jirahub

Release 0.3.0

STScI

Feb 24, 2021

CONTENTS:

Jirahub provides a configurable tool for synchronization of issues between a GitHub repository and a JIRA project. With it, you can use GitHub for coding and ticket tracking while using JIRA for ticket tracking and project management.

CHAPTER

ONE

DOWNLOAD AND INSTALL

To download and install:

```
$ pip install jirahub
```

The package's sole requirements are PyGithub and JIRA. Both of these dependencies are installable via pip.

CHAPTER

TWO

JIRA CONFIGURATION

jirahub stores state on the JIRA issue in two custom fields, which you (or your JIRA administrator) will need to create. The first field stores the URL of a linked GitHub issue, and should be type "URL Field". The second stores a JSON object containing general jirahub metadata, and should be type "Text field (multi-line)".

CHAPTER THREE

JIRAHUB CONFIGURATION

Jirahub configuration is divided between environment variables (JIRA and GitHub credentials) and one or more .py files (all other parameters).

3.1 Environment variables

Your JIRA and GitHub credentials are provided to jirahub via environment variables:

Variable name	Description
JIRAHUB_JIRA_USERNAME	JIRA username of your jirahub bot
JIRAHUB_JIRA_PASSWORD	JIRA password of your jirahub bot
JIRAHUB_GITHUB_TOKEN	GitHub API token of your jirahub bot

3.2 Configuration file

The remaining parameters are specified in a Python configuration file. There are few required parameters, but jirahub takes no actions by default, so users must explicitly enable features that they wish to use. The *generate-config* command can be used to create an initial configuration file. The file is executed with the `c` variable bound to an instance of `jirahub.config.JirahubConfig`, which has two attributes, `jira` and `github`.

3.2.1 jira

These are parameters particular to JIRA. The `server` and `project_key` attributes are required.

Name	Description
c.jira.server	The URL of your JIRA server (e.g., https://my-jira.example.com)
c.jira.project_key	The project key of the JIRA project that will be synced
c.jira.github_issue_url_field_id	The integer ID of a JIRA custom field in which jirahub will write the URL of the linked GitHub issue.
c.jira.jirahub_metadata_field_id	The integer ID of a JIRA custom field in which jirahub will write metadata such as the ids of linked comments.
c.jira.closed_statuses	List of JIRA statuses that will be considered closed. All others will be treated as open, for the purposes of syncing GitHub open/closed status and filtering issues. These values are case-insensitive.
c.jira.close_status	JIRA status set on an issue when closed by the bot
c.jira.reopen_status	JIRA status set on an issue when re-opened by the bot
c.jira.open_status	JIRA status set on a newly created issue. Set to None to use your project's default for new issues.
c.jira.max_retries	Maximum number of retries on request failure
c.jira.notify_watchers	Set to `True` if watchers should be notified when an issue is updated by the bot
c.jira.sync_comments	Set to `True` if JIRA comments should be created from GitHub comments
c.jira.sync_status	Set to `True` if the JIRA issue status should be set based on the GitHub open/closed status
c.jira.sync_labels	Set to `True` if the JIRA issue's labels should match GitHub's labels
c.jira.sync_milestones	Set to `True` if the JIRA issue's fixVersions field should match GitHub's milestone
c.jira.create_tracking_comment	Set to `True` to create a comment on JIRA issues that links back to GitHub.
c.jira.redact_patterns	List of `re.Pattern` whose matches will be redacted from issue titles, issue bodies, and comment bodies copied over from GitHub
c.jira.issue_title_formatter	Callable that transforms the GitHub issue title before creating/updating it in JIRA. See *Custom formatters* for further detail.
c.jira.issue_body_formatter	Callable that transforms the GitHub issue body before creating/updating it in JIRA. See *Custom formatters* for further detail.
c.jira.comment_body_formatter	Callable that transforms the GitHub comment body before creating/updating it in JIRA. See *Custom formatters* for further detail.
c.jira.issue_filter	Callable that selects GitHub issues that will be created in JIRA. See *Issue filters* for further detail.
c.jira.before_issue_create	List of callables that transform the fields used to create a new JIRA issue. This can (for example) be used to override jirahub's behavior, or set values for arbitrary custom fields. See *Issue hooks* for further detail.

