

Migrating from Travis CI with GitHub Actions Importer

In this article

- About migrating from Travis CI with GitHub Actions Importer
- Installing the GitHub Actions Importer CLI extension
- Configuring credentials
- Perform an audit of Travis CI
- Forecast potential build runner usage
- Perform a dry-run migration of a Travis CI pipeline
- Perform a production migration of a Travis CI pipeline
- Reference
- Legal notice

Learn how to use GitHub Actions Importer to automate the migration of your Travis CI pipelines to GitHub Actions.

[Legal notice](#)

About migrating from Travis CI with GitHub Actions Importer

The instructions below will guide you through configuring your environment to use GitHub Actions Importer to migrate Travis CI pipelines to GitHub Actions.

Prerequisites

- A Travis CI account or organization with pipelines and jobs that you want to convert to GitHub Actions workflows.
- Access to create a Travis CI API access token for your account or organization.
- An environment where you can run Linux-based containers, and can install the necessary tools.
 - Docker is [installed](#) and running.
 - [GitHub CLI](#) is installed.

Note: The GitHub Actions Importer container and CLI do not need to be installed on the same server as your CI platform.

Limitations

There are some limitations when migrating from Travis CI pipelines to GitHub Actions with GitHub Actions Importer.

Manual tasks

Certain Travis CI constructs must be migrated manually. These include:

- Secrets
- Unknown job properties

For more information on manual migrations, see "[Migrating from Travis CI to GitHub Actions](#)."

Travis CI project languages

GitHub Actions Importer transforms Travis CI project languages by adding a set of preconfigured build tools and a default build script to the transformed workflow. If no language is explicitly declared, GitHub Actions Importer assumes a project language is Ruby.

For a list of the project languages supported by GitHub Actions Importer, see "[Supported project languages](#)."

Installing the GitHub Actions Importer CLI extension

- 1 Install the GitHub Actions Importer CLI extension:

Bash



```
gh extension install github/gh-actions-importer
```

- 2 Verify that the extension is installed:

```
$ gh actions-importer -h
Options:
  -?, -h, --help  Show help and usage information

Commands:
  update      Update to the latest version of GitHub Actions Importer.
  version     Display the version of GitHub Actions Importer.
  configure   Start an interactive prompt to configure credentials used to
authenticate with your CI server(s).
  audit       Plan your CI/CD migration by analyzing your current CI/CD
footprint.
  forecast    Forecast GitHub Actions usage from historical pipeline
utilization.
  dry-run     Convert a pipeline to a GitHub Actions workflow and output its
yaml file.
  migrate     Convert a pipeline to a GitHub Actions workflow and open a pull
request with the changes.
```

Configuring credentials

The `configure` CLI command is used to set required credentials and options for GitHub Actions Importer when working with Travis CI and GitHub.

- 1 Create a GitHub personal access token (classic). For more information, see "[Managing your personal access tokens](#)."

Your token must have the `workflow` scope.

After creating the token, copy it and save it in a safe location for later use.

- 2 Create a Travis CI API access token. For more information, see [Get your Travis CI API token](#) in the Travis CI documentation.

After creating the token, copy it and save it in a safe location for later use.

- 3 In your terminal, run the GitHub Actions Importer `configure` CLI command:

```
gh actions-importer configure
```

The `configure` command will prompt you for the following information:

- For "Which CI providers are you configuring?", use the arrow keys to select `Travis CI`, press `Space` to select it, then press `Enter`.
- For "Personal access token for GitHub", enter the value of the personal access token (classic) that you created earlier, and press `Enter`.
- For "Base url of the GitHub instance", enter the URL for your GitHub Enterprise Server instance, and press `Enter`.
- For "Personal access token for Travis CI", enter the value for the Travis CI API access token that you created earlier, and press `Enter`.
- For "Base url of the Travis CI instance", enter the URL of your Travis CI instance, and press `Enter`.
- For "Travis CI organization name", enter the name of your Travis CI organization, and press `Enter`.

