note: Once configured this does not need to be done again *unless* the requirements change, e.g. Patchwork requires an updated version of Django.

11.2.3 Initialize the Database

One installed, the database must be configured. We will assume you have root access to the database for these steps.

To begin, export your database credentials as follows:

```
(.venv)$ db_user=root
(.venv)$ db_pass=password
```

Now, create the database. If this is your first time configuring the database, you must create a `patchwork` user (or similar) along with the database instance itself. The commands below will do this, dropping existing databases if necessary:

```
(.venv)$ mysql -u$db_user -p$db_pass << EOF
DROP DATABASE IF EXISTS patchwork;
CREATE DATABASE patchwork CHARACTER SET utf8;
GRANT ALL PRIVILEGES ON patchwork.* TO 'patchwork'@'localhost'
    IDENTIFIED BY 'password';
EOF
```

Note: The `patchwork` username and `password` password are the defaults expected by the provided `dev` settings files. If using something different, export the `DATABASE_USER` and `DATABASE_PASSWORD` variables described in the *Environment Variables* section below. Alternatively, you can create your own settings file with these variables hardcoded and change the value of `DJANGO_SETTINGS_MODULE` as described below.

11.2.4 Load Initial Data

Before continuing, we need to tell Django where it can find our configuration. Patchwork provides a default development `settings.py` file for this purpose. To use this, export the `DJANGO_SETTINGS_MODULE` environment variable as described below:

```
(.venv)$ export DJANGO_SETTINGS_MODULE=patchwork.settings.dev
```

Alternatively you can provide your own `settings.py` file and provide the path to that instead.

Once done, we need to create the tables in the database. This can be done using the `migrate` command of the `manage.py` executable:

```
(.venv)$ ./manage.py migrate
```

Next, you should load the initial fixtures into Patchwork. These initial fixtures provide.

`default_tags.xml`
: The tags that Patchwork will extract from mails. For example: `Acked-By`, `Reviewed-By`

`default_states.xml`
: The states that a patch can be in. For example: `Accepted`, `Rejected`

default_projects.xml
: A default project that you can then upload patches for

These can be loaded using the `loaddata` command:

```
(.venv)$ ./manage.py loaddata default_tags default_states default_projects
```

You should also take the opportunity to create a "superuser". You can do this using the aptly-named `createsuperuser` command:

```
(.venv)$ ./manage.py createsuperuser
```

11.3 Import Mailing List Archives

Regardless of your installation method of choice, you will probably want to load some real emails into the system. This can be done manually, however it's generally much easier to download an archive from a Mailman instance and load these using the `parsearchive` command. You can do this like so:

```
(.venv)$ mm_user=<myusername>
(.venv)$ mm_pass=<mypassword>
(.venv)$ mm_host=https://lists.ozlabs.org
(.venv)$ mm_url=$mm_host/private/patchwork.mbox/patchwork.mbox
(.venv)$ curl -F username=$mm_user -F password=$mm_pass -k -O $mm_url
```

where `mm_user` and `mm_pass` are the username and password you have registered with on the Mailman instance found at `mm_host`.

Note: We provide instructions for downloading archives from the Patchwork mailing list, but almost any instance of Mailman will allow downloading of archives as seen above; simply change the `pw_url` variable defined. You can find more informations about this here.

Load these archives into Patchwork. Depending on the size of the downloaded archives this may take some time:

```
(.venv)$ ./manage.py parsearchive patchwork.mbox
```

Finally, run the server and browse to the IP address of your board using your browser of choice:

```
(.venv)$ ./manage.py runserver 0.0.0.0:8000
```

Once finished, you can kill the server (`Ctrl+C`) and exit the virtual environment:

```
(.venv)$ deactivate
$
```

Should you wish to re-enter this environment, simply source the `activate` script again.

11.4 Django Debug Toolbar

Patchwork installs and enables the 'Django Debug Toolbar' application by default when using development settings and requirements. This provides a configurable set of panels that display various debug information about the current request/response and, when clicked, display more details about the panel's content.

Important: By default, the toolbar is only displayed if you are developing on `localhost`. If developing on a different machine, you should configure an SSH tunnel such that, for example, `localhost:8000` points to `[DEV_MACHINE_IP]:8000`.

For more information, refer to the documentation.

11.5 Django Database Backup

Patchwork installs and enables the 'Django Database Backup' application by default when using development settings and requirements. This provides the following management commands, which can be useful for hacking on Patchwork:

- `dbbackup`
- `dbrestore`
- `mediabackup`
- `mediarestore`

For more information, refer to the documentation.

11.6 Environment Variables

The following environment variables are available to configure settings.

`DATABASE_NAME=patchwork`
: Name of the database

`DATABASE_USER=patchwork`
: Username to access the database with

`DATABASE_PASSWORD=password`
: Password to access the database with<

`DATABASE_TYPE=mysql`
: Type of database to use. Options: `mysql`, `postgres`

CHAPTER

TWELVE

RELEASE PROCESS

12.1 Versioning

There are two types of versioning in play in Patchwork: the version for Patchwork itself (i.e. the code or *core*) and the version for the *REST API <../api/rest>*.

12.1.1 Patchwork Code

Since version 1.0, Patchwork has implemented a version of Semantic Versioning . To summarise, releases take the format **MAJOR.MINOR.PATCH** (or just **MAJOR.MINOR**). We increment:

1. **MAJOR** version when we make major UI changes or functionality updates
2. **MINOR** version when we make minor UI changes or functionality updates
3. **PATCH** version when we make make bug fixes, dependency updates etc.

In Git, each release will have a tag indicating the version number. In addition, each release series has it's own branch called *stable/MAJOR.MINOR* to allow backporting of bugfixes or security updates to older versions.

12.1.2 REST API

The REST API also uses a variant of *Semantic Versioning*. To summarise, API versions take the format **MAJOR.MINOR**. We increment:

1. **MAJOR** version when we make breaking changes to the API. This generally means removing an API or fields in an API.
2. **MINOR** version when we add functionality in a backwards-compatible manner. This generally means adding new fields and endpoint.

These version numbers are exposed via the API and it's possible to request a specific version in the URL. Refer to the *API Guide <../api/rest>* for more information.

12.2 Release Cycle

There is no cadence for releases: they are made available as necessary.

12.3 Supported Versions

Typically all development should occur on `main`. While we will backport bugfixes and security updates, we will not backport any new features. This is to ensure stability for users of these versions of Patchwork.

12.4 Release Checklist

The follow steps apply to all releases:

- Documentation has been updated with latest release version
- Documentation references latest supported version of Django
- 'alpha' tag has been removed from `__version__` in `patchwork/__init__.py`
- Commit has been tagged with an annotated tag. The tag should take the form *v[MAJOR].[MINOR].[PATCH]*, e.g. *v2.0.1*. The message should read:

  ```
  Version [MAJOR].[MINOR].[PATCH]
  ```

- A GitHub Release, with text corresponding to an abbreviated form of the release notes for that cycle, has been created
- An email describing the release and top-level overview of the changes has been sent to the mailing list. Refer to the emails for Patchwork v2.0.0 and Patchwork v2.0.1 for examples.

The following only apply to full releases, or those where the **MAJOR** or **MINOR** number is incremented:

- A new branch called `stable/MAJOR.MINOR` has been created from the tagged commit

Once released, bump the version found in `patchwork/__init__.py` once again.

12.5 Backporting

We will occasionally backport bugfixes and security updates. When backporting a patch, said patch should first be merged into `main`. Once merged, you can backport by cherry-picking commits, using the `-x` flag for posterity:

```
$ git cherry-pick -x <main_commit>
```

There may be some conflicts; resolve these, uncommenting the *Conflicts* line when committing:

```
Conflicts
        patchwork/bin/pwclient
```

When enough patches have been backported, you should release a new **PATCH** release.

12.5.1 Backport criteria

We consider bug fixes and security updates to the Patchwork code itself valid for backporting, along with fixes to documentation and developer tooling. We do not, however, consider the following backportable:

Features
: Backporting features is complicated and introduces instability in what is supposed to be stable release. If new features are required, users should update their Patchwork version.

API changes
: Except for bug fixes that resolve 5xx-class errors or fix security issues. This also applies to API versions.

Requirement changes
: Requirements on a stable branch are provided as a "snapshot in time" and, as with features, should not change so as to prevent instability being introduced in a stable branch. In addition, stable requirements are not a mechanism to alert users to security vulnerabilities and should not be considered as such. Users of stable branches should either rely on distro-provided dependencies, which generally maintain a snapshot-in-time fork of packages and selectively backport fixes to them, or manage dependencies manually. In cases, where using a distro-provided package necessitates minor changes to the Patchwork code, these can be discussed on a case-by-case basis.

CHAPTER

THIRTEEN

USING THE APIS

Patchwork provides two APIs: the legacy *XML-RPC API* and the *REST API*. You can use these APIs to interact with Patchwork programmatically and to develop your own clients.

For quick usage examples of the APIs, refer to the documentation. For examples of existing clients, refer to *Clients*.

CHAPTER

FOURTEEN

STATIC ASSETS

Patchwork relies on a number of third-party JavaScript libraries. These, along with their supporting assets and the Patchwork-only libraries and assets, are described below.

14.1 css

bootstrap.min.css
: CSS for the *Bootstrap* library.

  Refer to the *js* section below for more information on *Bootstrap*.

selectize.bootstrap3.css
: CSS for the *Selectize* library.

  Refer to the *js* section below for more information on *Selectize*.

style.css
: Custom, Patchwork styling. Mostly a collection of overrides for default Bootstrap styles.

  Part of Patchwork.

14.2 fonts

glyphicons-halflings-regular.*
: Library of precisely prepared monochromatic icons and symbols, created with an emphasis to simplicity and easy orientation. Provided as part of the Bootstrap library.

  These are in multiple formats to support different browsers/environments. Refer to the *js* section below for more information on Bootstrap.

14.3 js

bootstrap.js
: The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web.

  This is used for the main UI of Patchwork.

  Website
  : https://getbootstrap.com/

GitHub
https://github.com/twbs/bootstrap/

Version
3.2.0

`bundle.js`
Utility functions for bundle patch list manipulation (re-ordering patches, etc.)

Part of Patchwork.

`clipboard.min.js`
Modern copy to clipboard. No Flash. Just 3kb gzipped

This is used to allow us to "click to copy" various elements in the UI.

Website
https://clipboardjs.com/

GitHub
https://github.com/zenorocha/clipboard.js/

Version
1.7.1

`jquery.js`
jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

This is used across Patchwork, including by the likes of `bundle.js`, as well as by the various plugins below.

Website
https://jquery.com/

GitHub
https://github.com/jquery/jquery

Version
3.6.0

`jquery.checkboxes.js`
A jQuery plugin that gives you nice powers over your checkboxes.

This is used to allow shift-select of checkboxes on the patch list page.

Website
http://rmariuzzo.github.io/checkboxes.js

GitHub
https://github.com/rmariuzzo/checkboxes.js

Version
1.2.2

`jquery.stickytableheaders.js`
A jQuery plugin that makes large tables more usable by having the table header stick to the top of the screen when scrolling.

This is used to ensure the heads on the patch list page stay at the top as we scroll.

GitHub
https://github.com/jmosbech/StickyTableHeaders

Version
0.1.24

`jquery.tablednd.js`
jQuery plug-in to drag and drop rows in HTML tables.

This is used by the bundle patch list to allow us to control the order of the patches in said bundle.

Website
http://www.isocra.com/2008/02/table-drag-and-drop-jquery-plugin/

GitHub
https://github.com/isocra/TableDnD

Version
1.0.5

`js.cookie.min.js`
Library used to handle cookies.

This is used to get the `csrftoken` cookie for AJAX requests in JavaScript.

GitHub
https://github.com/js-cookie/js-cookie/

Version
3.0.0

`rest.js.`
Utility module for REST API requests to be used by other Patchwork JS files.

Part of Patchwork.

`selectize.min.js`
Selectize is the hybrid of a `textbox` and `<select>` box. It's jQuery based and it has autocomplete and native-feeling keyboard navigation; useful for tagging, contact lists, etc.

