Explore Postman's command-line companion

The Postman CLI is a secure command-line companion for Postman. It's signed and supported by Postman, like the Postman app. The Postman CLI supports the following features:

- Sign in and sign out of Postman from the command line.
- Run a collection with its collection ID or path, and send the run results to the Postman cloud by default.
- Generate a local collection run report with built-in reporters.
- Check API definitions against configured API governance and API security rules.

> The Postman CLI requires a valid Postman API key. For more information, see Generate and use Postman API keys.

Comparing the Postman CLI and Newman

The table below shows a high-level comparison of the Postman CLI and Newman.

Postman CLI	Newman
Created by Postman	Created by Postman
Maintained and supported by Postman	Open source; supported by community contributions
Supports collection runs	Supports collection runs
Automatically sends collection run results to Postman by default	Supports ingesting run results to Postman using a reporter
Package is signed by Postman	Package isn't signed by Postman
Distributed as a downloadable package	Distributed on npm
Downloadable programmatically	Downloadable programmatically
Not available as a library	Available as a library
Supports sign in and sign out	Doesn't support sign in and sign out
Checks API specifications against your configured API governance and API security rules	Doesn't check API specifications against your configured API governance and API security rules

Decide which command-line companion to use

You can use the Postman CLI or Newman to run and test collections from the command line. One may be a better fit, depending on your use case or preferences.

For example, assume you already manage your own security for open-source software, and you want to run collections from a script. Also assume you want visibility into any software you build into your CI/CD pipeline. Newman would be a good fit for this use case because Newman's repository is public and Newman isn't signed or secured by Postman.

Here's another example. Assume you don't already support or secure any open-source software, and you want the software you use to be signed and secured by its developer. The Postman CLI would be a good fit for this use case because the Postman CLI is signed and secured by Postman.

> Learn how to install the Postman CLI.

About the Postman CLI and Postman API usage

Some Postman CLI commands use the Postman API to fetch data from and send data to Postman's servers. These commands count toward your monthly Postman API usage. The number of Postman API calls you can make each month depends on your Postman plan.

The following Postman CLI commands make calls to the Postman API and count toward your Postman API usage:

- `postman login` - Uses one call to authenticate a user with a Postman API key.
- `postman collection run` - Uses one call to fetch a collection by ID, one call to fetch an environment (if any), and one call to send data back to Postman.
- `postman spec lint` - Uses one call to fetch the API governance rules and one call to send a report back to Postman.
- `postman api lint` - Uses one call to fetch the API governance and security rules and one call to send a report back to Postman.

> Learn more about Postman CLI command options.

Install the Postman CLI

Download and install the Postman CLI by following the instructions for your operating system below.

> The Postman CLI supports the same system requirements as the Postman desktop app. For a complete list of requirements, see Install and update Postman.

Windows installation

Run the following command to install the Postman CLI for Windows. This will download and run an install script. The script creates a `%USERPROFILE%\AppData\Local\Microsoft\WindowsApps` directory if it doesn't exist yet, then installs the `postman` binary there.

```
powershell.exe -NoProfile -InputFormat None -ExecutionPolicy AllSigned -Command "[System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://dl-cli.pstmn.io/install/win64.ps1'))"
```

macOS (Apple silicon) installation

Run the following command to install the Postman CLI for Macs with an Apple silicon processor. This will download an install script and run it. The install script creates a `/usr/local/bin` directory if it doesn't exist yet, then installs the `postman` binary there.

```
curl -o- "https://dl-cli.pstmn.io/install/osx_arm64.sh" | sh
```

macOS (Intel) installation

Run the following command to install the Postman CLI for Macs with an Intel processor. This will download an install script and run it. The install script creates a `/usr/local/bin` directory if it doesn't exist yet, then installs the `postman` binary there.

```
curl -o- "https://dl-cli.pstmn.io/install/osx_64.sh" | sh
```

Linux installation

Run the following command to install the Postman CLI for the latest Linux version. This will download an install script and run it. The install script creates a `/usr/local/bin` directory if it doesn't exist yet, then installs the `postman` binary there.

```
curl -o- "https://dl-cli.pstmn.io/install/linux64.sh" | sh
```

Update your CLI installation

To update your Postman CLI installation to the latest version, run the same command you used to install it. The new version will overwrite the earlier version. For details about the latest changes and features, see the Postman CLI release notes.

CLI release notes.

Postman CLI command options

This topic covers the commands and options supported by the Postman CLI.

Basic command line options

`postman`

The base command of the Postman CLI is `postman`. Run this command with the syntax and options detailed below.

Options

Option	Details
`--help`, `-h`	Returns information about Postman CLI commands and options.
`--version`, `-v`	Returns the version number for the Postman CLI.

Example

```
postman -v
```

Sign in and out

You can use the Postman CLI to sign in and out of Postman with the `login` and `logout` commands.

`postman login`

This command authenticates the user and locally caches the Postman API key. The `login` command requires one option, `--with-api-key`, that accepts the Postman API key. Signing in is required once per session. You remain signed in until you use the `logout` command or your Postman API key expires.

Options

Option	Details
`--with-api-key <api-key>`	Authenticate the user with the given API key.

Example

```
postman login --with-api-key ABCD-1234-1234-1234-1234-1234
```

`postman logout`

This command signs you out of Postman and deletes the stored API key.

Example

```
postman logout
```

Run collections

> Postman supports running HTTP collections. You can't run multi-protocol collections.

You can run your collections with the `postman collection run` command:

`postman collection run`

This command runs a collection and sends the run results to the Postman cloud. You can specify the collection with its file path or Collection ID.

> You can find the collection ID in Postman. Click **Collections** in the sidebar and select a collection. Then click **Info** ⓘ in the right sidebar to view or copy the collection ID.

Options

Option	Details
`--bail [optional modifiers]`	Specifies whether to stop a collection run on encountering the first error. `--bail` can optionally accept two modifiers: `--folder` and `--failure`. `--folder` skips the entire collection run if there are any errors. If a test fails, `--failure` stops the collection run after completing the current script.
`--color [value]`	Controls colors in the CLI output. Accepts `on`, `off`, and `auto`. The default is `auto`. With `auto`, the Postman CLI attempts to automatically turn color on or off based on the color support in the terminal. This behavior can be changed by using the `on` or `off` value.
`--cookie-jar [path]`	Specifies the file path for a `JSON` cookie jar. This uses the `tough-cookie` library to deserialize the file.
`--delay-request [number]`	Specifies a delay (in milliseconds) between requests. The default is 0.
`--disable-unicode`	Replaces all symbols in the output with their plain text equivalents.
`--environment [UID] or [file-path]`, `-e`	Specifies an environment file path or UID.
`--env-var "[environment-variable-name]=[environment-variable-value]"`	Specifies environment variables in a `key=value` format. Multiple CLI environment variables can be added by using `--env-var` multiple times, for example: `--env-var "this=that" --env-var "alpha=beta"`.
`--export-cookie-jar [path]`	Specifies the path where the Postman CLI will output the final cookie jar file after completing a run. This uses the `tough-cookie` library to serialize the file.
`--global-var "[global-variable-name]=[global-variable-value]"`	Specifies global variables in a `key=value` format. Multiple CLI global variables can be added by using `--global-var` multiple times, for example: `--global-var "this=that" --global-var "alpha=beta"`.
`--globals [file-path]`, `-g`	Specifies a path to a file containing global variables. Global variables are similar to environment variables but have lower precedence and can be overridden by environment variables having the same name.
`--integration-id [ID]`	Specifies an integration ID when using an integration with CI/CD. This sends the Postman CLI results to the correct integration in Postman. When you generate the Postman CLI command for a Git integrated API collection, the integration ID is automatically added to the command. To learn how to generate the Postman CLI command with the integration ID, go to Configure the Postman CLI for CI.
`--iteration-count [number]`, `-n`	Specifies the number of times the collection will run when used in conjunction with the iteration data file.
`--iteration-data [file-path]`, `-d`	Specifies the local file path to a data file (JSON or CSV) to use for each iteration.
`-i [requestUID] or [folderUID]`	Runs only the specified folder UID or request UID from the collection. Multiple items can be run in order by specifying `-i` multiple times, for example: `postman collection run collectionUID -i folder1UID -i folder2UID`.
`-i [requestName] or [folderName]`	Runs only the specified folder name or request name from the collection. If there are duplicate names, the Postman CLI runs the folder or request that appears first.
`--ignore-redirects`	Prevents the Postman CLI from automatically following 3XX redirect responses.
`--insecure`, `-k`	Turns off SSL verification checks and enables self-signed SSL certificates.
`--no-insecure-file-read`	Prevents reading of files situated outside of the working directory.
`--silent`	Turns off terminal output.
`--ssl-client-cert-list <path>`	Specifies the path to a client certificates configuration (JSON).
`--ssl-client-cert <path>`	Specifies the path to a client certificate (PEM).
`--ssl-client-key <path>`	Specifies the path to a client certificate private key.
`--ssl-client-passphrase <passphrase>`	Specifies the client certificate passphrase (for a protected key).
`--ssl-extra-ca-certs <path>`	Specifies more trusted CA certificates (PEM).
`--suppress-exit-code`, `-x`	Specifies whether to override the default exit code for the current run.
`--timeout [number]`	Specifies the time (in milliseconds) to wait for the entire collection run to complete.
`--timeout-request [number]`	Specifies a time (in milliseconds) to wait for requests to return a response. The default is 0.
`--timeout-script [number]`	Specifies the time (in milliseconds) to wait for scripts to complete. The default is 0.
`--verbose`	Shows detailed information for the collection run and each request sent.
`--working-dir [path]`	Sets the path of the working directory to use while reading files with relative paths. This defaults to the current directory.
`--reporters [reporter]`, `-r [reporter]`	Generates a local report for the collection run in the specified format: `cli`, `json`, `junit`, and `html`. If the `-reporters` option isn't specified, the `cli` report is output by default. To learn more, go to Generate collection run reports using the Postman CLI.