An example of the output of the `configure` command is shown below.

```
$ gh actions-importer configure
✓ Which CI providers are you configuring?: Travis CI
Enter the following values (leave empty to omit):
✓ Personal access token for GitHub: *****
✓ Base url of the GitHub instance: https://github.com
✓ Personal access token for Travis CI: *****
✓ Base url of the Travis CI instance: https://travis-ci.com
✓ Travis CI organization name: actions-importer-labs
Environment variables successfully updated.
```

- 4 In your terminal, run the GitHub Actions Importer `update` CLI command to connect to GitHub Packages Container registry and ensure that the container image is updated to the latest version:

```
gh actions-importer update
```

The output of the command should be similar to below:

```
Updating ghcr.io/actions-importer/cli:latest...
ghcr.io/actions-importer/cli:latest up-to-date
```

Perform an audit of Travis CI

You can use the `audit` command to get a high-level view of all pipelines in a Travis CI server.

The `audit` command performs the following steps:

- 1 Fetches all of the projects defined in a Travis CI server.
- 2 Converts each pipeline to its equivalent GitHub Actions workflow.
- 3 Generates a report that summarizes how complete and complex of a migration is possible with GitHub Actions Importer.

Running the audit command [↗](#)

To perform an audit of a Travis CI server, run the following command in your terminal:

```
gh actions-importer audit travis-ci --output-dir tmp/audit
```

Inspecting the audit results [↗](#)

The files in the specified output directory contain the results of the audit. See the `audit_summary.md` file for a summary of the audit results.

The audit summary has the following sections.

Pipelines [↗](#)

The "Pipelines" section contains a high-level statistics regarding the conversion rate done by GitHub Actions Importer.

Listed below are some key terms that can appear in the "Pipelines" section:

- **Successful** pipelines had 100% of the pipeline constructs and individual items converted automatically to their GitHub Actions equivalent.
- **Partially successful** pipelines had all of the pipeline constructs converted, however, there were some individual items that were not converted automatically to their GitHub Actions equivalent.
- **Unsupported** pipelines are definition types that are not supported by GitHub Actions Importer.
- **Failed** pipelines encountered a fatal error when being converted. This can occur for one of three reasons:
 - The pipeline was misconfigured and not valid in Bamboo.
 - GitHub Actions Importer encountered an internal error when converting it.
 - There was an unsuccessful network response that caused the pipeline to be inaccessible, which is often due to invalid credentials.

Build steps [↗](#)

The "Build steps" section contains an overview of individual build steps that are used across all pipelines, and how many were automatically converted by GitHub Actions Importer.

Listed below are some key terms that can appear in the "Build steps" section:

- A **known** build step is a step that was automatically converted to an equivalent action.
- An **unknown** build step is a step that was not automatically converted to an equivalent action.
- An **unsupported** build step is a step that is either:
 - Fundamentally not supported by GitHub Actions.

- Configured in a way that is incompatible with GitHub Actions.
- An **action** is a list of the actions that were used in the converted workflows. This can be important for:
 - If you use GitHub Enterprise Server, gathering the list of actions to sync to your instance.
 - Defining an organization-level allowlist of actions that are used. This list of actions is a comprehensive list of actions that your security or compliance teams may need to review.

Manual tasks [↗](#)

The "Manual tasks" section contains an overview of tasks that GitHub Actions Importer is not able to complete automatically, and that you must complete manually.

Listed below are some key terms that can appear in the "Manual tasks" section:

- A **secret** is a repository or organization-level secret that is used in the converted pipelines. These secrets must be created manually in GitHub Actions for these pipelines to function properly. For more information, see "[Using secrets in GitHub Actions](#)."
- A **self-hosted runner** refers to a label of a runner that is referenced in a converted pipeline that is not a GitHub-hosted runner. You will need to manually define these runners for these pipelines to function properly.

Files [↗](#)

The final section of the audit report provides a manifest of all the files that were written to disk during the audit.

Each pipeline file has a variety of files included in the audit, including:

- The original pipeline as it was defined in GitHub.
- Any network responses used to convert the pipeline.
- The converted workflow file.
- Stack traces that can be used to troubleshoot a failed pipeline conversion.

Additionally, the `workflow_usage.csv` file contains a comma-separated list of all actions, secrets, and runners that are used by each successfully converted pipeline. This can be useful for determining which workflows use which actions, secrets, or runners, and can be useful for performing security reviews.

Forecast potential build runner usage [↗](#)

You can use the `forecast` command to forecast potential GitHub Actions usage by computing metrics from completed pipeline runs in your Travis CI server.