Website
https://selectize.github.io/selectize.js/

GitHub
https://github.com/selectize/selectize.js

Version
0.13.5

CHAPTER

FIFTEEN

THE REST API

Patchwork provides a REST API. This API can be used to retrieve and modify information about patches, projects and more.

This guide provides an overview of how one can interact with the REST API. For detailed information on type and response format of the various resources exposed by the API, refer to the web browsable API. This can be found at:

> https://patchwork.example.com/api/1.3/

where *patchwork.example.com* refers to the URL of your Patchwork instance.

If all you want is reference guides, skip straight to *Schemas*.

Important: The REST API can be enabled/disabled by the administrator: it may not be available in every instance. Refer to `/about` on your given instance for the status of the API, e.g.

> https://patchwork.ozlabs.org/about

New in version 2.0: The REST API was introduced in Patchwork v2.0. Users of earlier Patchwork versions should instead refer to *XML-RPC API* documentation.

Changed in version 2.1: The API version was bumped to v1.1 in Patchwork v2.1. The older v1.0 API is still supported. For more information, refer to *Supported Versions*.

Changed in version 2.2: The API version was bumped to v1.2 in Patchwork v2.2. The older APIs are still supported. For more information, refer to *Supported Versions*.

Changed in version 3.1: The API version was bumped to v1.3 in Patchwork v3.1. The older APIs are still supported. For more information, refer to *Supported Versions*.

15.1 Getting Started

The easiest way to start experimenting with the API is to use the web browsable API, as described above.

REST APIs run over plain HTTP(S), thus, the API can be interfaced using applications or libraries that support this widespread protocol. One such application is curl, which can be used to both retrieve and send information to the REST API. For example, to get the version of the REST API for a Patchwork instance hosted at *patchwork.example.com*, run:

```
$ curl -s 'https://patchwork.example.com/api/1.3/' | python -m json.tool
{
    "bundles": "https://patchwork.example.com/api/1.3/bundles/",
    "covers": "https://patchwork.example.com/api/1.3/covers/",
    "events": "https://patchwork.example.com/api/1.3/events/",
```

(continues on next page)

(continued from previous page)

```
    "patches": "https://patchwork.example.com/api/1.3/patches/",
    "people": "https://patchwork.example.com/api/1.3/people/",
    "projects": "https://patchwork.example.com/api/1.3/projects/",
    "series": "https://patchwork.example.com/api/1.3/series/",
    "users": "https://patchwork.example.com/api/1.3/users/"
}
```

In addition, a huge variety of libraries are available for interacting with and parsing the output of REST APIs. The requests library is wide-spread and well-supported. To repeat the above example using *requests*:, run

```
$ python
>>> import json
>>> import requests
>>> r = requests.get('https://patchwork.example.com/api/1.3/')
>>> print(json.dumps(r.json(), indent=2))
{
    "bundles": "https://patchwork.example.com/api/1.3/bundles/",
    "covers": "https://patchwork.example.com/api/1.3/covers/",
    "events": "https://patchwork.example.com/api/1.3/events/",
    "patches": "https://patchwork.example.com/api/1.3/patches/",
    "people": "https://patchwork.example.com/api/1.3/people/",
    "projects": "https://patchwork.example.com/api/1.3/projects/",
    "series": "https://patchwork.example.com/api/1.3/series/",
    "users": "https://patchwork.example.com/api/1.3/users/"
}
```

Tools like *curl* and libraries like *requests* can be used to build anything from small utilities to full-fledged clients targeting the REST API. For an overview of existing API clients, refer to *Clients*.

Tip: While you can do a lot with existing installations, it's possible that you might not have access to all resources or may not wish to modify any existing resources. In this case, it might be better to *deploy your own instance of Patchwork locally* and experiment with that instead.

15.2 Versioning

By default, all requests will receive the latest version of the API: currently `1.3`:

```
GET /api HTTP/1.1
```

You should explicitly request this version through the URL to prevent API changes breaking your application:

```
GET /api/1.3 HTTP/1.1
```

Older API versions will be deprecated and removed over time. For more information, refer to *Supported Versions*.

15.3 Schema

Responses are returned as JSON. Blank fields are returned as `null`, rather than being omitted. Timestamps use the ISO 8601 format, times are by default in UTC:

```
YYYY-MM-DDTHH:MM:SSZ
```

Requests should use either query parameters or form-data, depending on the method. Further information is provided below.

15.3.1 Summary Representations

Some resources are particularly large or expensive to compute. When listing these resources, a summary representation is returned that omits certain fields. To get all fields, fetch the detailed representation. For example, listing patches will return summary representations for each patch:

```
GET /patches HTTP/1.1
```

15.3.2 Detailed Representations

When fetching an individual resource, all fields will be returned. For example, fetching a patch with an ID of `123` will return all available fields for that particular resource:

```
GET /patches/123 HTTP/1.1
```

15.4 Parameters

Most API methods take optional parameters. For `GET` requests, these parameters are mostly used for filtering and should be passed as a HTTP query string parameters:

```
$ curl 'https://patchwork.example.com/api/patches?state=under-review'
```

For all other types of requests, including `POST` and `PATCH`, these parameters should be encoded as JSON with a `Content-Type` of `application/json` or passed as form-encoded data:

```
$ curl -X PATCH \
  --header "Content-Type: application/json" \
  --data '{"state":"under-review"}' \
  'http://localhost:8000/api/patches/123/'
```

```
$ curl -X PATCH \
  --form 'state=under-review' \
  'https://patchwork.example.com/api/patches/123'
```

Important: If you do not include the `Content-Type` header in your request, you will receive a `HTTP 200 (OK)` but the resource will not be updated. This header **must** be included.

Changed in version 2.1: API version 1.1 allows filters to be specified multiple times. Prior to this, only the last value for a given filter key would be used.

15.5 Authentication

Patchwork supports authentication using your username and password (basic authentication) or with a token (token authentication). The latter is recommended.

To authenticate with token authentication, you must first obtain a token. This can be done from your profile, e.g. https://patchwork.example.com/user. Once you have a token, run:

```
$ curl -H "Authorization: Token ${token}" \
    'https://patchwork.example.com/api/'
```

To authenticate using basic auth, you should use your Patchwork username and password. To do this, run:

```
$ curl -u ${username}:${password} \
    'https://patchwork.example.com/api/'
```

Not all resources require authentication. Those that do will return `404 (Not Found)` if authentication is not provided to avoid leaking information.

15.6 Pagination

Requests that return multiple items will be paginated by 30 items by default, though this can vary from instance to instance. You can change page using the `?page` parameter. You can also set custom page sizes up to 100 on most endpoints using the `?per_page` parameter.

```
$ curl 'https://patchwork.example.com/api/patches?page=2&per_page=100'
```

15.6.1 Link Header

The Link header includes pagination information:

```
Link: <https://patchwork.example.com/api/patches?page=3&per_page=100>; rel="next",
  <https://patchwork.example.com/api/patches?page=50&per_page=100>; rel="last"
```

The possible `rel` values are:

Name	Description
`next`	The link relation for the immediate next page of results.
`last`	The link relation for the last page of results.
`first`	The link relation for the first page of results.
`prev`	The link relation for the immediate previous page of results.

15.7 Supported Versions

API Version	Since	Supported?
1.0	2.0	✓
1.1	2.1	✓
1.2	2.2	✓
1.3	3.1	✓

Further information about this and more can typically be found in the release notes.

15.8 Schemas

Auto-generated schema documentation is provided below.

15.8.1 API v1.0

GET /api/1.0/

List API resources.

Example request:

```
GET /api/1.0/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "bundles": "https://example.com",
    "covers": "https://example.com",
    "events": "https://example.com",
    "patches": "https://example.com",
    "people": "https://example.com",
    "projects": "https://example.com",
    "users": "https://example.com",
    "series": "https://example.com"
}
```

GET /api/1.0/bundles/

List bundles.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.

- **q** (*string*) – A search term.
- **project** (*string*) – An ID or linkname of a project to filter bundles by.
- **owner** (*string*) – An ID or username of a user to filter bundles by.
- **public** (*string*) – Show only public (*true*) or private (*false*) bundles.

Example request:

```
GET /api/1.0/bundles/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "name": "string",
        "owner": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ],
        "public": true,
        "mbox": "https://example.com"
```

(continues on next page)

(continued from previous page)

```
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/bundles/{id}/`

Show a bundle.

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
GET /api/1.0/bundles/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
```

(continues on next page)

(continued from previous page)

```
            "id": 1,
            "url": "https://example.com",
            "msgid": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.0/covers/

List cover letters.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter cover letters by.
- **series** (*string*) – An ID of a series to filter cover letters by.
- **submitter** (*string*) – An ID or email address of a person to filter cover letters by.

Example request:

```
GET /api/1.0/covers/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)


```
[
    {
        "id": 1,
        "url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "msgid": "string",
        "date": "string",
        "name": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "series": [
            {
                "id": 1,
                "url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ]
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/covers/{id}/`

Show a cover letter.

Parameters

- **id** (*integer*) – A unique integer value identifying this cover letter.

Example request:

```
GET /api/1.0/covers/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "headers": {},
    "content": "string"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "detail": "string"
}
```

`GET /api/1.0/covers/{id}/comments/`

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent cover letter.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.0/covers/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "msgid": "string",
        "date": "string",
        "subject": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {}
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/events/`

List events.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter events by.
- **category** (*string*) – An event category to filter events by. These categories are subject to change depending on the version of Patchwork deployed and are not subject to the versioning constraints present across the rest of the API.
- **series** (*integer*) – An ID of a series to filter events by.
- **patch** (*integer*) – An ID of a patch to filter events by.
- **cover** (*integer*) – An ID of a cover letter to filter events by.

Example request:

```
GET /api/1.0/events/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "category": "cover-created",
        "project": {
            "id": 1,
```

(continues on next page)

(continued from previous page)


```
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-completed",
        "project": {
            "id": 1,
```


(continues on next page)

(continued from previous page)


```
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "series": {
                "id": 1,
                "url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-state-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
```


(continues on next page)

(continued from previous page)


```
            },
            "previous_state": "string",
            "current_state": "string"
        }
    },
    {
        "id": 1,
        "category": "patch-relation-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_relation": "string",
            "current_relation": "string"
        }
    },
    {
        "id": 1,
        "category": "patch-delegated",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            },
            "current_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "check-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "check": {
                "id": 1,
                "url": "https://example.com",
                "date": "string",
                "state": "pending",
```


(continues on next page)

(continued from previous page)


```
                "target_url": "https://example.com",
                "context": "string"
            }
        }
    },
    {
        "id": 1,
        "category": "series-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "series-completed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "name": "string",
                "date": "string",
```


(continues on next page)

(continued from previous page)


```
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "cover-comment-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-comment-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
```


(continues on next page)

(continued from previous page)


```
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string"
            }
        }
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/patches/`

List patches.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter patches by.
- **series** (*integer*) – An ID of a series to filter patches by.
- **submitter** (*string*) – An ID or email address of a person to filter patches by.
- **delegate** (*string*) – An ID or username of a user to filter patches by.
- **state** (*string*) – A slug representation of a state to filter patches by.
- **archived** (*string*) – Show only archived (*true*) or non-archived (*false*) patches.