Examples

```
postman collection run /myCollectionFolderName/myCollectionFile.json

postman collection run 12345678-12345ab-1234-1ab2-1ab2-ab1234112a12
```

Governance and security

> API governance and security rules are available with Postman Enterprise plans.

API governance is the practice of applying a defined set of standards consistently across the API design and testing phases of your development process. The Postman CLI includes commands that checks your API specifications in Spec Hub and the Postman API Builder against your team's configured Postman API governance and security rules.

`postman spec lint`

This command runs validation checks with governance rules against an API specification in Spec Hub. Provide the ID for a specification that's in OpenAPI 3.0 format.

> Click ⓘ **Specification Info** in the right sidebar of a specification and copy its ID.

Options

Option	Details
`--fail-severity [severity]`, `-f`	Triggers an exit failure code for rule violations at or higher than the specified severity level. The options, in order of lowest to highest severity, are `HINT`, `INFO`, `WARNING`, and `ERROR` (default).
`--output [output format]`, `-o`	Controls the output format for issues found in the OpenAPI 3.0 specification. Accepts `JSON` or `CSV`. Defaults to table view if no output format is specified. See examples of JSON and CSV output.

Example

```
postman spec lint 12345678-12345ab-1234-1ab2-1ab2-ab1234112a12
```

Example output

You can change the output of governance rule violations to JSON or CSV. If you don't specify an output, it defaults to table view.

The following is an example of the output in JSON format:

```
{
  "violations": [
    {
      "line number": "4",
      "path": "info",
      "severity": "WARNING",
      "issue": "The info object should have a description."
    },
    {
      "line number": "10",
      "path": "paths./spacecrafts/{spacecraftId}/",
      "severity": "ERROR",
      "issue": "There should be no trailing slashes on paths."
    },
    {
      "line number": "20",
      "path": "paths./spacecrafts/{spacecraftId}/.get.responses",
      "severity": "WARNING",
      "issue": "Operation should return a 2xx HTTP status code"
    },
    {
      "line number": "20",
      "path": "paths./spacecrafts/{spacecraftId}/.get.responses",
      "severity": "WARNING",
      "issue": "Operation should return a 5xx HTTP status code"
    }
  ]
}
```

The following is an example of the output in CSV format:

```
line number,path,severity,issue
4,info,WARNING,The info object should have a description.
10,paths./spacecrafts/{spacecraftId}/,ERROR,There should be no trailing slashes on paths.
20,paths./spacecrafts/{spacecraftId}/.get.responses,WARNING,Operation should return a 2xx HTTP status code
20,paths./spacecrafts/{spacecraftId}/.get.responses,WARNING,Operation should return a 5xx HTTP status code
```

`postman api lint`

This command runs validation checks for governance and security rules against the API specification provided in the Postman config file, a local file, or a UUID. The `api lint` command shows a warning if it's unable to find the API ID to send data back to Postman.

> This command supports APIs in the Postman API Builder and aren't linked to Git.

Options

Option	Details
`--fail-severity [severity]`, `-f`	Triggers an exit failure code for rule violations at or higher than the specified severity level. The options, in order of lowest to highest severity, are `HINT`, `INFO`, `WARN`, and `ERROR` (default).
`--suppress-exit-code`, `-x`	Specifies whether to override the default exit code for the current run.

Example

```
postman api lint my-definition-file.json
postman api lint 12345678-12345ab-1234-1ab2-1ab2-ab1234112a12
```

Publish an API version

You can publish API versions in the Postman API Builder from the command line with the Postman CLI. Use the Postman CLI to automate the API version publishing process.

`postman publish api`

Publish a snapshot of an API for the given `apiId`. All elements linked to the API are published by default. You can choose which elements to publish by using other command options.

When publishing an API that's linked with Git, you must enter the command from inside the local Git repo. Also, you must provide paths to the schema directory and collection paths instead of IDs.

Options

Option	Details
`--name <name>`	Specifies the name of the version to publish.
`--release-notes <releaseNotes>`	Enter release notes as a string in quotes for the version to publish. This option supports Markdown.
`--collections <collectionIds/paths...>`	Specifies the collections to publish. If the API is linked with Git, provide the `filePath` instead of the ID.
`--api-definition <apiDefinitionId/directory/file>`	Specifies the API specification to publish. If the API is linked with Git, provide the `schemaDirectoryPath` or `schemaRootFilePath` instead of the ID.
`--do-not-poll`	Specifies not to poll for completion status of the publish action.
`--suppress-exit-code, -x`	Specifies whether to override the default exit code for the current run.

Example for repos not linked with Git

```
postman api publish <apiId> --name v1\
--release-notes "# Some release notes information"\
--collections <collectionId1> <collectionId2>\
--api-definition <apiDefinitionId>
```

Examples for repos linked with Git

The options for the `api publish` command differ depending on if you specified a schema folder or schema root file when setting up the Git integration. Git integrations added in Postman v10.18 or later use a schema root file. Git integrations added in other Postman versions use a schema folder. Learn more about connecting an API in the Postman API Builder to a Git repository.

- If the API uses a schema folder, publish the API using the `--api-definition <schemaDirectoryPath>` option:

  ```
  postman api publish <apiId> --name v1\
  --release-notes "# Some release notes information"\
  --collections <collectionPath1> <collectionPath2>\
  --api-definition <schemaDirectoryPath>
  ```

- If the API uses a schema root file, publish the API using the `--api-definition <schemaRootFilePath>` option:

  ```
  postman api publish <apiId> --name v1\
  --release-notes "# Some release notes information"\
  --collections <collectionPath1> <collectionPath2>\
  --api-definition <schemaRootFilePath>
  ```

> If you specify a file when a folder is required, or a folder when a file is required, the `api publish` command returns the following error: `API Definition <file/folder> isn't part of API <apiId>`. Try the command again using the other option.

Run a collection using the Postman CLI

You can use the Postman CLI to manually run collections to test the functionality of your API. You can also use the Postman CLI to automate collection runs in CI/CD pipelines.

> Postman supports running HTTP collections. You can't run multi-protocol collections.

When the Postman CLI runs a collection, the collection and its tests run locally, and the results are sent to the Postman cloud using an API call.

> The Postman CLI doesn't support OAuth 2.0 authentication. To learn how to use an OAuth 2.0 token with the Postman CLI, see OAuth 2.0 overview.

