git-pw

Release 2.2.0

Oct 01, 2021

Contents

CHAPTER 1

git-pw (Patchwork subcommand for Git)

1.1 Overview

git-pw is a tool for integrating Git with Patchwork, the web-based patch tracking system.

Important: *git-pw* only supports Patchwork 2.0+ and REST API support must be enabled on the server end. You can check for support by browsing `/about` for your given instance. If this page returns a 404, you are using Patchwork < 2.0.

The pwclient utility can be used to interact with older Patchwork instances or instances with the REST API disabled.

1.2 Installation

The easiest way to install *git-pw* and its dependencies is using `pip`. To do so, run:

```
$ pip install git-pw
```

You can also install *git-pw* manually. First, install the required dependencies. On Fedora, run:

```
$ sudo dnf install python3-requests python3-click python3-pbr \
    python3-arrow python3-tabulate python3-yaml
```

On Ubuntu, run:

```
$ sudo apt-get install python3-requests python3-click python3-pbr \
    python3-arrow python3-tabulate python3-yaml
```

Once dependencies are installed, clone this repo and run `setup.py`:

```
$ git clone https://github.com/getpatchwork/git-pw
$ cd git-pw
$ pip install --user .  # or 'sudo python setup.py install'
```

1.3 Getting Started

To begin, you'll need to configure Git settings appropriately. The following settings are **required**:

pw.server The URL for the Patchwork instance's API. This should include the API version:

```
https://patchwork.ozlabs.org/api/1.2
```

You can discover the API version supported by your instance by comparing the server version, found at `/about`, with the API versions provided in the documentation. For example, if your server is running Patchwork version 3.0.x, you should use API version 1.2.

pw.project The project name or list-id. This will appear in the URL when using the web UI:

```
https://patchwork.ozlabs.org/project/{project_name}/list/
```

For read-write access, you also need authentication - you can use either API tokens or a username/password combination:

pw.token The API token for your Patchwork account.

pw.username The username for your Patchwork account.

pw.password The password for your Patchwork account.

If only read-only access is desired, credentials can be omitted.

The following settings are **optional** and may need to be set depending on your Patchwork instance's configuration:

pw.states The states that can be applied to a patch using the `git pw patch update` command. Should be provided in slug form (`changes-requested` instead of `Changes Requested`). Only required if your Patchwork instance uses non-default states.

You can set these settings using the `git config` command. This should be done in the repo in which you intend to apply patches. For example, to configure the Patchwork project, run:

```
$ git config pw.server 'https://patchwork.ozlabs.org/api/1.1/'
$ git config pw.project 'patchwork'
```

1.4 Development

If you're interested in contributing to *git-pw*, first clone the repo:

```
$ git clone https://github.com/getpatchwork/git-pw
$ cd git-pw
```

Create a *virtualenv*, then install the package in editable mode:

```
$ virtualenv .venv
$ source .venv/bin/activate
$ pip install --editable .
```

1.5 Usage

See *Usage*.

1.6 Release Notes

See *Release Notes*.

CHAPTER 2

Usage

2.1 git-pw

git-pw is a tool for integrating Git with Patchwork.

git-pw can interact with individual patches, complete patch series, and customized bundles. The three major subcommands are *patch*, *bundle*, and *series*.

The git-pw utility is a wrapper which makes REST calls to the Patchwork service. To use git-pw, you must set up your environment by configuring your Patchwork server URL and either an API token or a username and password. To configure the server URL, run:

```
git config pw.server http://pw.server.com/path/to/patchwork
```

To configure the token, run:

```
git config pw.token token
```

Alternatively, you can pass these options via command line parameters or environment variables.

For more information on any of the commands, simply pass `--help` to the appropriate command.

```
git-pw [OPTIONS] COMMAND [ARGS]...
```

Options

--debug
Output more information about what's going on.

--token <TOKEN>
Authentication token. Defaults to the value of 'git config pw.token'.