Running the forecast command [↗](#)

To perform a forecast of potential GitHub Actions usage, run the following command in your terminal. By default, GitHub Actions Importer includes the previous seven days in the forecast report.

```
gh actions-importer forecast travis-ci --output-dir tmp/forecast
```

Inspecting the forecast report [↗](#)

The `forecast_report.md` file in the specified output directory contains the results of the forecast.

Listed below are some key terms that can appear in the forecast report:

- The **job count** is the total number of completed jobs.
- The **pipeline count** is the number of unique pipelines used.
- **Execution time** describes the amount of time a runner spent on a job. This metric can be used to help plan for the cost of GitHub-hosted runners.
 - This metric is correlated to how much you should expect to spend in GitHub Actions. This will vary depending on the hardware used for these minutes. You can use the [GitHub Actions pricing calculator](#) to estimate the costs.
- **Queue time** metrics describe the amount of time a job spent waiting for a runner to be available to execute it.
- **Concurrent jobs** metrics describe the amount of jobs running at any given time. This metric can be used to define the number of runners you should configure.

Additionally, these metrics are defined for each queue of runners in Travis CI. This is especially useful if there is a mix of hosted or self-hosted runners, or high or low spec machines, so you can see metrics specific to different types of runners.

Perform a dry-run migration of a Travis CI pipeline [↗](#)

You can use the `dry-run` command to convert a Travis CI pipeline to an equivalent GitHub Actions workflow. A dry-run creates the output files in a specified directory, but does not open a pull request to migrate the pipeline.

To perform a dry run of migrating your Travis CI pipelines to GitHub Actions, run the following command in your terminal, replacing `my-travis-ci-repository` with the name of your Travis CI repository.

```
gh actions-importer dry-run travis-ci --travis-ci-repository my-travis-ci-repository --output-dir tmp/dry-run
```

You can view the logs of the dry run and the converted workflow files in the specified output directory.

If there is anything that GitHub Actions Importer was not able to convert automatically, such as unknown build steps or a partially successful pipeline, you might want to create custom transformers to further customize the conversion process. For more information, see "[Extending GitHub Actions Importer with custom transformers](#)."

Perform a production migration of a Travis CI pipeline [↗](#)

You can use the `migrate` command to convert a Travis CI pipeline and open a pull request with the equivalent GitHub Actions workflow.

Running the migrate command [↗](#)

To migrate a Travis CI pipeline to GitHub Actions, run the following command in your terminal, replacing the `target-url` value with the URL for your GitHub repository, and `my-travis-ci-repository` with the name of your Travis CI repository.

```
gh actions-importer migrate travis-ci --target-url https://github.com/octo-org/octo-repo --output-dir tmp/migrate --travis-ci-repository my-travis-ci-
```

The command's output includes the URL to the pull request that adds the converted workflow to your repository. An example of a successful output is similar to the following:

```
$ gh actions-importer migrate travis-ci --target-url https://github.com/octo-org/octo-repo --output-dir tmp/migrate --travis-ci-repository my-travis-ci-repository
[2022-08-20 22:08:20] Logs: 'tmp/migrate/log/actions-importer-20220916-014033.log'
[2022-08-20 22:08:20] Pull request: 'https://github.com/octo-org/octo-repo/pull/1'
```

Inspecting the pull request

The output from a successful run of the `migrate` command contains a link to the new pull request that adds the converted workflow to your repository.

Some important elements of the pull request include:

- In the pull request description, a section called **Manual steps**, which lists steps that you must manually complete before you can finish migrating your pipelines to GitHub Actions. For example, this section might tell you to create any secrets used in your workflows.
- The converted workflows file. Select the **Files changed** tab in the pull request to view the workflow file that will be added to your GitHub Enterprise Server repository.

When you are finished inspecting the pull request, you can merge it to add the workflow to your GitHub Enterprise Server repository.

Reference

This section contains reference information on environment variables, optional arguments, and supported syntax when using GitHub Actions Importer to migrate from Travis CI.

Using environment variables

GitHub Actions Importer uses environment variables for its authentication configuration. These variables are set when following the configuration process using the `configure` command. For more information, see the "[Configure credentials for GitHub Actions Importer](#)" section.