Example request:

```
GET /api/1.0/patches/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "msgid": "string",
        "date": "string",
        "name": "string",
        "commit_ref": "string",
        "pull_url": "https://example.com",
        "state": "string",
        "archived": true,
        "hash": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "delegate": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "mbox": "https://example.com",
        "series": [
            {
                "id": 1,
                "url": "https://example.com",
                "name": "string",
```

(continues on next page)

(continued from previous page)

```
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ],
        "check": "pending",
        "checks": "https://example.com",
        "tags": {}
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/patches/{id}/`

Show a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
GET /api/1.0/patches/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
```

(continues on next page)

(continued from previous page)


```
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`PATCH /api/1.0/patches/{id}/`

Update a patch (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PATCH /api/1.0/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
```

(continues on next page)

(continued from previous page)


```
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.0/patches/{id}/

Update a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PUT /api/1.0/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
```

(continues on next page)

(continued from previous page)


```
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 400 Bad Request – Invalid Request

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.0/patches/{id}/comments/`

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.0/patches/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "msgid": "string",
        "date": "string",
        "subject": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {}
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

GET /api/1.0/patches/{patch_id}/checks/

List checks.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.

- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **user** (*string*) – An ID or username of a user to filter checks by.
- **state** (*string*) – A check state to filter checks by.
- **context** (*string*) – A check context to filter checks by.

Example request:

```
GET /api/1.0/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "user": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "date": "string",
        "state": "pending",
        "target_url": "https://example.com",
        "context": "string",
        "description": "string"
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`POST /api/1.0/patches/{patch_id}/checks/`

Create a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Example request:

```
POST /api/1.0/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

- 400 Bad Request – Invalid Request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "target_url": [
        "string"
    ],
    "context": [
        "string"
    ],
    "description": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.0/patches/{patch_id}/checks/{check_id}/

Show a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.
- **check_id** (*integer*) – A unique integer value identifying this check.

Example request:

```
GET /api/1.0/patches/{patch_id}/checks/{check_id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.0/people/`

List people.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.0/people/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com",
        "user": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    }
]
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/people/{id}/`

Show a person.

Parameters

- **id** (*integer*) – A unique integer value identifying this person.

Example request:

```
GET /api/1.0/people/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "email": "name@example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    }
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.0/projects/`

List projects.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.0/projects/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "maintainers": [
            {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        ]
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/projects/{id}/`

Show a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
GET /api/1.0/projects/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ]
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.0/projects/{id}/

Update a project (partial).

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PATCH /api/1.0/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ]
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.0/projects/{id}/

Update a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PUT /api/1.0/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ]
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.0/series/`

List series.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **submitter** (*string*) – An ID or email address of a person to filter series by.
- **project** (*string*) – An ID or linkname of a project to filter series by.

Example request:

```
GET /api/1.0/series/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "name": "string",
        "date": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "version": 1,
        "total": 1,
        "received_total": 1,
        "received_all": true,
        "mbox": "https://example.com",
        "cover_letter": {
            "id": 1,
            "url": "https://example.com",
            "msgid": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ]
    }
```

(continues on next page)

(continued from previous page)

```
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/series/{id}/`

Show a series.

Parameters

- **id** (*integer*) – A unique integer value identifying this series.

Example request:

```
GET /api/1.0/series/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "name": "string",
    "date": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "version": 1,
    "total": 1,
    "received_total": 1,
    "received_all": true,
```

(continues on next page)

(continued from previous page)


```
    "mbox": "https://example.com",
    "cover_letter": {
        "id": 1,
        "url": "https://example.com",
        "msgid": "string",
        "date": "string",
        "name": "string",
        "mbox": "https://example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "msgid": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ]
}
```


- 404 Not Found – Not found

  Example response:


```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```


`GET /api/1.0/users/`

List users.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:


```
GET /api/1.0/users/ HTTP/1.1
Host: example.com
```


Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    }
]
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.0/users/{id}/`

Show a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
GET /api/1.0/users/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
```

(continues on next page)

(continued from previous page)

```
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.0/users/{id}/

Update a user (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PATCH /api/1.0/users/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
```

(continues on next page)

(continued from previous page)

```
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.0/users/{id}/

Update a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PUT /api/1.0/users/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```
    "first_name": "string",
    "last_name": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

15.8.2 API v1.1

GET /api/1.1/

List API resources.

Example request:

```
GET /api/1.1/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "bundles": "https://example.com",
    "covers": "https://example.com",
    "events": "https://example.com",
    "patches": "https://example.com",
    "people": "https://example.com",
    "projects": "https://example.com",
    "users": "https://example.com",
    "series": "https://example.com"
}
```

GET /api/1.1/bundles/

List bundles.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **project** (*string*) – An ID or linkname of a project to filter bundles by.
- **owner** (*string*) – An ID or username of a user to filter bundles by.
- **public** (*string*) – Show only public (*true*) or private (*false*) bundles.

Example request:

```
GET /api/1.1/bundles/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "name": "string",
        "owner": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ],
        "public": true,
        "mbox": "https://example.com"
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/bundles/{id}/`

Show a bundle.

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
GET /api/1.1/bundles/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.1/covers/

List cover letters.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter cover letters by.
- **series** (*string*) – An ID of a series to filter cover letters by.
- **submitter** (*string*) – An ID or email address of a person to filter cover letters by.

Example request:

```
GET /api/1.1/covers/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
```

(continues on next page)

(continued from previous page)


```
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "msgid": "string",
        "date": "string",
        "name": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "mbox": "https://example.com",
        "series": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ],
        "comments": "https://example.com"
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/covers/{id}/`

Show a cover letter.

Parameters

- **id** (*integer*) – A unique integer value identifying this cover letter.

Example request:

```
GET /api/1.1/covers/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)


```
{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "headers": {},
    "content": "string"
}
```


- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.1/covers/{id}/comments/`

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent cover letter.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.1/covers/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "web_url": "https://example.com",
        "msgid": "string",
        "date": "string",
        "subject": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {}
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/events/`

List events.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter events by.
- **category** (*string*) – An event category to filter events by. These categories are subject to change depending on the version of Patchwork deployed and are not subject to the versioning constraints present across the rest of the API.
- **series** (*integer*) – An ID of a series to filter events by.
- **patch** (*integer*) – An ID of a patch to filter events by.
- **cover** (*integer*) – An ID of a cover letter to filter events by.

Example request:

```
GET /api/1.1/events/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "category": "cover-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
```

(continues on next page)

(continued from previous page)


```
        "date": "string",
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-completed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-state-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_state": "string",
            "current_state": "string"
        }
```


(continues on next page)

(continued from previous page)


```
    },
    {
        "id": 1,
        "category": "patch-relation-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_relation": "string",
            "current_relation": "string"
        }
    },
    {
        "id": 1,
        "category": "patch-delegated",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
```


(continues on next page)

(continued from previous page)


```
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            },
            "current_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "check-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "check": {
                "id": 1,
                "url": "https://example.com",
                "date": "string",
                "state": "pending",
                "target_url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
                "context": "string"
            }
        }
    },
    {
        "id": 1,
        "category": "series-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "series-completed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
```


(continues on next page)

(continued from previous page)


```
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "cover-comment-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-comment-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "date": "string",
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string"
            }
        }
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/patches/`

List patches.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter patches by.
- **series** (*integer*) – An ID of a series to filter patches by.
- **submitter** (*string*) – An ID or email address of a person to filter patches by.
- **delegate** (*string*) – An ID or username of a user to filter patches by.
- **state** (*string*) – A slug representation of a state to filter patches by.

- **archived** (*string*) – Show only archived (*true*) or non-archived (*false*) patches.

Example request:

```
GET /api/1.1/patches/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "msgid": "string",
        "date": "string",
        "name": "string",
        "commit_ref": "string",
        "pull_url": "https://example.com",
        "state": "string",
        "archived": true,
        "hash": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "delegate": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "mbox": "https://example.com",
        "series": [
```

(continues on next page)

(continued from previous page)


```
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ],
        "comments": "https://example.com",
        "check": "pending",
        "checks": "https://example.com",
        "tags": {}
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/patches/{id}/`

Show a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
GET /api/1.1/patches/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
```


(continues on next page)

(continued from previous page)


```
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.1/patches/{id}/

Update a patch (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PATCH /api/1.1/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
```

(continues on next page)

(continued from previous page)


```
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 400 Bad Request – Invalid Request

Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
```


(continues on next page)

(continued from previous page)

```
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`PUT /api/1.1/patches/{id}/`

Update a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PUT /api/1.1/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "msgid": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
```

(continues on next page)

(continued from previous page)

```
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```

- 400 Bad Request – Invalid Request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```
    "detail": "string"
}
```

GET /api/1.1/patches/{id}/comments/

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.1/patches/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "web_url": "https://example.com",
        "msgid": "string",
        "date": "string",
        "subject": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {}
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

GET /api/1.1/patches/{patch_id}/checks/

List checks.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **user** (*string*) – An ID or username of a user to filter checks by.
- **state** (*string*) – A check state to filter checks by.
- **context** (*string*) – A check context to filter checks by.

Example request:

```
GET /api/1.1/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "user": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
```

(continues on next page)

(continued from previous page)

```
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "date": "string",
        "state": "pending",
        "target_url": "https://example.com",
        "context": "string",
        "description": "string"
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`POST /api/1.1/patches/{patch_id}/checks/`

Create a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Example request:

```
POST /api/1.1/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

- 400 Bad Request – Invalid Request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "target_url": [
        "string"
    ],
    "context": [
        "string"
    ],
    "description": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.1/patches/{patch_id}/checks/{check_id}/`

Show a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.
- **check_id** (*integer*) – A unique integer value identifying this check.

Example request:

```
GET /api/1.1/patches/{patch_id}/checks/{check_id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```
    "detail": "string"
}
```

`GET /api/1.1/people/`

List people.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.1/people/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com",
        "user": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    }
]
```

- 403 Forbidden – Forbidden

Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/people/{id}/`

Show a person.

Parameters

- **id** (*integer*) – A unique integer value identifying this person.

Example request:

```
GET /api/1.1/people/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "email": "name@example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    }
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.1/projects/

List projects.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.1/projects/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "maintainers": [
            {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        ],
        "subject_match": "string"
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/projects/{id}/`

Show a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
GET /api/1.1/projects/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "detail": "string"
}
```

PATCH /api/1.1/projects/{id}/

Update a project (partial).

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PATCH /api/1.1/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string"
}
```

- 400 Bad Request – Bad request

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

- 403 Forbidden – Forbidden

Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`PUT /api/1.1/projects/{id}/`

Update a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PUT /api/1.1/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.1/series/`

List series.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **submitter** (*string*) – An ID or email address of a person to filter series by.
- **project** (*string*) – An ID or linkname of a project to filter series by.

Example request:

```
GET /api/1.1/series/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com"
        },
        "name": "string",
```

(continues on next page)

(continued from previous page)


```
        "date": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "version": 1,
        "total": 1,
        "received_total": 1,
        "received_all": true,
        "mbox": "https://example.com",
        "cover_letter": {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ]
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/series/{id}/`

Show a series.

Parameters

- **id** (*integer*) – A unique integer value identifying this series.

Example request:

```
GET /api/1.1/series/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com"
    },
    "name": "string",
    "date": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "version": 1,
    "total": 1,
    "received_total": 1,
    "received_all": true,
    "mbox": "https://example.com",
    "cover_letter": {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "msgid": "string",
        "date": "string",
        "name": "string",
        "mbox": "https://example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.1/users/

List users.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.1/users/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    }
]
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.1/users/{id}/`

Show a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
GET /api/1.1/users/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.1/users/{id}/

Update a user (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PATCH /api/1.1/users/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`PUT /api/1.1/users/{id}/`

Update a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PUT /api/1.1/users/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

15.8.3 API v1.2

`GET /api/1.2/`

List API resources.

Example request:

```
GET /api/1.2/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "bundles": "https://example.com",
    "covers": "https://example.com",
    "events": "https://example.com",
    "patches": "https://example.com",
    "people": "https://example.com",
```

(continues on next page)

(continued from previous page)

```
    "projects": "https://example.com",
    "users": "https://example.com",
    "series": "https://example.com"
}
```

GET /api/1.2/bundles/

List bundles.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **project** (*string*) – An ID or linkname of a project to filter bundles by.
- **owner** (*string*) – An ID or username of a user to filter bundles by.
- **public** (*string*) – Show only public (*true*) or private (*false*) bundles.

Example request:

```
GET /api/1.2/bundles/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "name": "string",
```

(continues on next page)

(continued from previous page)


```
        "owner": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ],
        "public": true,
        "mbox": "https://example.com"
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`POST /api/1.2/bundles/`

Create a bundle.