--username <USERNAME>
Authentication username. Defaults to the value of 'git config pw.username'.

--password <PASSWORD>
Authentication password. Defaults to the value of 'git config pw.password'.

--server <SERVER>
Patchwork server address/hostname. Defaults to the value of 'git config pw.server'.

--project <PROJECT>
Patchwork project. Defaults the value of 'git config pw.project'.

--version
Show the version and exit.

Environment variables

PW_TOKEN

Provide a default for *--token*

PW_USERNAME

Provide a default for *--username*

PW_PASSWORD

Provide a default for *--password*

PW_SERVER

Provide a default for *--server*

PW_PROJECT

Provide a default for *--project*

2.1.1 bundle

Interact with 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.

```
git-pw bundle [OPTIONS] COMMAND [ARGS]...
```

add

Add one or more patches to a bundle.

Append the provided PATCH_IDS to bundle BUNDLE_ID.

Requires API version 1.2 or greater.

```
git-pw bundle add [OPTIONS] BUNDLE_ID PATCH_IDS...
```

Options

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

BUNDLE_ID
Required argument

PATCH_IDS
Required argument(s)

apply

Apply bundle.

Apply a bundle locally using the 'git-am' command. Any additional ARGS provided will be passed to the 'git-am' command.

```
git-pw bundle apply [OPTIONS] BUNDLE_ID [ARGS]...
```

Arguments

BUNDLE_ID
Required argument

ARGS
Optional argument(s)

create

Create a bundle.

Create a bundle with the given NAME and patches from PATCH_ID.

Requires API version 1.2 or greater.

```
git-pw bundle create [OPTIONS] NAME PATCH_IDS...
```

Options

--public, --private
Allow other users to view this bundle. If private, only you will be able to see this bundle.

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

NAME
Required argument

PATCH_IDS
Required argument(s)

delete

Delete a bundle.

Delete bundle BUNDLE_ID.

Requires API version 1.2 or greater.

```
git-pw bundle delete [OPTIONS] BUNDLE_ID
```

Options

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

BUNDLE_ID
Required argument

download

Download bundle in mbox format.

Download a bundle but do not apply it. `OUTPUT` is optional and can be an output full file path or a directory or `-` to output to `stdout`. If `OUTPUT` is not provided, the output path will be automatically chosen.

```
git-pw bundle download [OPTIONS] BUNDLE_ID [OUTPUT]
```

Arguments

BUNDLE_ID
Required argument

OUTPUT
Optional argument

list

List bundles.

List bundles on the Patchwork instance.

```
git-pw bundle list [OPTIONS] [NAME]
```

Options

--owner <OWNER>
Show only bundles with these owners. Should be an email, name or ID. Private bundles of other users will not be shown.

--sort <FIELD>
Sort output on given field.

Options id | -id | name | -name

--page <PAGE>
Page to retrieve items from. This is influenced by the size of LIMIT.

--limit <LIMIT>
Maximum number of items to show.

-c, --column <COLUMN>
Columns to be included in output.

Options ID | Name | Owner | Public

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

NAME
Optional argument

remove

Remove one or more patches from a bundle.

Remove the provided PATCH_IDS to bundle BUNDLE_ID.

Requires API version 1.2 or greater.

```
git-pw bundle remove [OPTIONS] BUNDLE_ID PATCH_IDS...
```

Options

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

BUNDLE_ID
Required argument

PATCH_IDS
Required argument(s)

show

Show information about bundle.

Retrieve Patchwork metadata for a bundle.

```
git-pw bundle show [OPTIONS] BUNDLE_ID
```

Options

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

BUNDLE_ID
Required argument

update

Update a bundle.

Update bundle BUNDLE_ID. If PATCH_IDs are specified, this will overwrite all patches in the bundle. Use 'bundle add' and 'bundle remove' to add or remove patches.

Requires API version 1.2 or greater.

```
git-pw bundle update [OPTIONS] BUNDLE_ID
```

Options

--name <name>

--patch <patch_ids>
Add the specified patch(es) to the bundle.