3.2.2 github

These are parameters particular to GitHub. The `repository` parameter is required.

Name	Description
c.github.repository	GitHub repository name with organization, e.g., spacetelescope/jwst
c.github.max_retries	Maximum number of retries on request failure
c.github.sync_comments	Set to `True` if GitHub comments should be created from JIRA comments
c.github.sync_status	Set to `True` if the GitHub issue status should be set based on the JIRA open/closed status
c.github.sync_labels	Set to `True` if the GitHub issue's labels should match JIRA's labels
c.github.sync_milestones	Set to `True` if the GitHub issue's fixVersions field should match JIRA's milestone
c.jira.create_tracking_comment	Set to `True` to create a comment on GitHub issues that links back to JIRA.
c.github.redact_patterns	List of `re.Pattern` whose matches will be redacted from issue titles, issue bodies, and comment bodies copied over from JIRA
c.github.issue_title_formatter	Callable that transforms the JIRA issue title before creating/updating it in GitHub. See *Custom formatters* for further detail.
c.github.issue_body_formatter	Callable that transforms the JIRA issue body before creating/updating it in GitHub. See *Custom formatters* for further detail.
c.github.comment_body_formatter	Callable that transforms the JIRA comment body before creating/updating it in GitHub. See *Custom formatters* for further detail.
c.github.issue_filter	Callable that selects JIRA issues that will be created in GitHub. See *Issue filters* for further detail.
c.github.before_issue_create	List of callables that transform the fields used to create a new GitHub issue. This can (for example) be used to override jirahub's behavior, or set values for fields (such as `assignee`) that aren't otherwise managed by jirahub. See *Issue hooks* for further detail.

3.2.3 general

These are parameters shared by GitHub and JIRA.

Name	Description
c.before_issue_update	List of callables that transform the fields used to update an issue. This can (for example) be used to override jirahub's behavior, or set values for arbitrary custom fields. See *Issue hooks* for further detail.

3.2.4 Multiple configuration files

To facilitate re-use of common parameters, jirahub commands will accept multiple configuration file paths.

CHAPTER

FOUR

COMMAND-LINE INTERFACE

Jirahub is controlled with the `jirahub` command. There are three subcommands: `generate-config`, `check-permissions`, and `sync`.

4.1 generate-config

The `generate-config` command will print a template jirahub configuration file to stdout:

```
$ jirahub generate-config > my-jirahub-config.py
```

4.2 check-permissions

Once you're satisfied with your configuration file, you can submit it to the `check-permissions` command for verification. Jirahub will attempt to connect to your JIRA server and GitHub repository and report any failures. It will also list any missing permissions from JIRA or GitHub that are required for the features selected in the configuration file. A successful check looks like this:

```
$ jirahub check-permissions my-jirahub-config.py
JIRA and GitHub permissions are sufficient
```

And an unsuccessful check:

```
$ jirahub check-permissions my-jirahub-config.py
JIRA and/or GitHub permissions must be corrected:
sync_comments is enabled, but JIRA user has not been granted the DELETE_OWN_COMMENTS␣
↪permission.
sync_status is enabled, but JIRA user has not been granted the CLOSE_ISSUES␣
↪permission.
GitHub rejected credentials.  Check JIRAHUB_GITHUB_TOKEN and try again.
```

4.3 sync

The `sync` command does the work of syncing issues and comments. At minimum, you must specify a configuration file. Additional options include:

- **–min-updated-at**: Restrict jirahub's activity to issues updated after this timestamp. The timestamp format is ISO-8601 in UTC with no timezone suffix (e.g., 1983-11-20T11:00:00).
- **–state-path**: Path to a JSON file containing the same timestamp described above, as well as a list of issues that failed. The file will be updated after each run.
- **–dry-run**: Query issues and report changes to the (verbose) log, but do not change any data.
- **–verbose**: Enable verbose logging