Example request:

```
POST /api/1.2/bundles/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "patches": [
        1
    ],
    "public": true
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```

- 400 Bad Request – Invalid Request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

```

(continues on next page)

(continued from previous page)

```
{
    "name": [
        "string"
    ],
    "patches": [
        "string"
    ],
    "public": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.2/bundles/{id}/

Show a bundle.

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
GET /api/1.2/bundles/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
```

(continues on next page)

(continued from previous page)


```
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```


- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.2/bundles/{id}/

Update a bundle (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
PATCH /api/1.2/bundles/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "name": "string",
    "patches": [
        1
    ],
    "public": true
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
```

(continues on next page)

(continued from previous page)

```
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "name": [
        "string"
    ],
    "patches": [
        "string"
    ],
    "public": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`PUT /api/1.2/bundles/{id}/`

Update a bundle.

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
PUT /api/1.2/bundles/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "patches": [
        1
    ],
    "public": true
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
```

(continues on next page)

(continued from previous page)

```
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "name": [
        "string"
    ],
    "patches": [
        "string"
    ],
    "public": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.2/covers/`

List cover letters.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter cover letters by.
- **series** (*string*) – An ID of a series to filter cover letters by.
- **submitter** (*string*) – An ID or email address of a person to filter cover letters by.
- **msgid** (*string*) – The cover message-id as a case-sensitive string, without leading or trailing angle brackets, to filter by.

Example request:

```
GET /api/1.2/covers/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "name": "string",
        "submitter": {
            "id": 1,
```

(continues on next page)

(continued from previous page)

```
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "mbox": "https://example.com",
        "series": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ],
        "comments": "https://example.com"
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.2/covers/{id}/`

Show a cover letter.

Parameters

- **id** (*integer*) – A unique integer value identifying this cover letter.

Example request:

```
GET /api/1.2/covers/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
```

(continues on next page)

(continued from previous page)


```
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "headers": {},
    "content": "string"
}
```


- 404 Not Found – Not found

Example response:


```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```


`GET /api/1.2/covers/{id}/comments/`

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent cover letter.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.2/covers/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "web_url": "https://example.com",
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "subject": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {}
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.2/events/`

List events.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter events by.
- **category** (*string*) – An event category to filter events by. These categories are subject to change depending on the version of Patchwork deployed and are not subject to the versioning constraints present across the rest of the API.
- **series** (*integer*) – An ID of a series to filter events by.
- **patch** (*integer*) – An ID of a patch to filter events by.
- **cover** (*integer*) – An ID of a cover letter to filter events by.

Example request:

```
GET /api/1.2/events/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "category": "cover-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
```

(continues on next page)

(continued from previous page)


```
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-completed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
```


(continues on next page)

(continued from previous page)


```
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-state-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_state": "string",
            "current_state": "string"
        }
    },
    {
        "id": 1,
        "category": "patch-relation-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
```


(continues on next page)

(continued from previous page)


```
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_relation": "string",
            "current_relation": "string"
        }
    },
    {
        "id": 1,
        "category": "patch-delegated",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
```


(continues on next page)

(continued from previous page)


```
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            },
            "current_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "check-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "check": {
                "id": 1,
                "url": "https://example.com",
                "date": "string",
                "state": "pending",
                "target_url": "https://example.com",
                "context": "string"
            }
        }
    },
    {
        "id": 1,
        "category": "series-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
```


(continues on next page)

(continued from previous page)


```
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "series-completed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
```


(continues on next page)

(continued from previous page)


```
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "cover-comment-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string"
            }
        }
    },
```


(continues on next page)

(continued from previous page)


```
    {
        "id": 1,
        "category": "patch-comment-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string"
            }
        }
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/

html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.2/patches/`

List patches.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter patches by.
- **series** (*integer*) – An ID of a series to filter patches by.
- **submitter** (*string*) – An ID or email address of a person to filter patches by.
- **delegate** (*string*) – An ID or username of a user to filter patches by.
- **state** (*string*) – A slug representation of a state to filter patches by.
- **archived** (*string*) – Show only archived (*true*) or non-archived (*false*) patches.
- **hash** (*string*) – The patch hash as a case-insensitive hexadecimal string, to filter by.
- **msgid** (*string*) – The patch message-id as a case-sensitive string, without leading or trailing angle brackets, to filter by.

Example request:

```
GET /api/1.2/patches/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
```

(continues on next page)

(continued from previous page)


```
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "name": "string",
        "commit_ref": "string",
        "pull_url": "https://example.com",
        "state": "string",
        "archived": true,
        "hash": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "delegate": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "mbox": "https://example.com",
        "series": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ],
        "comments": "https://example.com",
        "check": "pending",
        "checks": "https://example.com",
        "tags": {},
        "related": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ]
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.2/patches/{id}/`

Show a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
GET /api/1.2/patches/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
```


(continues on next page)

(continued from previous page)


```
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "related": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "headers": {},
```


(continues on next page)

(continued from previous page)

```
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.2/patches/{id}/

Update a patch (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PATCH /api/1.2/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1,
    "related": [
        1
    ]
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
```

(continues on next page)

(continued from previous page)


```
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
```


(continues on next page)

(continued from previous page)


```
    "related": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```


- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

- 409 Conflict – Conflict

Example response:

```
HTTP/1.1 409 Conflict
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.2/patches/{id}/

Update a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PUT /api/1.2/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1,
    "related": [
        1
    ]
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
```

(continues on next page)

(continued from previous page)


```
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
    "tags": {},
    "related": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

- 409 Conflict – Conflict

  Example response:

```
HTTP/1.1 409 Conflict
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.2/patches/{id}/comments/`

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.2/patches/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "web_url": "https://example.com",
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
```

(continues on next page)

(continued from previous page)


```
        "subject": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {}
    }
]
```


- 404 Not Found – Not found

  Example response:


```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

GET /api/1.2/patches/{patch_id}/checks/

List checks.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **user** (*string*) – An ID or username of a user to filter checks by.
- **state** (*string*) – A check state to filter checks by.
- **context** (*string*) – A check context to filter checks by.

Example request:

```
GET /api/1.2/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "user": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "date": "string",
        "state": "pending",
        "target_url": "https://example.com",
        "context": "string",
        "description": "string"
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`POST /api/1.2/patches/{patch_id}/checks/`

Create a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Example request:

```
POST /api/1.2/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

- 400 Bad Request – Invalid Request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "target_url": [
        "string"
    ],
    "context": [
        "string"
    ],
```

(continues on next page)

(continued from previous page)

```
    "description": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.2/patches/{patch_id}/checks/{check_id}/

Show a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.
- **check_id** (*integer*) – A unique integer value identifying this check.

Example request:

```
GET /api/1.2/patches/{patch_id}/checks/{check_id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
```

(continues on next page)

(continued from previous page)

```
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.2/people/

List people.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.2/people/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com",
        "user": {
            "id": 1,
```

(continues on next page)

(continued from previous page)

```
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    }
]
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

GET /api/1.2/people/{id}/

Show a person.

Parameters

- **id** (*integer*) – A unique integer value identifying this person.

Example request:

```
GET /api/1.2/people/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "email": "name@example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
```

(continues on next page)

(continued from previous page)

```
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    }
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.2/projects/`

List projects.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.2/projects/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
```

(continues on next page)

(continued from previous page)


```
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "maintainers": [
            {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        ],
        "subject_match": "string",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.2/projects/{id}/`

Show a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
GET /api/1.2/projects/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```


- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.2/projects/{id}/

Update a project (partial).

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PATCH /api/1.2/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
```

(continues on next page)

(continued from previous page)

```
    "commit_url_format": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.2/projects/{id}/

Update a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PUT /api/1.2/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
```

(continues on next page)

(continued from previous page)


```
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```


- 400 Bad Request – Bad request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```


- 404 Not Found – Not found

  Example response:


```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.2/series/`

List series.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **submitter** (*string*) – An ID or email address of a person to filter series by.
- **project** (*string*) – An ID or linkname of a project to filter series by.

Example request:

```
GET /api/1.2/series/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "name": "string",
        "date": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
```

(continues on next page)

(continued from previous page)


```
            "email": "name@example.com"
        },
        "version": 1,
        "total": 1,
        "received_total": 1,
        "received_all": true,
        "mbox": "https://example.com",
        "cover_letter": {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ]
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.2/series/{id}/`

Show a series.

Parameters

- **id** (*integer*) – A unique integer value identifying this series.

Example request:

```
GET /api/1.2/series/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "date": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "version": 1,
    "total": 1,
    "received_total": 1,
    "received_all": true,
    "mbox": "https://example.com",
    "cover_letter": {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "name": "string",
        "mbox": "https://example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
```

(continues on next page)

(continued from previous page)

```
            "mbox": "https://example.com"
        }
    ]
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.2/users/

List users.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.2/users/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    }
]
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.2/users/{id}/`

Show a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
GET /api/1.2/users/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.2/users/{id}/

Update a user (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PATCH /api/1.2/users/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
```

(continues on next page)

(continued from previous page)

```
        "show_ids": true
    }
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.2/users/{id}/

Update a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PUT /api/1.2/users/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string",
    "settings": {
```

(continues on next page)

(continued from previous page)

```
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

15.8.4 API v1.3 (latest)

`GET /api/1.3/`

List API resources.

Example request:

```
GET /api/1.3/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "bundles": "https://example.com",
    "covers": "https://example.com",
    "events": "https://example.com",
    "patches": "https://example.com",
    "people": "https://example.com",
    "projects": "https://example.com",
    "users": "https://example.com",
    "series": "https://example.com"
}
```

`GET /api/1.3/bundles/`

List bundles.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **project** (*string*) – An ID or linkname of a project to filter bundles by.
- **owner** (*string*) – An ID or username of a user to filter bundles by.
- **public** (*string*) – Show only public (*true*) or private (*false*) bundles.

Example request:

```
GET /api/1.3/bundles/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "name": "string",
        "owner": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ],
        "public": true,
        "mbox": "https://example.com"
    }
```

(continues on next page)

(continued from previous page)

```
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`POST /api/1.3/bundles/`

Create a bundle.

Example request:

```
POST /api/1.3/bundles/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "patches": [
        1
    ],
    "public": true
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
```

(continues on next page)

(continued from previous page)


```
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "name": [
        "string"
    ],
    "patches": [
        "string"
    ],
    "public": [
        "string"
    ]
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.3/bundles/{id}/

Show a bundle.

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
GET /api/1.3/bundles/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
```

(continues on next page)

(continued from previous page)

```
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.3/bundles/{id}/

Update a bundle (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
PATCH /api/1.3/bundles/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "patches": [
        1
    ],
    "public": true
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
```

(continues on next page)

(continued from previous page)


```
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```


- 400 Bad Request – Bad request

Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "name": [
        "string"
    ],
    "patches": [
        "string"
    ],
    "public": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.3/bundles/{id}/

Update a bundle.

Parameters

- **id** (*integer*) – A unique integer value identifying this bundle.

Example request:

```
PUT /api/1.3/bundles/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "patches": [
        1
    ],
    "public": true
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
```

(continues on next page)

(continued from previous page)


```
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "owner": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "patches": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "public": true,
    "mbox": "https://example.com"
}
```


- 400 Bad Request – Bad request

Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "name": [
        "string"
    ],
    "patches": [
        "string"
    ],
    "public": [
        "string"
```


(continues on next page)

(continued from previous page)

```
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.3/covers/

List cover letters.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter cover letters by.
- **series** (*string*) – An ID of a series to filter cover letters by.
- **submitter** (*string*) – An ID or email address of a person to filter cover letters by.
- **msgid** (*string*) – The cover message-id as a case-sensitive string, without leading or trailing angle brackets, to filter by.

Example request:

```
GET /api/1.3/covers/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "name": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "mbox": "https://example.com",
        "series": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ],
        "comments": "https://example.com"
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.3/covers/{id}/`

Show a cover letter.

Parameters

- **id** (*integer*) – A unique integer value identifying this cover letter.

Example request:

```
GET /api/1.3/covers/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
```

(continues on next page)

(continued from previous page)

```
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "headers": {},
    "content": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.3/covers/{id}/comments/

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent cover letter.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.3/covers/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "web_url": "https://example.com",
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
```

(continues on next page)

(continued from previous page)

```
        "subject": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {},
        "addressed": true
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

GET /api/1.3/covers/{cover_id}/comments/{comment_id}/

Show a cover comment.

Parameters

- **cover_id** (*integer*) – A unique integer value identifying the parent cover.
- **comment_id** (*integer*) – A unique integer value identifying this comment.