--public, --private
Allow other users to view this bundle. If private, only you will be able to see this bundle.

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

BUNDLE_ID
Required argument

2.1.2 patch

Interact with 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.

```
git-pw patch [OPTIONS] COMMAND [ARGS]...
```

apply

Apply patch.

Apply a patch locally using the 'git-am' command. Any additional ARGS provided will be passed to the 'git-am' command.

```
git-pw patch apply [OPTIONS] PATCH_ID [ARGS]...
```

Options

--series <SERIES>
Series to include dependencies from. Defaults to latest.

--deps, --no-deps
When applying the patch, include dependencies if available. Defaults to using the most recent series.

Arguments

PATCH_ID
Required argument

ARGS
Optional argument(s)

download

Download patch in diff or mbox format.

Download a patch but do not apply it. `OUTPUT` is optional and can be an output file path or a directory or - to output to `stdout`. If `OUTPUT` is not provided, the output path will be automatically chosen.

```
git-pw patch download [OPTIONS] PATCH_ID [OUTPUT]
```

Options

--diff
Show patch in diff format.

--mbox
Show patch in mbox format.

Arguments

PATCH_ID
Required argument

OUTPUT
Optional argument

list

List patches.

List patches on the Patchwork instance.

```
git-pw patch list [OPTIONS] [NAME]
```

Options

--state <STATE>
Show only patches matching these states. Should be slugified representations of states. The available states are instance dependant.

--submitter <SUBMITTER>
Show only patches by these submitters. Should be an email, name or ID.

--delegate <DELEGATE>
Show only patches with these delegates. Should be an email or username.

--hash <HASH>
Show only patches with these hashes.

--archived
Include patches that are archived.

--sort <FIELD>
Sort output on given field.

Options id | -id | name | -name | date | -date

--page <PAGE>
Page to retrieve items from. This is influenced by the size of LIMIT.

--limit <LIMIT>
Maximum number of items to show.

-c, --column <COLUMN>
Columns to be included in output.

Options ID | Date | Name | Submitter | State | Archived | Delegate

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

NAME
Optional argument

show

Show information about patch.

Retrieve Patchwork metadata for a patch.

```
git-pw patch show [OPTIONS] PATCH_ID
```

Options

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

PATCH_ID
Required argument

update

Update one or more patches.

Updates one or more Patches on the Patchwork instance. Some operations may require admin or maintainer permissions.

```
git-pw patch update [OPTIONS] PATCH_IDS...
```

Options

--commit-ref <COMMIT_REF>
Set the patch commit reference hash

--state <STATE>
Set the patch state. Should be a slugified representation of a state. The available states are instance dependant and can be configured using 'git config pw.states'.

Options new | under-review | accepted | rejected | rfc | not-applicable | changes-requested | awaiting-upstream | superseded | deferred

--delegate <DELEGATE>
Set the patch delegate. Should be unique user identifier: either a username or a user's email address.

--archived <ARCHIVED>
Set the patch archived state.

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

PATCH_IDS
Required argument(s)

2.1.3 series

Interact with 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.

```
git-pw series [OPTIONS] COMMAND [ARGS]...
```

apply

Apply series.

Apply a series locally using the 'git-am' command. Any additional ARGS provided will be passed to the 'git-am' command.

```
git-pw series apply [OPTIONS] SERIES_ID [ARGS]...
```

Arguments

SERIES_ID
Required argument

ARGS
Optional argument(s)

download

Download series in mbox format.

Download a series but do not apply it. `OUTPUT` is optional and can be an output path or `-` to output to `stdout`. If `OUTPUT` is not provided, the output path will be automatically chosen.

```
git-pw series download [OPTIONS] SERIES_ID [OUTPUT]
```

Options

--separate
Download each series patch to a separate file

--combined
Download all series patches to one file

Arguments

SERIES_ID
Required argument

OUTPUT
Optional argument

list

List series.