4.3.1 Jirahub sync as a cron job

Users will likely want to run `jirahub sync` in a cron job, so that it can regularly poll JIRA/GitHub for changes. We recommend use of the lockrun tool to avoid overlap between jirahub processes. Your cron line might look something like this:

```
*/5 * * * * lockrun --lockfile=/path/to/jirahub.lockrun -- jirahub sync /path/to/my-
↪jirahub-config.py --state-path /path/to/jirahub-state.json >> /path/to/jirahub.log␣
↪2>&1
```

CHAPTER FIVE

CUSTOM FORMATTERS

The `issue_title_formatter`, `issue_body_formatter`, and `comment_body_formatter` parameters allow you to customize how the issue and comment text fields are written to the linked issue. The issue formatters are callables that receive two arguments, the original `jirahub.entities.Issue` that is being synced, and the title/body string. The title/body has already been modified by jirahub; it has been redacted, if that feature is enabled, and the formatting has been transformed to suit the target service. The following formatter adds a "JIRAHUB: " prefix to JIRA issue titles:

```
def custom_formatter(issue, title):
  return "JIRAHUB: " + title

c.jira.issue_title_formatter = custom_formatter
```

The original issue title/body (without jirahub's modifications) is available from the issue object:

```
def custom_formatter(issue, body):
  return "This is the original body: " + issue.body

c.jira.issue_body_formatter = custom_formatter
```

If you need access to a custom field that isn't recognized by jirahub, that is available via the `raw_issue`, which contains the `jira.resources.Issue` or `github.Issue` that was used to construct the jirahub Issue.

```
def custom_formatter(issue, body):
  return "This is some custom field value: " + issue.raw_issue.body

c.jira.issue_body_formatter = custom_formatter
```

The `comment_body_formatter` is similar, except that it receives three arguments, the original `jirahub.entities.Issue`, the `jirahub.entities.Comment`, and the comment body.

```
def custom_formatter(issue, comment, body):
  return "Check out this great comment from GitHub: " + body

c.jira.comment_body_formatter = custom_formatter
```

The unmodified comment body is available from `comment.body`, and the JIRA/GitHub comment object from `comment.raw_comment`.

CHAPTER SIX

ISSUE FILTERS

The `issue_filter` parameter allows you to select issues that will be created in the target service. The filter is a callable that receives a single argument, the original `jirahub.entities.Issue` that is a candidate for sync, and returns True to create it, or False to ignore it. For example, this filter only syncs issues with a certain label:

```python
def issue_filter(issue):
  return "sync-me" in issue.labels

c.jira.issue_filter = issue_filter
```

This feature can be used to sync issues based on "commands" issued by commenters:

```python
ADMINISTRATOR_USERNAMES = {
  "linda",
  "frank"
}

def issue_filter(issue):
  return any(c for c in issue.comments if c.user.username in ADMINISTRATOR_USERNAMES␣
↪and "SYNC ME PLEASE" in c.body)

c.jira.issue_filter = issue_filter
```

CHAPTER

SEVEN

ISSUE HOOKS

The before_issue_create and before_issue_update hooks allow you to transform the fields sent to JIRA/GitHub when an issue is created. They can override jirahub's behavior, or set custom fields that aren't otherwise managed by jirahub.

The create hooks are callables that receive 2 required arguments: the original jirahub.entities.Issue, and a dict of fields that will be used to create the issue. The 3rd optional argument, if present, will receive the the jirahub.IssueSync instance. Each callable must return a dict containing the transformed fields.

The update hooks are callables that receive 4 required arguments: the updated jirahub.entities.Issue, a dict of fields that will be used to modify that issue, the corresponding linked jirahub.entities.Issue, and another dict of fields. The 5th optional argument, if present, will receive the jirahub.IssueSync instance. Each callable must return two dict instances containing the transformed fields.

For example, this create hook sets a custom JIRA field:

```
def hook(issue, fields):
  fields["custom_jira_field"] = "some custom value"
  return fields

c.jira.before_issue_create.append(hook)
```

CHAPTER

EIGHT

MANUALLY LINKING ISSUES

It is possible to link existing GitHub and JIRA issues by hand by setting the GitHub issue URL field in JIRA. jirahub will begin syncing the two issues on next run. Take care that you don't link two JIRA issues to the same GitHub issue, that way lies peril (undefined behavior).