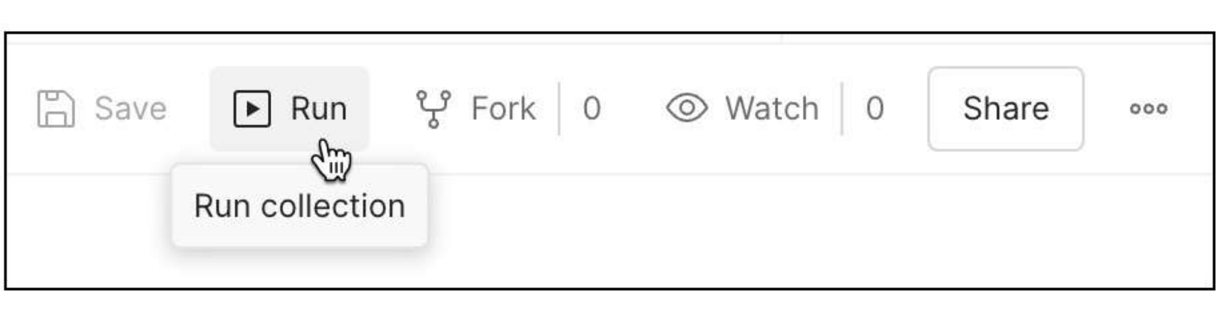
Run a collection locally with the Postman CLI

You can use the Postman CLI to run the requests in a collection or a folder.

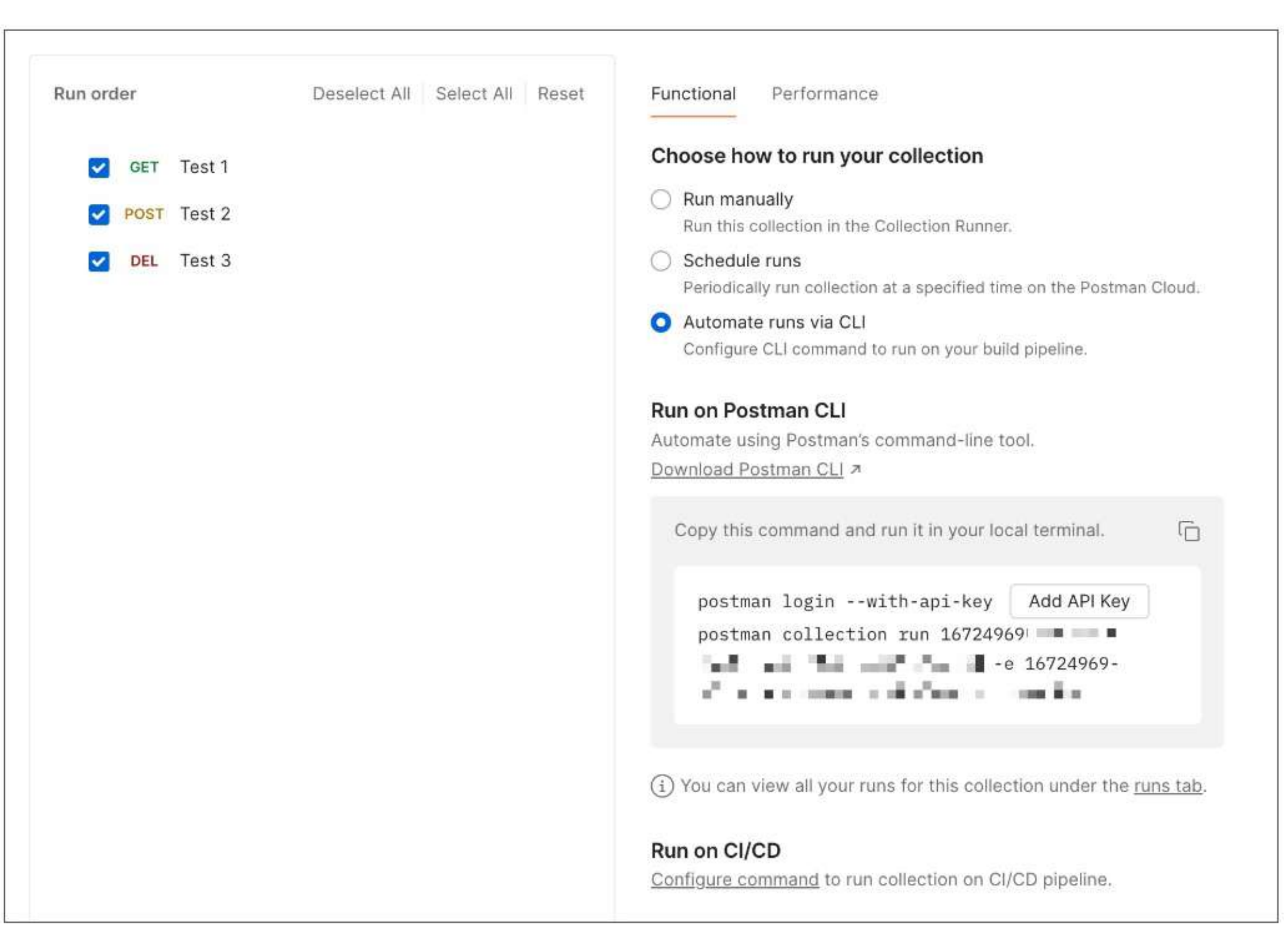
1. Download and install the Postman CLI.
2. Click **Collections** in the sidebar and select the collection or folder you want to run.

   > You can also run a collection or folder that's linked to an API. Learn more about adding a collection to an API in the Postman API Builder.

3. Click ▶ **Run**.

4. On the **Functional** tab, click **Automate runs via CLI**.

   Run order Deselect All Select All Reset

   ☑ GET Test 1
   ☑ POST Test 2
   ☑ DEL Test 3

   Functional Performance

   Choose how to run your collection

   ○ Run manually
   Run this collection in the Collection Runner.
   ○ Schedule runs
   Periodically run collection at a specified time on the Postman Cloud.
   ● Automate runs via CLI
   Configure CLI command to run on your build pipeline.

   Run on Postman CLI
   Automate using Postman's command-line tool.
   Download Postman CLI ↗

   Copy this command and run it in your local terminal.

   ```
   postman login --with-api-key  Add API Key
   postman collection run 16724969[redacted]
   [redacted] -e 16724969-
   [redacted]
   ```

   ⓘ You can view all your runs for this collection under the runs tab.

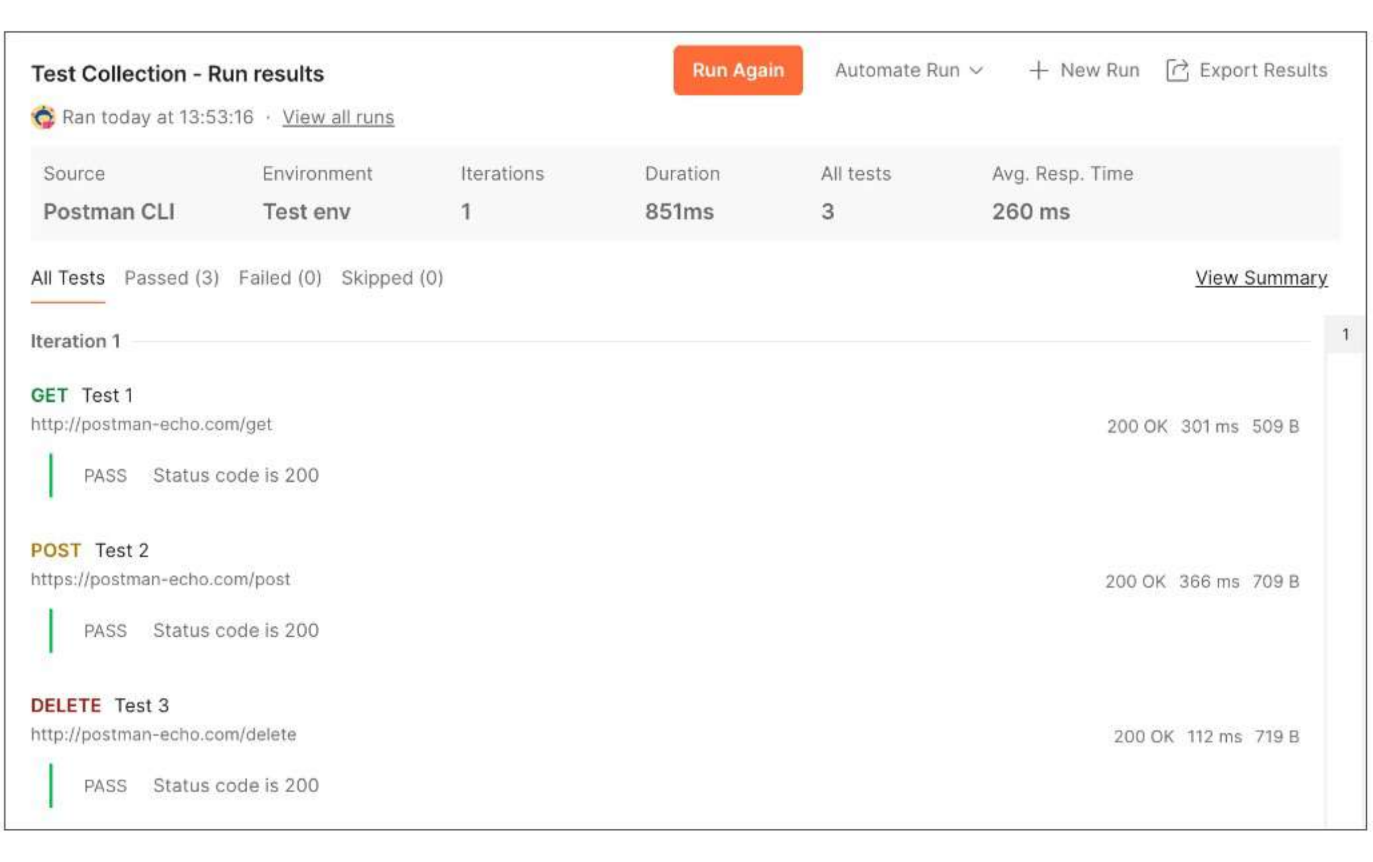
   Run on CI/CD
   Configure command to run collection on CI/CD pipeline.