List series on the Patchwork instance.

```
git-pw series list [OPTIONS] [NAME]
```

Options

--submitter <SUBMITTER>
Show only series by these submitters. Should be an email, name or ID.

--sort <FIELD>
Sort output on given field.

Options id | -id | name | -name | date | -date

--page <PAGE>
Page to retrieve items from. This is influenced by the size of LIMIT.

--limit <LIMIT>
Maximum number of items to show.

-c, --column <COLUMN>
Columns to be included in output.

Options ID | Date | Name | Version | Submitter

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

NAME
Optional argument

show

Show information about series.

Retrieve Patchwork metadata for a series.

```
git-pw series show [OPTIONS] SERIES_ID
```

Options

-f, --format <fmt>
Output format. Defaults to the value of 'git config pw.format' else 'table'.

Options simple | table | csv | yaml

Arguments

SERIES_ID
Required argument

CHAPTER 3

Release Notes

3.1 2.2.0

3.1.1 New Features

- The `-f` / `--format` option used for many `list` commands now supports a new option: `yaml`. This can be useful in environments where limited horizontal space is available.

3.2 2.1.0

3.2.1 New Features

- The `series download` command now accepts an optional toggle flag pair, `--separate` / `--combine`, to either download series patches to separate files or to a combined mbox file (default).
- The `patch download`, `bundle download`, and `series download` commands now accept a directory for the `OUTPUT` argument as well as a file. If a directory is provided, the file will use the name provided by Patchwork but will be downloaded to the specified directory rather than the current working directory.
- It is no longer necessary to provide credentials when interacting with read-only APIs. You will continue to be prompted for credentials for write access.

3.3 2.0.0

3.3.1 New Features

- It is now possible to filter patches by hash. For example:

```
git pw patch list --hash HASH
```

3.4 1.9.0

3.4.1 New Features

- The following `bundle` commands have been added:
  - `bundle create`
  - `bundle update`
  - `bundle delete`
  - `bundle add`
  - `bundle remove`

  Together, these allow for creation, modification and deletion of bundles. Bundles are custom, user-defined groups of patches that can be used to keep patch lists, preserving order, for future inclusion in a tree.

  These commands require API v1.2.

3.4.2 Other Notes

- *git-pw* 1.9.0 will be the last version to support Python 2.7. *git-pw* 2.0.0 will require Python 3.5 or greater.

3.5 1.8.0

3.5.1 New Features

- The `--state` option of the `git pw patch update` command now supports auto-complete for the default set of states provided by Patchwork. If necessary, these states can be overridden using the `pw.states` `git config` setting.

3.6 1.7.0

3.6.1 Upgrade Notes

- *git-pw* no longer officially supports PyPy.
- CSV-formatted output is now quoted by default.

3.6.2 Bug Fixes

- Resolved an issue that prevented viewing patch diffs/mboxes in stdout on Python 3.
- An info-level log is now correctly skipped when downloading patches, bundles or series to `STDOUT` on Python 3.

- An issue with the unicode data when using the CSV format on Python 2.7 has been resolved.

3.7 1.6.0

3.7.1 Upgrade Notes

- Support for Python 3.4 has been dropped.

3.8 1.5.2

3.8.1 Upgrade Notes

- Downloaded patches, series and bundles are now saved to a temporary directory instead of the current directory.

3.8.2 Bug Fixes

- An issue that resulted in invalid output on Python 3 when a patch or series was not successfully applied has been resolved.

3.9 1.5.0

3.9.1 New Features

- Many commands now take a `--format`/`-f` parameter, which can be used to control the output format. Three formats are currently supported:
  - `table` (default)
  - `simple` (a version of table with less markup)
  - `csv` (comma-separated output)
- All list comands now take a `--column`/`-c` parameter, which can be used to control what columns are output. This can be specified multiples times. All columns are output by default.