GitHub Actions Importer uses the following environment variables to connect to your Travis CI instance:

- `GITHUB_ACCESS_TOKEN` : The personal access token (classic) used to create pull requests with a converted workflow (requires the `workflow` scope).
- `GITHUB_INSTANCE_URL` : The URL to the target GitHub instance (for example, `https://github.com`).
- `TRAVIS_CI_ACCESS_TOKEN` : The Travis CI API access token used to view Travis CI resources.
- `TRAVIS_CI_ORGANIZATION` : The organization name of your Travis CI instance.
- `TRAVIS_CI_INSTANCE_URL` : The URL of the Travis CI instance.
- `TRAVIS_CI_SOURCE_GITHUB_ACCESS_TOKEN` : (Optional) The personal access token used to authenticate with your source GitHub instance. If not provided, `GITHUB_ACCESS_TOKEN` will be used instead.
- `TRAVIS_CI_SOURCE_GITHUB_INSTANCE_URL` : (Optional) The URL to the source GitHub

instance, such as <https://github.com>. If not provided, `GITHUB_INSTANCE_URL` will be used instead.

These environment variables can be specified in a `.env.local` file that is loaded by GitHub Actions Importer when it is run.

Using optional arguments [↗](#)

There are optional arguments you can use with the GitHub Actions Importer subcommands to customize your migration.

`--source-file-path` [↗](#)

You can use the `--source-file-path` argument with the `forecast`, `dry-run`, or `migrate` subcommands.

By default, GitHub Actions Importer fetches pipeline contents from source control. The `--source-file-path` argument tells GitHub Actions Importer to use the specified source file path instead.

For example:

```
gh actions-importer dry-run travis-ci --output-dir ./path/to/output/ --travis-ci-repository my-travis-ci-repository --source-file-path ./path/to/.travis.yml
```

`--allow-inactive-repositories` [↗](#)

You can use this argument to specify whether GitHub Actions Importer should include inactive repositories in an audit. If this option is not set, inactive repositories are not included in audits.

```
gh actions-importer dry-run travis-ci --output-dir ./path/to/output/ --travis-ci-repository my-travis-ci-repository --allow-inactive-repositories
```

`--config-file-path` [↗](#)

You can use the `--config-file-path` argument with the `audit`, `dry-run`, and `migrate` subcommands.

By default, GitHub Actions Importer fetches pipeline contents from source control. The `--config-file-path` argument tells GitHub Actions Importer to use the specified source files instead.

Audit example [↗](#)

In this example, GitHub Actions Importer uses the specified YAML configuration file to perform an audit.

```
gh actions-importer audit travis-ci --output-dir ./path/to/output/ --config-file-path ./path/to/travis-ci/config.yml
```

To audit a Travis CI instance using a configuration file, the file must be in the following format and each `repository_slug` value must be unique:

```
source_files:
- repository_slug: travis-org-name/travis-repo-name
  path: path/to/.travis.yml
```



```
- repository_slug: travis-org-name/some-other-travis-repo-name
  path: path/to/.travis.yml
```

Dry run example [↗](#)

In this example, GitHub Actions Importer uses the specified YAML configuration file as the source file to perform a dry run.

The pipeline is selected by matching the `repository_slug` in the configuration file to the value of the `--travis-ci-repository` option. The `path` is then used to pull the specified source file.

```
gh actions-importer dry-run travis-ci --travis-ci-repository travis-org-name/travis-repo-name --output-dir ./output/ --config-file-path ./path/to/travis-ci/config.yml
```

Supported project languages [↗](#)

GitHub Actions Importer supports migrating Travis CI projects in the following languages.

- android
- bash
- c
- clojure
- c++
- crystal
- c#
- d
- dart
- elixir
- erlang
- generic
- go
- groovy
- haskell
- haxe
- java
- julia
- matlab
- minimal
- nix
- node_js
- objective-c
- perl
- perl6
- php
- python
- r
- ruby
- rust
- scala
- sh
- shell

- `smalltalk`
- `swift`

Supported syntax for Travis CI pipelines [↗](#)

The following table shows the type of properties GitHub Actions Importer is currently able to convert. For more details about how Travis CI pipeline syntax aligns with GitHub Actions, see "[Migrating from Travis CI to GitHub Actions](#)".