   > If you're running a collection linked to an API that's connected to a Git repository, replace the collection ID with the path to the collection file in your remote repository.

5. Under **Run on Postman CLI**, click **Add API Key**. Do one of the following:
   - Click **Generate Key** to create a new API key. Enter a name for the API key and click **Generate**. Click **Copy** to copy the key and save it somewhere safe.
   - Click **Use Existing Key** and enter a valid API key.
6. Click **Insert Key**.
7. Click **Copy** to copy the commands.
8. Paste and run the commands in your terminal. After running the commands, the Postman CLI outputs a run report and a link to the run results in Postman.

   > In the run report, `test-scripts` refers to post-response scripts.

9. Follow the link to check the results in Postman.

   Test Collection - Run results
   Ran today at 13:53:16 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Postman CLI	Test env	1	851ms	3	260 ms

   All Tests Passed (3) Failed (0) Skipped (0)

   Iteration 1

   GET Test 1
   http://postman-echo.com/get 200 OK 301 ms 509 B
   PASS Status code is 200

   POST Test 2
   https://postman-echo.com/post 200 OK 366 ms 709 B
   PASS Status code is 200

   DELETE Test 3
   http://postman-echo.com/delete 200 OK 112 ms 719 B
   PASS Status code is 200

Run a collection in CI/CD

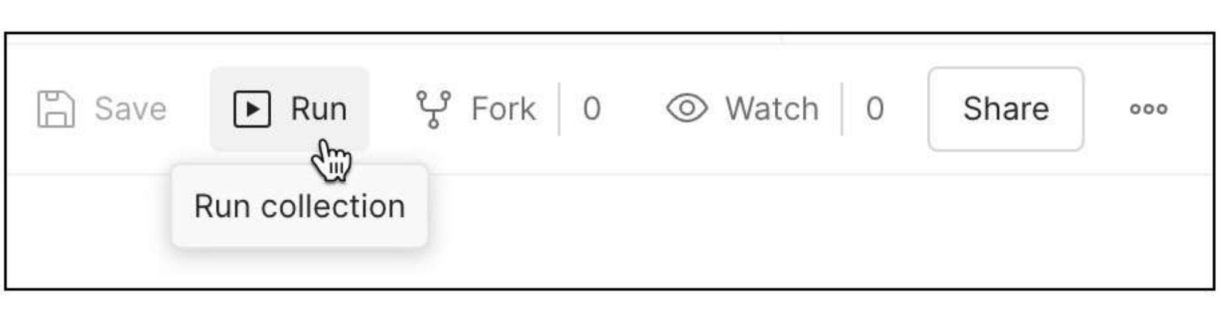
When the collection runs to your satisfaction, you can copy the commands into your CI/CD script to integrate them into your workflows. When adding the commands to your CI/CD script, you may want to replace the API key with a variable.

To run a collection in CI/CD, do the following:

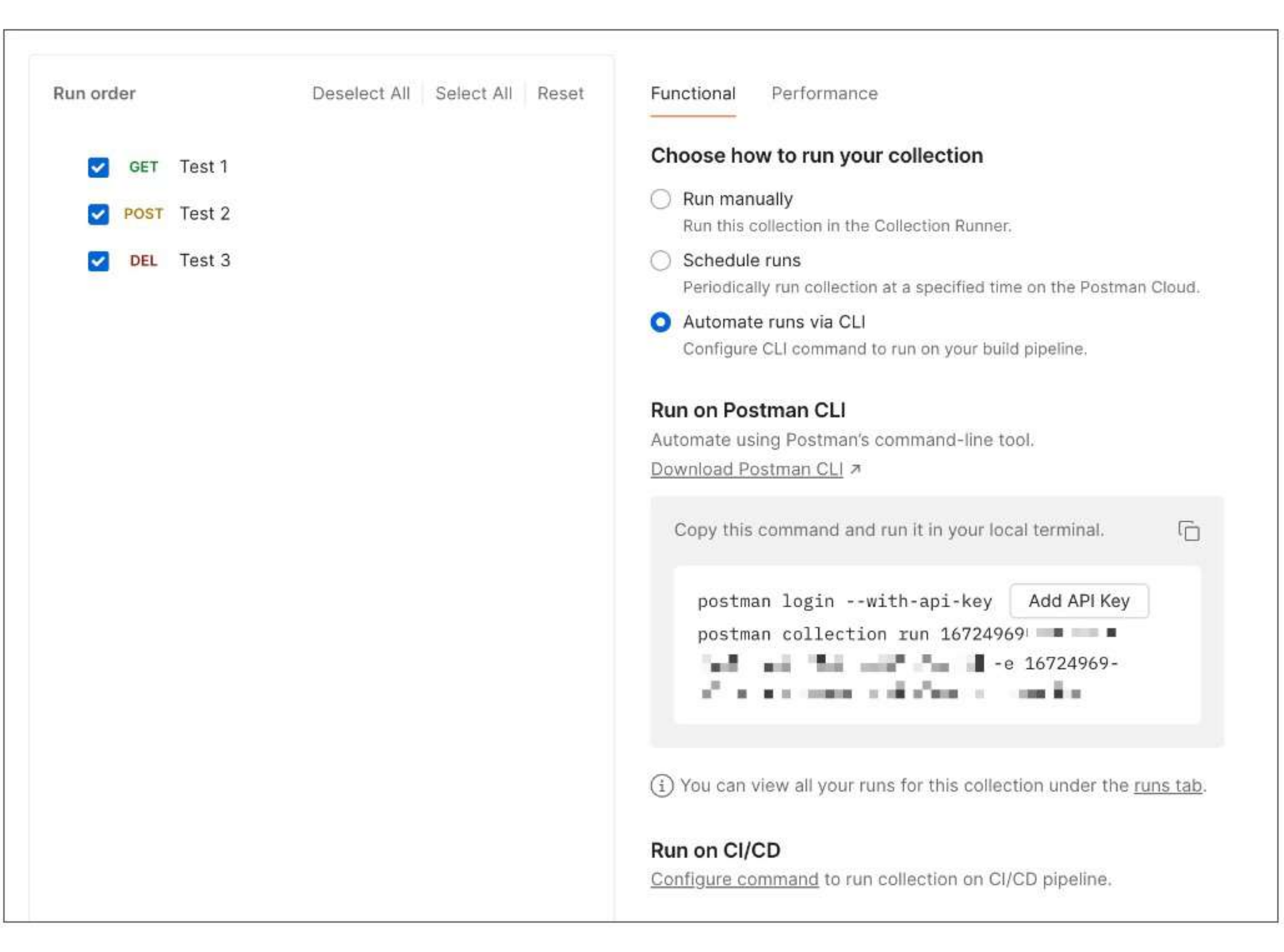
1. Click **Collections** in the sidebar and select the collection or folder you want to run.