3.10 1.4.0

3.10.1 New Features

- It is now possible to filter patches, bundles and series and the IDs of users that submitted or are delegated to the item in question. For example:

```
$ git pw patch list --submitter 1
```

- Filtering patches, bundles and series by user will now only display a warning if multiple matches are found for a given filter and you're using API v1.0. Previously this would have been an unconditional error.

3.10.2 Bug Fixes

- Resolve an issue that prevented the following filtering when using API v1.1:
  - Filter patches or series by submitter using the submitter's name
  - Filter patches by delegate using the delegate's email
  - Filter bundles by owner using the owner's email

3.11 1.3.0

3.11.1 New Features

- *git-pw* will now choose use the same rules as Git to select the pager used for `list` commands. This means the pager will be chosen based on a variety of environment variables and git config options:

  The order of preference is the `$GIT_PAGER` environment variable, then `core.pager` configuration, then `$PAGER`, and then the default chosen at compile time (usually `less`)

3.11.2 Upgrade Notes

- `git pw patch apply`, `git pw bundle apply` and `git pw series apply` will now pass any additional arugments provided through to `git am`. For example:

```
$ git pw patch apply 123 --signoff
```

  Previously it was necessary to escape these arguments with `--`.

- A warning is now raised when using multiple filters (e.g. `git pw bundle --owner foo --owner bar`) with Patchwork API v1.0. This is not supported and will result in confusing results.

3.12 1.2.0

3.12.1 New Features

- Patchwork API v1.1 is now supported.
- Patches, series and bundles downloaded using the `download` command will now be saved to a file by default instead of output to stdout. By default, this will use the name provided by Patchwork but you can override this by passing a specific filename. For example:

```
$ git pw patch download 1234 hello-world.patch
```

  You can also output to `stdout` using the `-` shortcut. For example:

```
$ git pw patch download 1234 -
```

- You can now list multiple patches for `git pw patch update`. This allows you to do bulk updates, e.g. for a series of patches (because series states are not yet supported). For example:

```
$ git pw patch update --state accepted 123 124 125 126
```

- It's now possible to use a bundle name in addition to the numeric ID for the `bundle download`, `bundle apply` and `bundle show` commands. For example:

```
$ git pw bundle show 'My sample bundle'
```

  As bundle names are not necessarily unique, this will fail if multiple bundles match the provided string.

3.12.2 Upgrade Notes

- Configuring a server without the API version, e.g. via `git config pw.server` will now result in a warning. An error will be raised in a future release of *git-pw*.
- Patches, series and bundles downloaded using the `download` command will now be saved to a file by default. To continue outputing to `stdout`, use `-`. For example:

```
$ git pw patch download 1234 -
```

3.12.3 Bug Fixes

- HTTP 404 and HTTP 5xx errors are now handled correctly.

3.13 1.1.2

3.13.1 Bug Fixes

- Patches are now automatically filtered by `new` state, as originally intended. You can override this by specifying the `--state` filter.
- Some issues with filtering patches, series and bundles by submitters and delegates, submitters, and owners respectively are resolved.
- API errors are now handled correctly.

3.14 1.1.0

3.14.1 Upgrade Notes

- Project configuration, e.g. via `git config pw.project`, is now required. Previously, not configuring a project would result in items for for all projects being listed. This did not scale for larger instances such as *patchwork.ozlabs.org* and proved confusing for some users. If this functionality is required, you must explicitly request it using the `*` character, e.g. `git pw patch list --project='*'`.

3.14.2 Bug Fixes

- The `git pw {patch,series,bundle} apply` commands will now save the downloaded patches before applying them. This avoids ascii/unicode issues on different versions of Python and avoids the need to load the entire patch into memory.

3.14.3 Other Notes

- Python 3.2 and 3.3 are no longer supported as both versions are EOL.

3.15 1.0.0

3.15.1 Prelude

Initial release of *git-pw* using the new REST API provided in Patchwork 2.0

Index

Symbols

A

B

G

N

O

P

S