Travis CI	GitHub Actions	Status
branches	<ul style="list-style-type: none"> • <code>on.<push>.<branches></code> 	Supported
build_pull_requests	<ul style="list-style-type: none"> • <code>on.<pull_request></code> 	Supported
env	<ul style="list-style-type: none"> • <code>env</code> • <code>jobs.<job_id>.env</code> • <code>jobs.<job_id>.steps.env</code> 	Supported
if	<ul style="list-style-type: none"> • <code>jobs.<job_id>.if</code> 	Supported
job	<ul style="list-style-type: none"> • <code>jobs.<job_id></code> • <code>jobs.<job_id>.name</code> 	Supported
matrix	<ul style="list-style-type: none"> • <code>jobs.<job_id>.strategy</code> • <code>jobs.<job_id>.strategy.fail-fast</code> • <code>jobs.<job_id>.strategy.matrix</code> 	Supported
os & dist	<ul style="list-style-type: none"> • <code>runners</code> 	Supported
scripts	<ul style="list-style-type: none"> • <code>jobs.<job_id>.steps</code> 	Supported
stages	<ul style="list-style-type: none"> • <code>jobs</code> 	Supported
env	<ul style="list-style-type: none"> • <code>on</code> 	Partially supported
branches	<ul style="list-style-type: none"> • <code>on.<push>.<tags></code> • <code>on.<push>.paths</code> 	Unsupported
build_pull_requests	<ul style="list-style-type: none"> • <code>on.<pull_request>.<branches></code> • <code>on.<pull_request>.<tags></code> • <code>on.<pull_request>.paths</code> 	Unsupported
cron triggers	<ul style="list-style-type: none"> • <code>on.schedule</code> • <code>on.workflow_run</code> 	Unsupported
env	<ul style="list-style-type: none"> • <code>jobs.<job_id>.timeout-minutes</code> • <code>on.<event_name>.types</code> 	Unsupported
job	<ul style="list-style-type: none"> • <code>jobs.<job_id>.container</code> 	Unsupported
os & dist	<ul style="list-style-type: none"> • <code>self hosted runners</code> 	Unsupported

For information about supported Travis CI constructs, see the [github/gh-actions-importer repository](#).

Environment variables syntax

GitHub Actions Importer uses the mapping in the table below to convert default Travis CI environment variables to the closest equivalent in GitHub Actions.

Travis CI	GitHub Actions
<code>\$CONTINUOUS_INTEGRATION</code>	<code>\$CI</code>
<code>\$USER</code>	<code>\${{ github.actor }}</code>
<code>\$HOME</code>	<code>\${{ github.workspace }}</code>
<code>\$TRAVIS_BRANCH</code>	<code>\${{ github.ref }}</code>
<code>\$TRAVIS_BUILD_DIR</code>	<code>\${{ github.workspace }}</code>
<code>\$TRAVIS_BUILD_ID</code>	<code>\${{ github.run_number }}</code>
<code>\$TRAVIS_BUILD_NUMBER</code>	<code>\${{ github.run_id }}</code>
<code>\$TRAVIS_COMMIT</code>	<code>\${{ github.sha }}</code>
<code>\$TRAVIS_EVENT_TYPE</code>	<code>\${{ github.event_name }}</code>
<code>\$TRAVIS_PULL_REQUEST_BRANCH</code>	<code>\${{ github.base_ref }}</code>
<code>\$TRAVIS_PULL_REQUEST</code>	<code>\${{ github.event.number }}</code>
<code>\$TRAVIS_PULL_REQUEST_SHA</code>	<code>\${{ github.head.sha }}</code>
<code>\$TRAVIS_PULL_REQUEST_SLUG</code>	<code>\${{ github.repository }}</code>
<code>\$TRAVIS_TAG</code>	<code>\${{ github.ref }}</code>
<code>\$TRAVIS_OS_NAME</code>	<code>\${{ runner.os }}</code>
<code>\$TRAVIS_JOB_ID</code>	<code>\${{ github.job }}</code>
<code>\$TRAVIS_REPO_SLUG</code>	<code>\${{ github.repository_owner/github.repository }}</code>
<code>\$TRAVIS_BUILD_WEB_URL</code>	<code>\${{ github.server_url }}/\${{ github.repository }}/actions/runs/\${{ github.run_id }}</code>

Legal notice

Portions have been adapted from <https://github.com/github/gh-actions-importer/> under the MIT license:

MIT License

Copyright (c) 2022 GitHub

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Legal

© 2023 GitHub, Inc. [Terms](#) [Privacy](#) [Status](#) [Pricing](#) [Expert services](#) [Blog](#)