   > You can also run a collection or folder that's linked to an API. Learn more about adding a collection to an API in the Postman API Builder.

2. Click ▶ **Run**.

3. On the **Functional** tab, click **Automate runs via CLI**.

   Run order Deselect All Select All Reset

   ☑ GET Test 1
   ☑ POST Test 2
   ☑ DEL Test 3

   Functional Performance

   Choose how to run your collection

   ○ Run manually
   Run this collection in the Collection Runner.
   ○ Schedule runs
   Periodically run collection at a specified time on the Postman Cloud.
   ● Automate runs via CLI
   Configure CLI command to run on your build pipeline.

   Run on Postman CLI
   Automate using Postman's command-line tool.
   Download Postman CLI ↗

   Copy this command and run it in your local terminal.

   ```
   postman login --with-api-key  Add API Key
   postman collection run 16724969[redacted]
   [redacted] -e 16724969-
   [redacted]
   ```

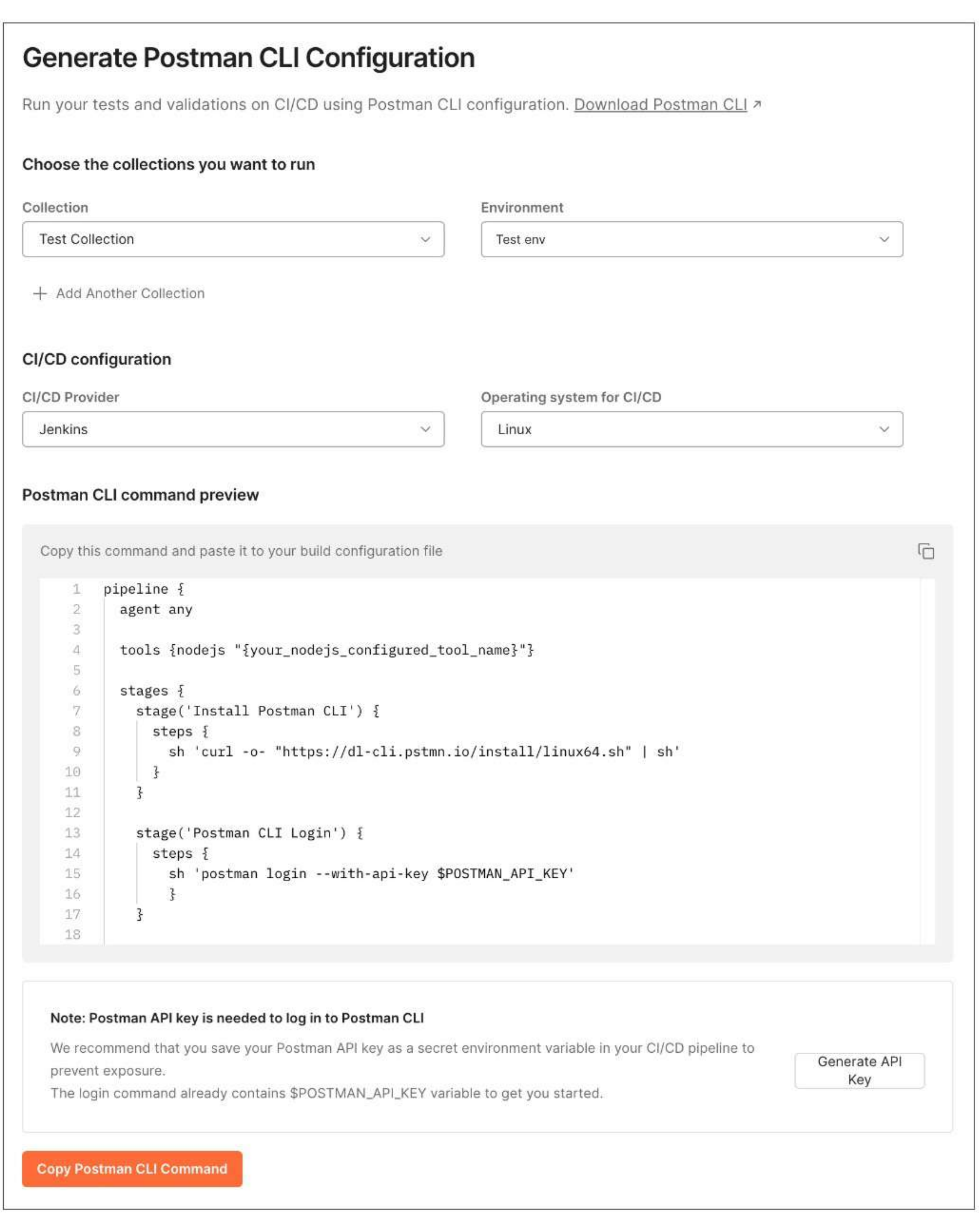
   ⓘ You can view all your runs for this collection under the runs tab.

   Run on CI/CD
   Configure command to run collection on CI/CD pipeline.

4. Under **Run on CI/CD**, click **Configure command**.
5. Click a **Collection** to run during pipeline builds. You can also select an **Environment** to use.

   > If needed, click + **Add Another Collection** to select more collections to run.

6. Click the **CI/CD Provider** and **Operating system** for your CI/CD pipeline.
7. To copy the Postman CLI configuration, click **Copy** or **Copy Postman CLI Command**.

   Generate Postman CLI Configuration

   Run your tests and validations on CI/CD using Postman CLI configuration. Download Postman CLI ↗

   Choose the collections you want to run

   Collection: Test Collection
   Environment: Test env

   + Add Another Collection

   CI/CD configuration

   CI/CD Provider: Jenkins
   Operating system for CI/CD: Linux

   Postman CLI command preview

   Copy this command and paste it to your build configuration file

   ```
   pipeline {
     agent any

     tools {nodejs "{your_nodejs_configured_tool_name}"}

     stages {
       stage('Install Postman CLI') {
         steps {
           sh 'curl -o- "https://dl-cli.pstmn.io/install/linux64.sh" | sh'
         }
       }

       stage('Postman CLI Login') {
         steps {
           sh 'postman login --with-api-key $POSTMAN_API_KEY'
           }
       }

   ```

   Note: Postman API key is needed to log in to Postman CLI

   We recommend that you save your Postman API key as a secret environment variable in your CI/CD pipeline to prevent exposure.
   The login command already contains $POSTMAN_API_KEY variable to get you started.

8. Add the Postman CLI configuration to your CI/CD script. This process depends on your CI tool.

Run a collection in a specific order

By default, when you generate the command to run a collection from the Collection Runner, a single Collection ID for the collection is specified. This runs the folders and requests in that collection in the sequence they're listed in the collection.

If you need to change the request order, select a request in the Collection Runner and drag it to its new location in the order. You can also remove an individual request from the run by clearing the checkbox next to its name.

When you change the folder and request sequence, the generated command also changes. In addition to the Collection ID, the generated command specifies a number of folder and request UIDs with the `-i` option. This runs each of the folders or requests in that specified order.

Run a collection that uses test data files

The Postman CLI can't run requests that use files in your local working directory to send body data. If your collection has requests that use files, upload your test data files to make them available to the Postman CLI.

Run a collection that uses packages

> Using the Postman CLI to run packages from your team's Package Library is available with Postman Professional and Enterprise plans.

You can use the Postman CLI to run collections with scripts that import packages from your team's Package Library. Learn how to add packages to the Package Library, and import packages into your scripts.

You can also use the Postman CLI to run collections that import external packages from npm or JSR package registries.

Generate collection run reports using the Postman CLI

The Postman CLI has built-in reporters you can use to generate reports for your collection runs. The following reporters are available: CLI, JSON, JUnit, and HTML. You can use more than one reporter for a run, and you can customize the report output to meet your needs.