Example request:

```
GET /api/1.3/covers/{cover_id}/comments/{comment_id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "web_url": "https://example.com",
```

(continues on next page)

(continued from previous page)


```
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "subject": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "content": "string",
    "headers": {},
    "addressed": true
}
```


- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`PATCH /api/1.3/covers/{cover_id}/comments/{comment_id}/`

Update a cover comment (partial).

Parameters

- **cover_id** (*integer*) – A unique integer value identifying the parent cover.
- **comment_id** (*integer*) – A unique integer value identifying this comment.

Example request:

```
PATCH /api/1.3/covers/{cover_id}/comments/{comment_id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "addressed": true
}
```

Status Codes

- 200 OK – **Example response:**


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
```


(continues on next page)

(continued from previous page)


```
    "web_url": "https://example.com",
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "subject": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "content": "string",
    "headers": {},
    "addressed": true
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "addressed": [
        "string"
    ]
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```


- 404 Not Found – Not found

  Example response:


```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```


`GET /api/1.3/events/`

List events.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter events by.
- **category** (*string*) – An event category to filter events by. These categories are subject to change depending on the version of Patchwork deployed and are not subject to the versioning constraints present across the rest of the API.
- **series** (*integer*) – An ID of a series to filter events by.
- **patch** (*integer*) – An ID of a patch to filter events by.
- **cover** (*integer*) – An ID of a cover letter to filter events by.

Example request:

```
GET /api/1.3/events/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "category": "cover-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
```

(continues on next page)

(continued from previous page)


```
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
```


(continues on next page)

(continued from previous page)


```
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-completed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
```


(continues on next page)

(continued from previous page)


```
    {
        "id": 1,
        "category": "patch-state-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_state": "string",
            "current_state": "string"
        }
    },
    {
        "id": 1,
        "category": "patch-relation-changed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_relation": "string",
            "current_relation": "string"
        }
    },
    {
        "id": 1,
        "category": "patch-delegated",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
```


(continues on next page)

(continued from previous page)


```
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "previous_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            },
            "current_delegate": {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "check-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
```


(continues on next page)

(continued from previous page)


```
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "check": {
                "id": 1,
                "url": "https://example.com",
                "date": "string",
                "state": "pending",
                "target_url": "https://example.com",
                "context": "string"
            }
        }
    },
    {
        "id": 1,
        "category": "series-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
```


(continues on next page)

(continued from previous page)


```
            "email": "name@example.com"
        },
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
    {
        "id": 1,
        "category": "series-completed",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "series": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        }
    },
```


(continues on next page)

(continued from previous page)


```
    {
        "id": 1,
        "category": "cover-comment-created",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "cover": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string"
            }
        }
    },
    {
        "id": 1,
        "category": "patch-comment-created",
        "project": {
```


(continues on next page)

(continued from previous page)


```
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "date": "string",
        "actor": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "payload": {
            "patch": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            },
            "comment": {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string"
            }
        }
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.3/patches/`

List patches.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **project** (*string*) – An ID or linkname of a project to filter patches by.
- **series** (*integer*) – An ID of a series to filter patches by.
- **submitter** (*string*) – An ID or email address of a person to filter patches by.
- **delegate** (*string*) – An ID or username of a user to filter patches by.
- **state** (*string*) – A slug representation of a state to filter patches by.
- **archived** (*string*) – Show only archived (*true*) or non-archived (*false*) patches.
- **hash** (*string*) – The patch hash as a case-insensitive hexadecimal string, to filter by.
- **msgid** (*string*) – The patch message-id as a case-sensitive string, without leading or trailing angle brackets, to filter by.

Example request:

```
GET /api/1.3/patches/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
```

(continues on next page)

(continued from previous page)


```
            "list_archive_url": "https://example.com",
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "name": "string",
        "commit_ref": "string",
        "pull_url": "https://example.com",
        "state": "string",
        "archived": true,
        "hash": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "delegate": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "mbox": "https://example.com",
        "series": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "name": "string",
                "date": "string",
                "version": 1,
                "mbox": "https://example.com"
            }
        ],
        "comments": "https://example.com",
        "check": "pending",
        "checks": "https://example.com",
        "tags": {},
        "related": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
```


(continues on next page)

(continued from previous page)

```
                "mbox": "https://example.com"
            }
        ]
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.3/patches/{id}/`

Show a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
GET /api/1.3/patches/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
```

(continues on next page)

(continued from previous page)


```
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "related": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
```


(continues on next page)

(continued from previous page)

```
    ]
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.3/patches/{id}/

Update a patch (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PATCH /api/1.3/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1,
    "related": [
        1
    ]
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
```

(continues on next page)

(continued from previous page)


```
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
    "related": [
        {
            "id": 1,
            "url": "https://example.com",
```


(continues on next page)

(continued from previous page)


```
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```


- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

- 409 Conflict – Conflict

Example response:

```
HTTP/1.1 409 Conflict
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.3/patches/{id}/

Update a patch.

Parameters

- **id** (*integer*) – A unique integer value identifying this patch.

Example request:

```
PUT /api/1.3/patches/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "delegate": 1,
    "related": [
        1
    ]
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
```

(continues on next page)

(continued from previous page)


```
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "msgid": "string",
    "list_archive_url": "string",
    "date": "string",
    "name": "string",
    "commit_ref": "string",
    "pull_url": "https://example.com",
    "state": "string",
    "archived": true,
    "hash": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "delegate": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "mbox": "https://example.com",
    "series": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "name": "string",
            "date": "string",
            "version": 1,
            "mbox": "https://example.com"
        }
    ],
    "comments": "https://example.com",
    "check": "pending",
    "checks": "https://example.com",
    "tags": {},
```


(continues on next page)

(continued from previous page)


```
    "related": [
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ],
    "headers": {},
    "content": "string",
    "diff": "string",
    "prefixes": [
        "string"
    ]
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "delegate": [
        "string"
    ],
    "commit_ref": [
        "string"
    ],
    "archived": [
        "string"
    ]
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```


- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

- 409 Conflict – Conflict

Example response:

```
HTTP/1.1 409 Conflict
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.3/patches/{id}/comments/`

List comments

Parameters

- **id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.3/patches/{id}/comments/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "web_url": "https://example.com",
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "subject": "string",
        "submitter": {
```

(continues on next page)

(continued from previous page)


```
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "content": "string",
        "headers": {},
        "addressed": true
    }
]
```


- 404 Not Found – Not found

  Example response:


```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

GET /api/1.3/patches/{patch_id}/comments/{comment_id}/

Show a patch comment.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.
- **comment_id** (*integer*) – A unique integer value identifying this comment.

Example request:

```
GET /api/1.3/patches/{patch_id}/comments/{comment_id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "web_url": "https://example.com",
    "msgid": "string",
    "list_archive_url": "string",
```


(continues on next page)

(continued from previous page)


```
    "date": "string",
    "subject": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "content": "string",
    "headers": {},
    "addressed": true
}
```


- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.3/patches/{patch_id}/comments/{comment_id}/

Update a patch comment (partial).

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.
- **comment_id** (*integer*) – A unique integer value identifying this comment.

Example request:

```
PATCH /api/1.3/patches/{patch_id}/comments/{comment_id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "addressed": true
}
```

Status Codes

- 200 OK – **Example response:**


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "web_url": "https://example.com",
    "msgid": "string",
```


(continues on next page)

(continued from previous page)


```
    "list_archive_url": "string",
    "date": "string",
    "subject": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "content": "string",
    "headers": {},
    "addressed": true
}
```


- 400 Bad Request – Invalid Request

  Example response:


```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "addressed": [
        "string"
    ]
}
```


- 403 Forbidden – Forbidden

  Example response:


```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```


- 404 Not Found – Not found

  Example response:


```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```


GET /api/1.3/patches/{patch_id}/checks/

List checks.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **user** (*string*) – An ID or username of a user to filter checks by.
- **state** (*string*) – A check state to filter checks by.
- **context** (*string*) – A check context to filter checks by.

Example request:

```
GET /api/1.3/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "user": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        },
        "date": "string",
        "state": "pending",
        "target_url": "https://example.com",
        "context": "string",
        "description": "string"
    }
]
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`POST /api/1.3/patches/{patch_id}/checks/`

Create a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.

Example request:

```
POST /api/1.3/patches/{patch_id}/checks/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
```

(continues on next page)

(continued from previous page)

```
    "description": "string"
}
```

- 400 Bad Request – Invalid Request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "state": [
        "string"
    ],
    "target_url": [
        "string"
    ],
    "context": [
        "string"
    ],
    "description": [
        "string"
    ]
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.3/patches/{patch_id}/checks/{check_id}/`

Show a check.

Parameters

- **patch_id** (*integer*) – A unique integer value identifying the parent patch.
- **check_id** (*integer*) – A unique integer value identifying this check.

Example request:

```
GET /api/1.3/patches/{patch_id}/checks/{check_id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    },
    "date": "string",
    "state": "pending",
    "target_url": "https://example.com",
    "context": "string",
    "description": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.3/people/`

List people.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.3/people/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com",
        "user": {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    }
]
```

- 403 Forbidden – Forbidden

Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

GET /api/1.3/people/{id}/

Show a person.

Parameters

- **id** (*integer*) – A unique integer value identifying this person.

Example request:

```
GET /api/1.3/people/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "email": "name@example.com",
    "user": {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
        "last_name": "string",
        "email": "name@example.com"
    }
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.3/projects/`

List projects.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.3/projects/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "maintainers": [
            {
                "id": 1,
                "url": "https://example.com",
                "username": "string",
                "first_name": "string",
                "last_name": "string",
                "email": "name@example.com"
            }
        ],
        "subject_match": "string",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    }
]
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.3/projects/{id}/`

Show a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
GET /api/1.3/projects/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`PATCH /api/1.3/projects/{id}/`

Update a project (partial).

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PATCH /api/1.3/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.3/projects/{id}/

Update a project.

Parameters

- **id** (*string*) – A unique integer value identifying this project.

Example request:

```
PUT /api/1.3/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "name": "string",
    "link_name": "string",
    "list_id": "string",
    "list_email": "name@example.com",
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com",
    "maintainers": [
        {
            "id": 1,
            "url": "https://example.com",
            "username": "string",
            "first_name": "string",
            "last_name": "string",
            "email": "name@example.com"
        }
    ],
    "subject_match": "string",
    "list_archive_url": "https://example.com",
    "list_archive_url_format": "https://example.com",
    "commit_url_format": "string"
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "web_url": "https://example.com",
    "scm_url": "https://example.com",
    "webscm_url": "https://example.com"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

`GET /api/1.3/series/`

List series.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.
- **before** (*string*) – Latest date-time to retrieve results for.
- **since** (*string*) – Earliest date-time to retrieve results for.
- **submitter** (*string*) – An ID or email address of a person to filter series by.
- **project** (*string*) – An ID or linkname of a project to filter series by.

Example request:

```
GET /api/1.3/series/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "project": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "link_name": "string",
            "list_id": "string",
            "list_email": "name@example.com",
            "web_url": "https://example.com",
            "scm_url": "https://example.com",
            "webscm_url": "https://example.com",
            "list_archive_url": "https://example.com",
```

(continues on next page)

(continued from previous page)


```
            "list_archive_url_format": "https://example.com",
            "commit_url_format": "string"
        },
        "name": "string",
        "date": "string",
        "submitter": {
            "id": 1,
            "url": "https://example.com",
            "name": "string",
            "email": "name@example.com"
        },
        "version": 1,
        "total": 1,
        "received_total": 1,
        "received_all": true,
        "mbox": "https://example.com",
        "cover_letter": {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        },
        "patches": [
            {
                "id": 1,
                "url": "https://example.com",
                "web_url": "https://example.com",
                "msgid": "string",
                "list_archive_url": "string",
                "date": "string",
                "name": "string",
                "mbox": "https://example.com"
            }
        ]
    }
]
```


Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.3/series/{id}/`

Show a series.

Parameters

- **id** (*integer*) – A unique integer value identifying this series.