About built-in reporters

Each Postman CLI reporter generates a local report with details about a collection run, including the requests sent, response codes and times, and the number of passed and failed tests. JSON and HTML reports also include details about failed requests ran from scripts at the collection and folder levels. The following reporters are available:

- **CLI** - Displays a report in the terminal. The CLI report is displayed by default if no reporter options are specified.
- **JSON** - Creates a JSON file containing the report. By default, the report structure follows the Postman CLI JSON reporter schema. You can optionally generate a report using the Newman JSON reporter schema (example).
- **JUnit** - Creates an XML file containing the report. To learn more about JUnit reporting, see the JUnit documentation.
- **HTML** - Creates an HTML file containing the report. You can filter your collection run iterations by test failures or errors, or by test failures only. Then select the iteration you'd like to view from the filtered results. You can also filter the requests within each iteration to show requests that encountered test failures or errors, or errors only.

Use a built-in reporter

To generate a report for a collection run, use the `-r` or `--reporters` option. Then specify the reporter you want to use to generate the report: `cli`, `json`, `junit`, or `html`. Use the following syntax:

```
postman collection run <collection-id> -r
<reporter>
```

Copy

> You can find the collection ID in Postman. Click **Collections** in the sidebar and select a collection. Then click **Info** ⓘ in the right sidebar to view or copy the collection ID.

By default, reports are created inside the directory `postman-cli-reports` in the current working directory. If the `postman-cli-reports` directory doesn't exist, it's automatically created. You can also specify an output directory. The filename includes the collection name and a system timestamp in 24-hour format: `collection-name-yyyy-mm-dd-hh-mm-ss`.

Use more than one reporter

You can specify one or more reporters for a collection run. If you specify more than one reporter, separate the reporter names in a comma-separated list, for example, `-r json,junit`.

By default, CLI reporter output is shown in the terminal when you run a collection. If you specify one or more reporters, for example `-r json`, then CLI reporter output isn't shown. In this case, to show the CLI reporter, you must explicitly specify it along with the other reporters, for example: `-r cli,json`.

The following example runs the `cli` and `json` reporters:

```
postman collection run <collection-id> -r cli,json
```

Copy

Configure built-in reporters

You can customize the output of each reporter by including one or more options when running the command. Use the following syntax:

```
postman collection run <collection-id> -r json <--
reporter-json-option1> <--reporter-json-option2>
```

Copy

The following example runs the `json` reporter with the omit headers and omit response bodies options:

```
postman collection run <collection-id> -r json --
reporter-json-omitHeaders --reporter-json-
omitResponseBodies
```

Copy

Configure the CLI reporter

The CLI reporter is turned on by default when running a collection with the Postman CLI and prints the report to the terminal. Use the following options to configure the CLI reporter.

Option	Details
`--reporter-cli-silent`	Turns off the CLI reporter, and no output is displayed in the terminal.
`--reporter-cli-show-timestamps`	Prints the local time that each request was made.
`--reporter-cli-no-summary`	Doesn't print the statistical summary table.
`--reporter-cli-no-failures`	Doesn't print details for run failures.
`--reporter-cli-no-assertions`	Turns off output for assertions as they happen.
`--reporter-cli-no-success-assertions`	Turns off output for successful assertions as they happen.
`--reporter-cli-no-console`	Turns off `console.log()` output (and other console methods) from pre-request and post-response scripts.
`--reporter-cli-no-banner`	Turns off the banner shown at the beginning of each collection run.

Configure the JSON, JUnit, and HTML reporters

The JSON, JUnit, and HTML reporters create a report file in the specified format in your working directory. Use the following options to configure the reporter. Replace `<reporter>` with the reporter you are using: `json`, `junit`, or `html`.

Option	Details
`--reporter-<reporter>-export <path>`	Specify a path to save the report. By default, reports are saved to the `/postman-cli-reports` directory in your current working directory. If the directory doesn't exist, it will be created automatically. If the specified path is an existing directory, the report file will be saved within it.
`--reporter-<reporter>-omitRequestBodies`	(JSON and HTML reporters only) Remove all request bodies from the report.
`--reporter-<reporter>-omitResponseBodies`	(JSON and HTML reporters only) Remove all response bodies from the report.
`--reporter-<reporter>-omitHeaders`	(JSON and HTML reporters only) Remove all request and response headers from the report.
`--reporter-<reporter>-omitAllHeadersAndBody`	(JSON and HTML reporters only) Remove all request and response headers, and all request and response bodies, from the report.
`--reporter-json-structure newman`	(JSON reporter only) Generate a JSON report using the Newman schema. By default, JSON reports use the native structure of the Postman CLI.

Configure options for more than one reporter

If you're using more than one reporter, you can specify an option for a single reporter using the syntax `--reporter-<reporter>-<option>`. Replace `<reporter>` with the reporter you are using: `cli`, `json`, `junit`, or `html`. Replace `<option>` with the reporter option you want to apply to the reporter.

```
postman collection run <collection-id> -r json,html
--reporter-json-omitHeaders
```

Copy

If you want all reporters to accept the same option, you can specify a reporter option for all reporters using the syntax `--reporter-[option]`.

```
postman collection run <collection-id> -r json,html
--reporter-omitHeaders
```

Copy

> When using more than one reporter, options specified for a single reporter take precedence over options specified for all reporters.

Install the Postman CLI

Download and install the Postman CLI by following the instructions for your operating system below.

> The Postman CLI supports the same system requirements as the Postman desktop app. For a complete list of requirements, see Install and update Postman.

Windows installation

Run the following command to install the Postman CLI for Windows. This will download and run an install script. The script creates a `%USERPROFILE%\AppData\Local\Microsoft\WindowsApps` directory if it doesn't exist yet, then installs the `postman` binary there.

```
powershell.exe -NoProfile -InputFormat None -
ExecutionPolicy AllSigned -Command "
[System.Net.ServicePointManager]::SecurityProtocol =
3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://dl-
cli.pstmn.io/install/win64.ps1'))"
```

Copy

macOS (Apple silicon) installation

Run the following command to install the Postman CLI for Macs with an Apple silicon processor. This will download an install script and run it. The install script creates a `/usr/local/bin` directory if it doesn't exist yet, then installs the `postman` binary there.

```
curl -o- "https://dl-
cli.pstmn.io/install/osx_arm64.sh" | sh
```

Copy

macOS (Intel) installation

Run the following command to install the Postman CLI for Macs with an Intel processor. This will download an install script and run it. The install script creates a `/usr/local/bin` directory if it doesn't exist yet, then installs the `postman` binary there.

```
curl -o- "https://dl-
cli.pstmn.io/install/osx_64.sh" | sh
```

Copy

Linux installation

Run the following command to install the Postman CLI for the latest Linux version. This will download an install script and run it. The install script creates a `/usr/local/bin` directory if it doesn't exist yet, then installs the `postman` binary there.

```
curl -o- "https://dl-
cli.pstmn.io/install/linux64.sh" | sh
```

Copy

Update your CLI installation

To update your Postman CLI installation to the latest version, run the same command you used to install it. The new version will overwrite the earlier version. For details about the latest changes and features, see the Postman CLI release notes.

CLI release notes.

Postman CLI command options

This topic covers the commands and options supported by the Postman CLI.

Basic command line options

`postman`

The base command of the Postman CLI is `postman`. Run this command with the syntax and options detailed below.

Options

Option	Details
`--help`, `-h`	Returns information about Postman CLI commands and options.
`--version`, `-v`	Returns the version number for the Postman CLI.

Example

```
postman -v
```

Sign in and out

You can use the Postman CLI to sign in and out of Postman with the `login` and `logout` commands.

`postman login`

This command authenticates the user and locally caches the Postman API key. The `login` command requires one option, `--with-api-key`, that accepts the Postman API key. Signing in is required once per session. You remain signed in until you use the `logout` command or your Postman API key expires.

Options

Option	Details
`--with-api-key <api-key>`	Authenticate the user with the given API key.

Example

```
postman login --with-api-key ABCD-1234-1234-1234-1234-1234
```

`postman logout`

This command signs you out of Postman and deletes the stored API key.

Example

```
postman logout
```

Run collections

> Postman supports running HTTP collections. You can't run multi-protocol collections.

You can run your collections with the `postman collection run` command:

`postman collection run`

This command runs a collection and sends the run results to the Postman cloud. You can specify the collection with its file path or Collection ID.

> You can find the collection ID in Postman. Click **Collections** in the sidebar and select a collection. Then click **Info** ⓘ in the right sidebar to view or copy the collection ID.

Options

Option	Details
`--bail [optional modifiers]`	Specifies whether to stop a collection run on encountering the first error. `--bail` can optionally accept two modifiers: `--folder` and `--failure`. `--folder` skips the entire collection run if there are any errors. If a test fails, `--failure` stops the collection run after completing the current script.
`--color [value]`	Controls colors in the CLI output. Accepts `on`, `off`, and `auto`. The default is `auto`. With `auto`, the Postman CLI attempts to automatically turn color on or off based on the color support in the terminal. This behavior can be changed by using the `on` or `off` value.
`--cookie-jar [path]`	Specifies the file path for a `JSON` cookie jar. This uses the `tough-cookie` library to deserialize the file.
`--delay-request [number]`	Specifies a delay (in milliseconds) between requests. The default is 0.
`--disable-unicode`	Replaces all symbols in the output with their plain text equivalents.
`--environment [UID] or [file-path]`, `-e`	Specifies an environment file path or UID.
`--env-var "[environment-variable-name]=[environment-variable-value]"`	Specifies environment variables in a `key=value` format. Multiple CLI environment variables can be added by using `--env-var` multiple times, for example: `--env-var "this=that" --env-var "alpha=beta"`.
`--export-cookie-jar [path]`	Specifies the path where the Postman CLI will output the final cookie jar file after completing a run. This uses the `tough-cookie` library to serialize the file.
`--global-var "[global-variable-name]=[global-variable-value]"`	Specifies global variables in a `key=value` format. Multiple CLI global variables can be added by using `--global-var` multiple times, for example: `--global-var "this=that" --global-var "alpha=beta"`.
`--globals [file-path]`, `-g`	Specifies a path to a file containing global variables. Global variables are similar to environment variables but have lower precedence and can be overridden by environment variables having the same name.
`--integration-id [ID]`	Specifies an integration ID when using an integration with CI/CD. This sends the Postman CLI results to the correct integration in Postman. When you generate the Postman CLI command for a Git integrated API collection, the integration ID is automatically added to the command. To learn how to generate the Postman CLI command with the integration ID, go to Configure the Postman CLI for CI.
`--iteration-count [number]`, `-n`	Specifies the number of times the collection will run when used in conjunction with the iteration data file.
`--iteration-data [file-path]`, `-d`	Specifies the local file path to a data file (JSON or CSV) to use for each iteration.
`-i [requestUID] or [folderUID]`	Runs only the specified folder UID or request UID from the collection. Multiple items can be run in order by specifying `-i` multiple times, for example: `postman collection run collectionUID -i folder1UID -i folder2UID`.
`-i [requestName] or [folderName]`	Runs only the specified folder name or request name from the collection. If there are duplicate names, the Postman CLI runs the folder or request that appears first.
`--ignore-redirects`	Prevents the Postman CLI from automatically following 3XX redirect responses.
`--insecure`, `-k`	Turns off SSL verification checks and enables self-signed SSL certificates.
`--no-insecure-file-read`	Prevents reading of files situated outside of the working directory.
`--silent`	Turns off terminal output.
`--ssl-client-cert-list <path>`	Specifies the path to a client certificates configuration (JSON).
`--ssl-client-cert <path>`	Specifies the path to a client certificate (PEM).
`--ssl-client-key <path>`	Specifies the path to a client certificate private key.
`--ssl-client-passphrase <passphrase>`	Specifies the client certificate passphrase (for a protected key).
`--ssl-extra-ca-certs <path>`	Specifies more trusted CA certificates (PEM).
`--suppress-exit-code`, `-x`	Specifies whether to override the default exit code for the current run.
`--timeout [number]`	Specifies the time (in milliseconds) to wait for the entire collection run to complete.
`--timeout-request [number]`	Specifies a time (in milliseconds) to wait for requests to return a response. The default is 0.
`--timeout-script [number]`	Specifies the time (in milliseconds) to wait for scripts to complete. The default is 0.
`--verbose`	Shows detailed information for the collection run and each request sent.
`--working-dir [path]`	Sets the path of the working directory to use while reading files with relative paths. This defaults to the current directory.
`--reporters [reporter]`, `-r [reporter]`	Generates a local report for the collection run in the specified format: `cli`, `json`, `junit`, and `html`. If the `-reporters` option isn't specified, the `cli` report is output by default. To learn more, go to Generate collection run reports using the Postman CLI.

Examples

```
postman collection run /myCollectionFolderName/myCollectionFile.json

postman collection run 12345678-12345ab-1234-1ab2-1ab2-ab1234112a12
```

Governance and security

> API governance and security rules are available with Postman Enterprise plans.

API governance is the practice of applying a defined set of standards consistently across the API design and testing phases of your development process. The Postman CLI includes commands that checks your API specifications in Spec Hub and the Postman API Builder against your team's configured Postman API governance and security rules.

`postman spec lint`

This command runs validation checks with governance rules against an API specification in Spec Hub. Provide the ID for a specification that's in OpenAPI 3.0 format.

> Click ⓘ **Specification Info** in the right sidebar of a specification and copy its ID.

Options

Option	Details
`--fail-severity [severity]`, `-f`	Triggers an exit failure code for rule violations at or higher than the specified severity level. The options, in order of lowest to highest severity, are `HINT`, `INFO`, `WARNING`, and `ERROR` (default).
`--output [output format]`, `-o`	Controls the output format for issues found in the OpenAPI 3.0 specification. Accepts `JSON` or `CSV`. Defaults to table view if no output format is specified. See examples of JSON and CSV output.

Example

```
postman spec lint 12345678-12345ab-1234-1ab2-1ab2-ab1234112a12
```

Example output

You can change the output of governance rule violations to JSON or CSV. If you don't specify an output, it defaults to table view.

The following is an example of the output in JSON format:

```
{
  "violations": [
    {
      "line number": "4",
      "path": "info",
      "severity": "WARNING",
      "issue": "The info object should have a description."
    },
    {
      "line number": "10",
      "path": "paths./spacecrafts/{spacecraftId}/",
      "severity": "ERROR",
      "issue": "There should be no trailing slashes on paths."
    },
    {
      "line number": "20",
      "path": "paths./spacecrafts/{spacecraftId}/.get.responses",
      "severity": "WARNING",
      "issue": "Operation should return a 2xx HTTP status code"
    },
    {
      "line number": "20",
      "path": "paths./spacecrafts/{spacecraftId}/.get.responses",
      "severity": "WARNING",
      "issue": "Operation should return a 5xx HTTP status code"
    }
  ]
}
```

The following is an example of the output in CSV format:

```
line number,path,severity,issue
4,info,WARNING,The info object should have a description.
10,paths./spacecrafts/{spacecraftId}/,ERROR,There should be no trailing slashes on paths.
20,paths./spacecrafts/{spacecraftId}/.get.responses,WARNING,Operation should return a 2xx HTTP status code
20,paths./spacecrafts/{spacecraftId}/.get.responses,WARNING,Operation should return a 5xx HTTP status code
```

`postman api lint`

This command runs validation checks for governance and security rules against the API specification provided in the Postman config file, a local file, or a UUID. The `api lint` command shows a warning if it's unable to find the API ID to send data back to Postman.

> This command supports APIs in the Postman API Builder and aren't linked to Git.

Options

Option	Details
`--fail-severity [severity]`, `-f`	Triggers an exit failure code for rule violations at or higher than the specified severity level. The options, in order of lowest to highest severity, are `HINT`, `INFO`, `WARN`, and `ERROR` (default).
`--suppress-exit-code`, `-x`	Specifies whether to override the default exit code for the current run.

Example

```
postman api lint my-definition-file.json
postman api lint 12345678-12345ab-1234-1ab2-1ab2-ab1234112a12
```

Publish an API version

You can publish API versions in the Postman API Builder from the command line with the Postman CLI. Use the Postman CLI to automate the API version publishing process.

`postman publish api`

Publish a snapshot of an API for the given `apiId`. All elements linked to the API are published by default. You can choose which elements to publish by using other command options.