Example request:

```
GET /api/1.3/series/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "web_url": "https://example.com",
    "project": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "link_name": "string",
        "list_id": "string",
        "list_email": "name@example.com",
        "web_url": "https://example.com",
        "scm_url": "https://example.com",
        "webscm_url": "https://example.com",
        "list_archive_url": "https://example.com",
        "list_archive_url_format": "https://example.com",
        "commit_url_format": "string"
    },
    "name": "string",
    "date": "string",
    "submitter": {
        "id": 1,
        "url": "https://example.com",
        "name": "string",
        "email": "name@example.com"
    },
    "version": 1,
    "total": 1,
    "received_total": 1,
    "received_all": true,
    "mbox": "https://example.com",
    "cover_letter": {
        "id": 1,
        "url": "https://example.com",
        "web_url": "https://example.com",
        "msgid": "string",
        "list_archive_url": "string",
        "date": "string",
        "name": "string",
        "mbox": "https://example.com"
    },
    "patches": [
```

(continues on next page)

(continued from previous page)

```
        {
            "id": 1,
            "url": "https://example.com",
            "web_url": "https://example.com",
            "msgid": "string",
            "list_archive_url": "string",
            "date": "string",
            "name": "string",
            "mbox": "https://example.com"
        }
    ]
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

GET /api/1.3/users/

List users.

Query Parameters

- **page** (*integer*) – A page number within the paginated result set.
- **per_page** (*integer*) – Number of results to return per page.
- **order** (*string*) – Which field to use when ordering the results.
- **q** (*string*) – A search term.

Example request:

```
GET /api/1.3/users/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "url": "https://example.com",
        "username": "string",
        "first_name": "string",
```

(continues on next page)

(continued from previous page)

```
        "last_name": "string",
        "email": "name@example.com"
    }
]
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

Response Headers

- **Link** – Links to related resources, in the format defined by [RFC 5988](https://tools.ietf.org/html/rfc5988#section-5). This will include a link with relation type *next* to the next page and *prev* to the previous page, if there is a next or previous page. It will also include links with the relation type *first* and *last* pointing to the first and last page, respectively.

`GET /api/1.3/users/{id}/`

Show a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
GET /api/1.3/users/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PATCH /api/1.3/users/{id}/

Update a user (partial).

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PATCH /api/1.3/users/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
```

(continues on next page)

(continued from previous page)

```
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

PUT /api/1.3/users/{id}/

Update a user.

Parameters

- **id** (*integer*) – A unique integer value identifying this user.

Example request:

```
PUT /api/1.3/users/{id}/ HTTP/1.1
Host: example.com
```

(continues on next page)

(continued from previous page)

```
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "url": "https://example.com",
    "username": "string",
    "first_name": "string",
    "last_name": "string",
    "email": "name@example.com",
    "settings": {
        "send_email": true,
        "items_per_page": 1,
        "show_ids": true
    }
}
```

- 400 Bad Request – Bad request

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "first_name": "string",
    "last_name": "string"
}
```

- 403 Forbidden – Forbidden

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```
    "detail": "string"
}
```

- 404 Not Found – Not found

  Example response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "detail": "string"
}
```

CHAPTER

SIXTEEN

THE XML-RPC API

Patchwork provides an XML-RPC API. This API can be used to be used to retrieve and modify information about patches, projects and more.

Important: The XML-RPC API can be enabled/disabled by the administrator: it may not be available in every instance. Refer to `/about` on your given instance for the status of the API, e.g.

https://patchwork.ozlabs.org/about

Alternatively, simply attempt to make a request to the API.

Deprecated since version 2.0: The XML-RPC API is a legacy API and has been deprecated in favour of the *REST API*. It may be removed in a future release.

16.1 Getting Started

The Patchwork XML-RPC API provides a number of "methods". Some methods require authentication (via HTTP Basic Auth) while others do not. Authentication uses your Patchwork account and the on-server documentation will indicate where it is necessary. We will only cover the unauthenticated method here for brevity - consult the xmlrpc documentation for more detailed examples:

To interact with the Patchwork XML-RPC API, a XML-RPC library should be used. Python provides such a library - xmlrpc - in its standard library. For example, to get the version of the XML-RPC API for a Patchwork instance hosted at *patchwork.example.com*, run:

```
$ python
>>> import xmlrpc.client
>>> rpc = xmlrpc.client.ServerProxy('http://patchwork.example.com/xmlrpc/')
>>> rpc.pw_rpc_version()
1.1
```

Once connected, the `rpc` object will be populated with a list of available functions (or procedures, in RPC terminology). In the above example, we used the `pw_rpc_version` method, however, it should be possible to use all the methods listed in the server documentation.

16.2 Further Information

Patchwork provides automatically generated documentation for the XML-RPC API. You can find this at the following URL:

https://patchwork.example.com/xmlrpc/

where *patchwork.example.com* refers to the URL of your Patchwork instance.

Changed in version 1.1: Automatic documentation generation for the Patchwork API was introduced in Patchwork v1.1. Prior versions of Patchwork do not offer this functionality.

CHAPTER

SEVENTEEN

UNRELEASED

17.1 v3.1.1

17.1.1 New Features

- The version of Django used is now checked on start-up. This can help avoid issues where deployers forget to update their Django version and see odd behavior as a result.

17.1.2 Bug Fixes

- Fix an issue whereby comment-based events would cause a HTTP 500 error for the events API (`/api/events`).

17.2 v3.1.0

17.2.1 Prelude

This release is one of the smaller releases. The primary new feature is the ability to mark comments as addressed/unaddressed, allowing maintainers to track work items. In addition, two new events have been added to the `/events` API: `cover-comment-created` and `patch-comment-created`. Recent versions of Python (3.10) and Django (3.2, 4.0) are now supported, while support for older Python (3.6) and Django (2.2, 3.0, 3.1) versions has been removed. More information on all the above is included below.

17.2.2 New Features

- Patch comments and cover letter comments can be marked 'addressed' or 'unaddressed' to reflect whether the comment has been addressed by the patch and cover letter submitter or a reviewer. The current state of a comment is shown in the header when showing a comment and users with edit permission can toggle the state using an adjacent button.
- Two new event types have been added: `cover-comment-created` and `patch-comment-created`. As their names would suggest, these track the creation of new cover letter and patch comments respectively.
- Django 3.2 is now supported.
- Django 4.0 is now supported.
- The `Link` header included in REST API responses now includes `first` and `last` relations, as described in RFC 5988. As their name would suggest, these can be used to navigate to the beginning and end of the resource.
- Python 3.10 is now supported.

17.2.3 Upgrade Notes

- Django 3.0 is no longer supported. It is no longer supported upstream and most distributions provide a newer version.
- Django 3.1 is no longer supported. It is no longer supported upstream and most distributions provide a newer version.
- Python 3.6 is no longer supported. It is no longer supported upstream and most distributions provide a newer version.
- Django 2.2 is no longer supported. Is is no longer supported upstream and most distributions provide a newer version.
- Database configuration has been added to `patchwork.settings.base`. Previously, this had to be specified in the `settings.py` file manually created by admins. The following environment variables can now be used to configure the settings:
  - `DATABASE_TYPE` (one of: `postgres`, `sqlite3`, `mysql`; default: `mysql`)
  - `DATABASE_HOST` (default: `localhost`)
  - `DATABASE_PORT` (default: <unset>)
  - `DATABASE_NAME` (default: `patchwork`)
  - `DATABASE_USER` (default: `patchwork`)
  - `DATABASE_PASSWORD` (default: `patchwork`)

  If you are manually defining `DATABASES` in your `settings.py` file, this should have no impact. However, you may wish to rework your deployment to use environment variables instead.
- Database configuration variables `PW_TEST_DB_*` are renamed to their corresponding `DATABASE_*` names to sync development & production environments setup. Some mistaken references to `DATABASE_PASS` are also replaced with `DATABASE_PASSWORD` to follow the convention.

17.2.4 Bug Fixes

- Fixed a compatability issue with Django 3.1 that prevented users from resetting their password. (#394)
- Comments and whitespace are now correctly stripped from the `Message-ID`, `In-Reply-To`, and `References` headers. One side effect of this change is that the parser is now stricter with regards to the format of the `msg-id` component of these headers: all identifiers must now be surrounded by angle brackets, e.g. <abcdef@example.com>. This is mandated in the spec and a review of mailing lists archives suggest it is broadly adhered to. Without these markers, there is no way to delimit `msg-id` from any surrounding comments and whitespace.

17.2.5 API Changes

- The API version has been updated to v1.3.
- A new REST API endpoint is available at `/api/covers/<cover_id>/comments/<comment_id>/`. This can be used to retrieve and update (e.g. `addressed` state) details about a specific cover comment.
- A new REST API endpoint is available at `/api/patches/<patch_id>/comments/<comment_id>/`. This can be used to retrieve and update (e.g. `addressed` state) details about a specific patch comment.
- The `list_archive_url` field will now be correctly shown for patch comments and cover letter comments. (#391)

17.3 v3.0.0

17.3.1 Prelude

There are two main changes in this release: the removal of Python 2.7 support and the resolution of the longstanding performance issues introduced by the `Submission` model. On top of this, there is the usual bump in requirements, a significant amount of fixes to the documentation for the REST API, and the squashing of all migrations introduced in versions up to and including v2.2.0.

17.3.2 New Features

- Django 3.0 is now supported.
- Django 3.1 is now supported.
- Django REST Framework 3.12 is now supported.
- Python 3.9 is now supported.

17.3.3 Upgrade Notes

- Django 1.11, 2.0 and 2.1 are no longer supported. These are no longer supported upstream and most distributions provide a newer version.
- djangorestframework 3.6, 3.7, 3.8 and 3.9 are no longer supported. These were only used with Django 1.11 to 2.1 and are not compatible with any version now supported by Patchwork.
- django-filter 1.1.0 is no longer supported. This was only used with Django 1.11 and is not compatible with any version now supported by Patchwork.
- Python 2.7 and 3.5 are no longer supported. These are no longer supported upstream and most distributions provide a newer version.

17.3.4 Bug Fixes

- An issue that preventing updating bundles via the REST API without updating the included patches has been resolved. (#357)
- The parser module now uses an atomic select-insert when creating new patch, cover letter and comment entries. This prevents the integrity errors from being logged in the DB logs. (#358)
- Resolve a bug that would prevent listing patches for a project via the browseable API view when logged in with admin permissions (issue #379)
- Previously, it was possible to create a project with a `linkname` containing invalid URL characters. This would result in broken URLs. We now validate this field and restrict characters to unicode slugs (unicode letters, numbers, underscores and hyphens).

17.3.5 API Changes

- The REST API now supports filtering patches and cover letters by message ID, using the `msgid` query parameter. Don't include leading or trailing angle brackets.

CHAPTER

EIGHTEEN

V3.0 SERIES ("GROSGRAIN")

18.1 v3.0.5

18.1.1 Bug Fixes

- Comments and whitespace are now correctly stripped from the `Message-ID`, `In-Reply-To`, and `References` headers. One side effect of this change is that the parser is now stricter with regards to the format of the `msg-id` component of these headers: all identifiers must now be surrounded by angle brackets, e.g. <`abcdef@example.com`>. This is mandated in the spec and a review of mailing lists archives suggest it is broadly adhered to. Without these markers, there is no way to delimit `msg-id` from any surrounding comments and whitespace.

18.2 v3.0.1

18.2.1 Bug Fixes

- Fixed a compatability issue with Django 3.1 that prevented users from resetting their password. (#394)

18.2.2 API Changes

- The `list_archive_url` field will now be correctly shown for patch comments and cover letter comments. (#391)

18.3 v3.0.0

18.3.1 Prelude

There are two main changes in this release: the removal of Python 2.7 support and the resolution of the longstanding performance issues introduced by the `Submission` model. On top of this, there is the usual bump in requirements, a significant amount of fixes to the documentation for the REST API, and the squashing of all migrations introduced in versions up to and including v2.2.0.

18.3.2 New Features

- Django 3.0 is now supported.
- Django 3.1 is now supported.
- Django REST Framework 3.12 is now supported.
- Python 3.9 is now supported.

18.3.3 Upgrade Notes

- Django 1.11, 2.0 and 2.1 are no longer supported. These are no longer supported upstream and most distributions provide a newer version.
- djangorestframework 3.6, 3.7, 3.8 and 3.9 are no longer supported. These were only used with Django 1.11 to 2.1 and are not compatible with any version now supported by Patchwork.
- django-filter 1.1.0 is no longer supported. This was only used with Django 1.11 and is not compatible with any version now supported by Patchwork.
- Python 2.7 and 3.5 are no longer supported. These are no longer supported upstream and most distributions provide a newer version.

18.3.4 Bug Fixes

- An issue that preventing updating bundles via the REST API without updating the included patches has been resolved. (#357)
- The parser module now uses an atomic select-insert when creating new patch, cover letter and comment entries. This prevents the integrity errors from being logged in the DB logs. (#358)
- Resolve a bug that would prevent listing patches for a project via the browseable API view when logged in with admin permissions (issue #379)
- Previously, it was possible to create a project with a `linkname` containing invalid URL characters. This would result in broken URLs. We now validate this field and restrict characters to unicode slugs (unicode letters, numbers, underscores and hyphens).

18.3.5 API Changes

- The REST API now supports filtering patches and cover letters by message ID, using the `msgid` query parameter. Don't include leading or trailing angle brackets.

CHAPTER

NINETEEN

V2.2 SERIES ("FLANNEL")

19.1 v2.2.4

19.1.1 API Changes

- The `list_archive_url` field will now be correctly shown for patch comments and cover letter comments. (#391)

19.2 v2.2.3

19.2.1 Bug Fixes

- Resolve a bug that would prevent listing patches for a project via the browseable API view when logged in with admin permissions (issue #379)

19.3 v2.2.2

19.3.1 Bug Fixes

- An issue that preventing updating bundles via the REST API without updating the included patches has been resolved. (#357)
- The parser module now uses an atomic select-insert when creating new patch, cover letter and comment entries. This prevents the integrity errors from being logged in the DB logs. (#358)

19.4 v2.2.1

19.4.1 API Changes

- The REST API now supports filtering patches and cover letters by message ID, using the `msgid` query parameter. Don't include leading or trailing angle brackets.

19.5 v2.2.0

19.5.1 New Features

- Patches can now be related to other patches (e.g. to cross-reference revisions). Relations can be set via the REST API by maintainers (currently only for patches of projects they maintain). Patches can belong to at most one relation at a time.
- Django 2.0 is now supported. This requires Python 3.
- Django 2.1 is now supported. This requires Python 3.
- Django 2.2 is now supported. This requires Python 3.
- The `patch-delegated`, `patch-state-changed` and `check-created` events now have an `actor` associated with them - the user that updated the patch or created the check. For other event types, this attribute is present but unset.
- Add a field to Project to store a link to the project's mailing list archive, and display that on the project info page.
- Add a field to Project to store a URL format for a Message-ID redirector for the project's mailing list archive, and display a link to the email thread for each patch.
- Exporting patchwork projects as mbox files and optionally compressing them is now possible with the `./manage exportproject` management command.
- The URL schema now uses message IDs, rather than patch IDs, by default. Old URLs will redirect to the new URLs.
- Python 3.7 is now supported.
- Python 3.8 is now supported.

19.5.2 Upgrade Notes

- django-filter 1.1 is now supported.
- django-filter 2.0 is now supported. This requires Python 3.
- Django REST Framework 3.10 is now supported.
- Django REST Framework 3.11 is now supported.
- Django REST Framework 3.7 is now supported.
- Django REST Framework 3.8 is now supported.
- Django REST Framework 3.9 is now supported.
- Python 3.4 is no longer supported. This is no longer supported upstream and most distributions provide a newer version.
- Django 1.8, 1.9 and 1.10 are no longer supported. These are no longer supported upstream and most distributions provide a newer version.
- djangorestframework 3.4 and 3.5 are no longer supported. These were only used with Django 1.8 to 1.10 and are not compatible with any version now supported by Patchwork.
- `pwclient` is no longer packaged with Patchwork. Instead, it is developed as a separate project on GitHub and available from PyPI.

19.5.3 Bug Fixes

- CVE-2019-13122 has been fixed. Andrew Donnellan discovered an XSS via the message-id field. A malicious user could send a patch with a message ID that included a script tag. Because of the quirks of the email RFCs, such a message ID can survive being sent through many mail systems, including Gmail, and be parsed and stored by Patchwork. When a user viewed a patch detail page for the patch with this message id, the script would be run. This is fixed by properly escaping the field before it is rendered.
- Queries to the REST API with filters are now significantly faster: slow database queries were reworked.
- To avoid triggering spam filters due to failed signature validation, many mailing lists mangle the From header to change the From address to be the address of the list, typically where the sender's domain has a strict DMARC policy enabled. This leads to incorrect senders being recorded. We now try to unmangle the From header using the X-Original-From or Reply-To headers, as used by Google Groups and Mailman respectively.
- Assigning maintained projects when creating a new user in the admin page was causing an error. This is now resolved.
- Long headers can be wrapped using CRLF followed by WSP (whitespace). This whitespace was not being stripped, resulting in errant whitespace being saved for the patch subject. This is resolved though existing patches and cover letters will need to be updated manually.
- An issue that resulted in checks for all patches being listed for each patch is resolved. (#203)
- An issue that prevented updating of delegates using the REST API is resolved. (#216)
- A project's `list_email`, `list_id` and `link_name` fields can no longer be updated via the REST API. This is a superuser-only operation that, for now, should only be done via the admin interface. (#217)
- It's now possible to assign patches to existing bundles from a user's TODO page. (#213)
- API resources with embedded series were not showing the `web_url` value for these series. This is now shown.
- Showing comments for a non-existant patch or cover letter was returning an empty response instead of a HTTP 404. This issue is resolved for both resources.
- Showing checks for a non-existant patch was returning an empty response instead of a HTTP 404. Similarly, attempting to create a new check against this patch would result in a HTTP 5xx error instead of a HTTP 404. Both issues are now resolved.
- Fields added in API v1.1 are now consistently excluded when requesting API v1.0, as was intended.
- #197 was the result of a issue with OzLabs instance and not Patchwork itself, and the fix included actually ended up corrupting subjects for everyone. It has now been reverted.
- The `pwclientrc` samples generated by Patchwork were previously not valid INI files. This issue is resolved. (#277)
- A bug that would result in patches from later series revisions being included in earlier revisions has been resolved.
- Previously, attempting to retrieve a patch that did not exist would result in a `HTTP 500` (Internal Server Error) being raised. This has been corrected and a `HTTP 404` (Not Found) is now raised instead.
- In the past, Patchwork used to support filtering patches that weren't delegated to anyone. This feature was removed in v1.1.0, as part of a patch designed to support delegation to anyone. However, that feature didn't scale and was later removed. The ability to delegate to anyone is now itself re-introduced.
- The delegate and submitter fields will remain populated when moving between different pages or changing filters. (#78)

19.5.4 API Changes

- Relations are available via `/patches/{patchID}/` endpoint, in the `related` field.
- Allow ordering events from the events API by date. This can be done by adding `order=date` or `order=-date` (the default) parameters.
- The `/event` API now exposes an `actor` attribute. It is possible to filter events by this attribute.
- The API version has been updated to v1.2.
- Projects now expose the `list_archive_url` and `list_archive_url_format` attributes.
- Patches, comments and cover letters now expose a `list_archive_url` attribute.
- The REST API now supports filtering patches by their hashes, using the `hash` query parameter.
- Bundles can now be created, updated and deleted via the REST API.

19.5.5 Security Notes

- Change the recommended method for generating the Django secret key to use a cryptographically secure random number generator.

19.5.6 Other Notes

- The performance of various pages has been improved with the addition of some database indexes and optimization of some queries.

CHAPTER

TWENTY

V2.1 SERIES ("EOLIENNE")

20.1 v2.1.6

20.1.1 Bug Fixes

- Queries to the REST API with filters are now significantly faster: slow database queries were reworked.
- An sql error was fixed in *lib/sql/grant-all.postgres.sql*.

20.2 v2.1.4

20.2.1 Bug Fixes

- CVE-2019-13122 has been fixed. Andrew Donnellan discovered an XSS via the message-id field. A malicious user could send a patch with a message ID that included a script tag. Because of the quirks of the email RFCs, such a message ID can survive being sent through many mail systems, including Gmail, and be parsed and stored by Patchwork. When a user viewed a patch detail page for the patch with this message id, the script would be run. This is fixed by properly escaping the field before it is rendered.
- The `pwclientrc` samples generated by Patchwork were previously not valid INI files. This issue is resolved. (#277)

20.3 v2.1.3

20.3.1 Bug Fixes

- #197 was the result of a issue with OzLabs instance and not Patchwork itself, and the fix included actually ended up corrupting subjects for everyone. It has now been reverted.
- In the past, Patchwork used to support filtering patches that weren't delegated to anyone. This feature was removed in v1.1.0, as part of a patch designed to support delegation to anyone. However, that feature didn't scale and was later removed. The ability to delegate to anyone is now itself re-introduced.

20.4 v2.1.2

20.4.1 Upgrade Notes

- django-filter 1.1 is now supported.
- Django REST Framework 3.7 is now supported.
- Django REST Framework 3.8 is now supported.
- Django REST Framework 3.9 is now supported.

20.4.2 Bug Fixes

- Assigning maintained projects when creating a new user in the admin page was causing an error. This is now resolved.
- Long headers can be wrapped using CRLF followed by WSP (whitespace). This whitespace was not being stripped, resulting in errant whitespace being saved for the patch subject. This is resolved though existing patches and cover letters will need to be updated manually.
- API resources with embedded series were not showing the `web_url` value for these series. This is now shown.
- Showing comments for a non-existant patch or cover letter was returning an empty response instead of a HTTP 404. This issue is resolved for both resources.
- Showing checks for a non-existant patch was returning an empty response instead of a HTTP 404. Similarly, attempting to create a new check against this patch would result in a HTTP 5xx error instead of a HTTP 404. Both issues are now resolved.
- Fields added in API v1.1 are now consistently excluded when requesting API v1.0, as was intended.

20.5 v2.1.1

20.5.1 Bug Fixes

- An issue that resulted in checks for all patches being listed for each patch is resolved. (#203)
- An issue that prevented updating of delegates using the REST API is resolved. (#216)
- A project's `list_email`, `list_id` and `link_name` fields can no longer be updated via the REST API. This is a superuser-only operation that, for now, should only be done via the admin interface. (#217)
- It's now possible to assign patches to existing bundles from a user's TODO page. (#213)
- The delegate and submitter fields will remain populated when moving between different pages or changing filters. (#78)

20.6 v2.1.0

20.6.1 Prelude

The key part of this release is a major performance fix - denormalising the project field into patch model so that counting a project's patches doesn't require a JOIN. This requires a migration and so isn't suitable for a stable backport. Event listing in the API has also been sped up by refactoring the queries.

This release also includes the feature development that had accrued in the mean time and numerous bug fixes.

The REST API version has been bumped to 1.1.

20.6.2 New Features

- Django 1.11 is now supported.
- Allow list filtering into multiple projects (and email dropping) based on subject prefixes. Enable by specifying a regular expression which needs to be matched in the subject on a per-project basis (field `subject_match`). Project with empty `subject_match` field (and matching `list_id`) serves as a default in case of no match.
- The `pwclient get` command will now download patches with a `.patch` extension.
- Python 3.6 is now supported.

20.6.3 Known Issues

- Series parsing in the presence of parallel mail processing is still unreliable.
- Several more minor issues can be browsed on our issue tracker.

20.6.4 Upgrade Notes

- Django 1.6 and 1.7 are no longer supported. These are no longer supported upstream and most distributions provide a newer version.
- django-filter 0.11 is no longer supported. This was only used with Django 1.6 and 1.7 and is not compatible with any version supported by Patchwork.

20.6.5 Bug Fixes

- If a patch was processed by Patchwork before series support was added, it will not have a series associated with it. As a result, it is not possible to extract the dependencies for that patch from the series. This was not previously handled correctly. A 404 is now raised if this occurs.
- A nasty race condition bug that could cause patches in a series to be dropped has been fixed.
- The `parsemail.sh` and `parsemail-batch.sh` scripts, found in `patchwork/bin`, will now default to using `python` rather than `python2` for calling `manage.py`. This resolves an issue when Patchwork is deployed with a virtualenv.

20.6.6 API Changes

- Links to related comments are now exposed when checking patch and cover letter details. The comments themselves are then available via `/patches/{patchID}/comments` and `/covers/{coverID}/comments` endpoints. Please note that comments are available only since API version 1.1
- Cover letters embedded in other responses now provide an mbox link, which can be used to download the cover letter and associated metadata (tags) in mbox format.
- Series, patches and cover letters can be filtered by submitter using email addresses. For example:

```
$ curl /covers/?submitter=stephen@that.guru
```

- Bundles can be filtered by owner, patches by delegate and checks by user using username. For example:

```
$ curl /bundles/?owner=stephenfin
```

- Filters can now be specified multiple times. For example:

```
$ curl /patches/?state=under-review&state=rfc
```

  This operates as a logical OR: it will retrieve patches that are either Under Review or RFC.

- The `/project` endpoint now exposes a `subject_match` attribute.
- Messages headers that use the same key, such as `Received:` are now combined into a list. Previously only one of the values would be output. This affects the `/covers` and `/patches` endpoints.

20.6.7 Other Notes

- The patch ID on the patch detail page can now be clicked to copy it. This is similar to what we already do on the patch list page.
- mbox files now contain all headers from the original email. This also means the `Subject:` header included will contain the original subject and not the parsed Patchwork's version.
- Unify timezones used – use UTC for both email submissions and internal events. Please note that this change doesn't modify already existing data so in case the instance's timezone is UTC+XX, events will appear out of order (as if they happened earlier) for XX hours in the events API feed.

CHAPTER

TWENTYONE

V2.0 SERIES (“DAZZLE”)

21.1 v2.0.4

21.1.1 Bug Fixes

- CVE-2019-13122 has been fixed. Andrew Donnellan discovered an XSS via the message-id field. A malicious user could send a patch with a message ID that included a script tag. Because of the quirks of the email RFCs, such a message ID can survive being sent through many mail systems, including Gmail, and be parsed and stored by Patchwork. When a user viewed a patch detail page for the patch with this message id, the script would be run. This is fixed by properly escaping the field before it is rendered.

21.2 v2.0.3

21.2.1 Bug Fixes

- If a patch was processed by Patchwork before series support was added, it will not have a series associated with it. As a result, it is not possible to extract the dependencies for that patch from the series. This was not previously handled correctly. A 404 is now raised if this occurs.
- The `parsemail.sh` and `parsemail-batch.sh` scripts, found in `patchwork/bin`, will now default to using `python` rather than `python2` for calling `manage.py`. This resolves an issue when Patchwork is deployed with a virtualenv.

21.3 v2.0.2

21.3.1 Bug Fixes

- Resolve some issues caused by parallel parsing of series.
- Poorly formatted email headers are now handled correctly.
- Patches with CRLF newlines are now parsed correctly and these line endings are stripped when saving patches.
- Resolved some issues with pagination.
- Emails from *git-pull-request* v2.14.3+ are now handled correctly.
- Token generation from the web UI is now disabled if the REST API is disabled. This was causing an exception.
- Non-breaking spaces in tags are now handled correctly.

- Patches with no space before the series marker, such as `PATCH1/8`, are now parsed correctly.

21.4 v2.0.1

21.4.1 Bug Fixes

- Handle requests for pages out of range.
- Fix SQL permissions scripts for tables and columns added in 2.0.
- Fix filtering of projects by name
- Fix "add to bundle" dropdown
- Performance improvements for the XML-RPC API

21.5 v2.0.0

21.5.1 Prelude

The v2.0.0 release includes many new features and bug fixes. For full information on the options avaiable, you should look at the full release notes in detail. However, there are two key features that make v2.0.0 a worthwhile upgrade:

- A REST API is now provided, which will eventually replace the legacy XML-RPC API
- Patch series and series cover letters are now supported

For further information on these features and the other changes in this release, review the full release notes.

21.5.2 New Features

- REST API.

  Previous versions of Patchwork provided an XML-RPC API. This was functional but there were a couple of issues around usability and general design. This API also provided basic versioning information but the existing clients, mostly *pwclient* variants, did not validate this version. Together, this left us with an API that needed work but no way to fix it without breaking every client out there.

  Rather than breaking all those users, make a clean break and provide another API method. REST APIs are the API method de jour providing a number of advantages over XML-RPC APIs, thus, a REST API is chosen. The following resources are exposed over this new API:

  - Bundles
  - Checks
  - Projects
  - People
  - Users
  - Patches
  - Series
  - Cover letters

For information on the usage of the API, refer to the documentation.

- Cover letters are now supported.

  Cover letters are often sent in addition to a series of patches. They do not contain a diff and can generally be identified as number 0 of a series. For example:

```
[PATCH 0/3] A cover letter
```

  Cover letters contain useful information that should not be discarded. Both cover letters and replies to these mails are now stored for use with series.

- Series are now supported.

  Series are groups of patches sent as one bundle. For example:

```
[PATCH 0/3] A cover letter
  [PATCH 1/3] The first patch
  [PATCH 2/3] The second patch
  [PATCH 3/3] The third patch
```

  While Patchwork already supports bundles, these must be created manually, defeating the purpose of using series in the first place. Series make use of the information provided in the emails themselves, avoiding this manual step. The series support implemented is basic and does not support versioning. This will be added in a future release.

- All comments now have a permalink which can be used to reference individual replies to patches and cover letters.
- Django Debug Toolbar is now enabled by defaut when using development settings.
- Django 1.9 and 1.10 are now supported.
- Python 3.5 is now supported.
- Docker support is now integrated for development usage. To use this, refer to the documentation.
- Series markers are now parsed from patches generated by the Mercurial Patchbomb extension.

21.5.3 Upgrade Notes

- The REST API is enabled by default.

  The REST API is enabled by default. It is possible to disable this API, though this functionality may be removed in a future release. Should you wish to disable this feature, configure the `ENABLE_REST_API` setting to `False`.

- The `parsemail.py` and `parsearchive.py` scripts have been replaced by the `parsemail` and `parsearchive` management commands. These can be called like any other management commands. For example:

```
$ ./manage.py parsemail [args...]
```

- The `DEFAULT_PATCHES_PER_PAGE` has been renamed as `DEFAULT_ITEMS_PER_PAGE` as it is now possible to list cover letters in addition to patches.
- The `context` field for patch checks must now be slug, or a string consisting of only ASCII letters, numbers, underscores or hyphens. While older, non-slugified strings won't cause issues, any scripts creating contexts must be updated where necessary.

21.5.4 Bug Fixes

- When downloading an mbox, a user's name will now be set to the name used in the last email recieved from them. Previously, the name used in the first email received from a user was used.
- *user at domain*-style email addresses, commonly found in Mailman archives, are now handled correctly.
- Unicode characters transmitted over the XML-RPC API are now handled correctly under Python 3
- The *pwclient* tool will no longer attempt to re-encode unicode to ascii bytes, which was a frequent cause of `UnicodeEncodeError` exceptions. Instead, a warning is produced if your environement is not configured for unicode.

21.5.5 Other Notes

- reno is now used for release note management.
- Patch diffs now download with a `diff` extension.

CHAPTER

TWENTYTWO

V1.1 SERIES ("CASHMERE")

22.1 1.1.3

This release fixes a number of issues with the 1.1.2 release.

22.1.1 Bug Fixes

- Some Python 3 issues are resolved in *pwclient*
- *pwclient* now functions as expected behind a proxy

22.2 1.1.2

This release fixed a number of issues with the 1.1.1 release.

22.2.1 Bug Fixes

- Headers containing invalid characters or codings are now parsed correctly
- Patches can no longer be delegated to any user

  This had significant performance impacts and has been reverted.

22.3 1.1.1

This release fixed a number of issues with the 1.1.0 release.

22.3.1 Bug Fixes

- Numerous issues in the *parsemail.py*, *parsearchive.py* and *parsemail.sh* scripts are resolved
- Permissions of database tables, as set by *grant-all* SQL scripts, are now set for tables added in Patchwork 1.1.0
- Some performance and usability regressions in the UI are resolved

22.4 1.1.0

This release focuses on usability and maintainability, and sets us up nicely for a v2.0.0 release in the near future. Feature highlights of v1.1.0 include:

- Automated delegation of patches, based on the files modified in said patches.
- Storing of test results, a.k.a. "checks", on a patch-by-patch basis.
- Delegation of patches to any registered Patchwork user (previously one had to be a registered maintainer).
- Overhaul of the web UI, which is now based on Bootstrap.
- Python 3 support.

22.4.1 New Features

- The web UI is updated to reflect modern web standards. Bootstrap 3.x is used.
- Python 3.4 is now supported
- Checks, which can be used to report the status of tests, have been added
- Automatic delegation of patches based on file path
- Automated documentation for the XML-RPC API. This can be found at the '/xmlrpc' in most Patchwork deployments
- Vagrant is now integrated for use during development

22.4.2 Upgrade Notes

- Patches can now be delegated to any Patchwork user.

CHAPTER

TWENTYTHREE

V1.0 SERIES (“BURLAP”)

23.1 1.0.0

This release changes a few admin-visible components of Patchwork, so upgrading involves a few steps.

23.1.1 New Features

- Patch tags are now supported

  Patch “tags”, such as *Acked-by*, *Reviewed-by*, are typically included in patches and replies. They provide important information as to the activity and “mergability” of a patch. These tags are now extracted from patches and included in the patch list.

- Django 1.7 and Django 1.8 are now supported
- tox support is integrated for use by developers

23.1.2 Upgrade Notes

- Migrations are now executed using the Django migrations framework.

  Future database migrations will be implemented using Django Migrations, rather than raw SQL scripts. Before switching to Django migrations, first apply any unapplied migrations in the *lib/sql/migration* folder. For example, on postgres:

  ```
  $ psql -f lib/sql/migration/015-add-patch-tags.sql patchwork
  $ psql -f lib/sql/grant-all.postgres.sql patchwork
  ```

  Once applied, configure the required Django Migration tables using the *migrate* managment command:

  ```
  $ ./manage.py migrate --fake-initial
  ```

- Moved Patchwork source from the *apps* directory to the top level directory.

  Any scripts or tools that call Patchwork applications, such as *parsemail.sh*, must be updated to reference the new location of these scripts. To do this, simply remove *apps/* from the path, i.e. *apps/patchwork/* becomes *patchwork*.

- The *patchwork-cron.py* script has been replaced by the *cron* management command.

  Any references to the former should be updated to the latter. The *cron* management command can be called like so:

```
$ ./manage.py cron
```

- The *settings.py* file has been updated to reflect modern Django practices.

  You may need to manually migrate your existing configuration to the new settings file(s). By default, settings are read from *patchwork/settings/production.py*. To migrate, use the provided template:

  ```
  $ cp patchwork/settings/production{.example,}.py
  ```

  Merge your previous settings, usually located in *apps/local_settings.py*, to this file.

  In addition, any scripts that set the *DJANGO_SETTINGS_MODULE* environment variable will need to be updated to reflect the new location, typically:

  ```
  DJANGO_SETTINGS_MODULE=patchwork.settings.production
  ```

- Django *staticfiles* is now used to to gather static files for for serving via a web server

  Static content should now be located in the folder indicated by *STATIC_ROOT*. This should point to somewhere sensible, such as the absolute path of *htdocs/static* in the Patchwork tree. Configure the *STATIC_ROOT* setting in your settings file, then run the *collectstatic* management command:

  ```
  $ ./manage.py collectstatic
  ```

  Finally, update your webserver configuration to serve the static content from this new location. Refer to the sample web configuration files provided in *lib* for more information.

- Django 1.5 is no longer supported
- Python 2.5 support was broken and is officially no longer supported

23.1.3 Deprecation Notes

- Django 1.6 support will be removed in a future release
- Raw SQL migration scripts, currently found at *lib/sql/migration*, will no longer be updated and will be removed in a future release. The Django Migration framework, found in Django 1.7 and above, should be used instead.

CHAPTER

TWENTYFOUR

V0.9 SERIES (“ALPACA”)

This represents the state of the Patchwork code before adopting semantic versioning, along with fabric-inspired release names. For information on the features available in this release, refer to the git logs.

HTTP ROUTING TABLE

/api

INDEX

Symbols

I

M

P