When publishing an API that's linked with Git, you must enter the command from inside the local Git repo. Also, you must provide paths to the schema directory and collection paths instead of IDs.

Options

Option	Details
`--name <name>`	Specifies the name of the version to publish.
`--release-notes <releaseNotes>`	Enter release notes as a string in quotes for the version to publish. This option supports Markdown.
`--collections <collectionIds/paths...>`	Specifies the collections to publish. If the API is linked with Git, provide the `filePath` instead of the ID.
`--api-definition <apiDefinitionId/directory/file>`	Specifies the API specification to publish. If the API is linked with Git, provide the `schemaDirectoryPath` or `schemaRootFilePath` instead of the ID.
`--do-not-poll`	Specifies not to poll for completion status of the publish action.
`--suppress-exit-code, -x`	Specifies whether to override the default exit code for the current run.

Example for repos not linked with Git

```
postman api publish <apiId> --name v1\
--release-notes "# Some release notes information"\
--collections <collectionId1> <collectionId2>\
--api-definition <apiDefinitionId>
```

Examples for repos linked with Git

The options for the `api publish` command differ depending on if you specified a schema folder or schema root file when setting up the Git integration. Git integrations added in Postman v10.18 or later use a schema root file. Git integrations added in other Postman versions use a schema folder. Learn more about connecting an API in the Postman API Builder to a Git repository.

- If the API uses a schema folder, publish the API using the `--api-definition <schemaDirectoryPath>` option:

  ```
  postman api publish <apiId> --name v1\
  --release-notes "# Some release notes information"\
  --collections <collectionPath1> <collectionPath2>\
  --api-definition <schemaDirectoryPath>
  ```

- If the API uses a schema root file, publish the API using the `--api-definition <schemaRootFilePath>` option:

  ```
  postman api publish <apiId> --name v1\
  --release-notes "# Some release notes information"\
  --collections <collectionPath1> <collectionPath2>\
  --api-definition <schemaRootFilePath>
  ```

> If you specify a file when a folder is required, or a folder when a file is required, the `api publish` command returns the following error: `API Definition <file/folder> isn't part of API <apiId>`. Try the command again using the other option.

Run a collection using the Postman CLI

You can use the Postman CLI to manually run collections to test the functionality of your API. You can also use the Postman CLI to automate collection runs in CI/CD pipelines.

> Postman supports running HTTP collections. You can't run multi-protocol collections.

When the Postman CLI runs a collection, the collection and its tests run locally, and the results are sent to the Postman cloud using an API call.

> The Postman CLI doesn't support OAuth 2.0 authentication. To learn how to use an OAuth 2.0 token with the Postman CLI, see OAuth 2.0 overview.

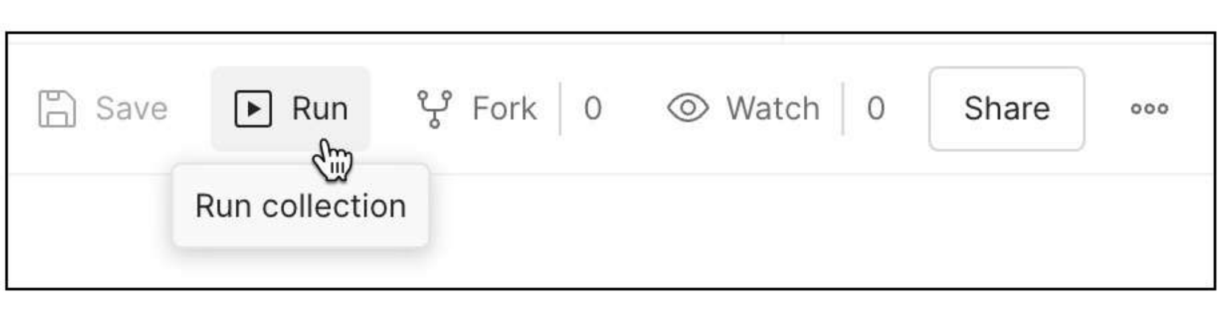
Run a collection locally with the Postman CLI

You can use the Postman CLI to run the requests in a collection or a folder.

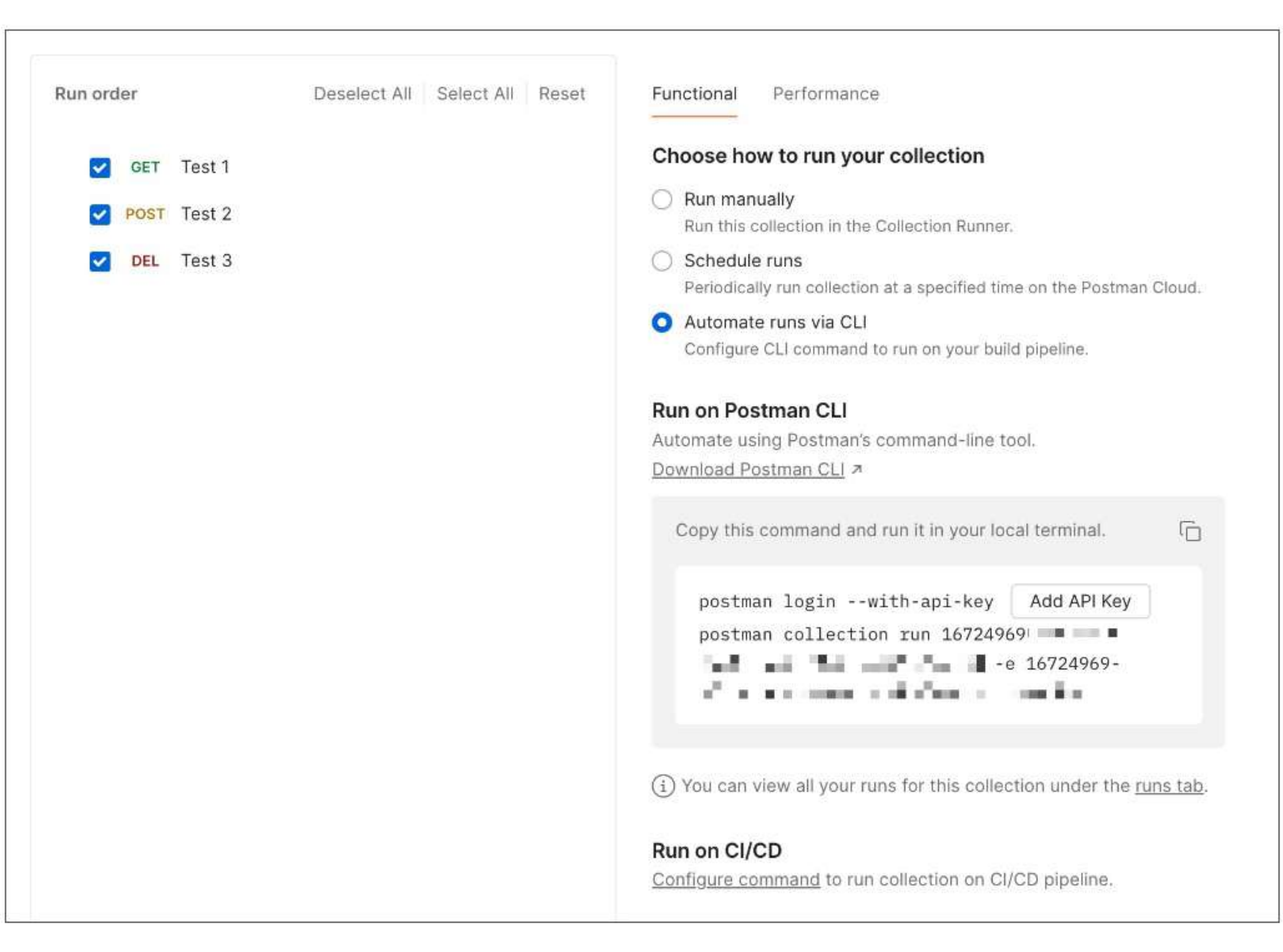
1. Download and install the Postman CLI.
2. Click **Collections** in the sidebar and select the collection or folder you want to run.

   > You can also run a collection or folder that's linked to an API. Learn more about adding a collection to an API in the Postman API Builder.

3. Click **Run**.



4. On the **Functional** tab, click **Automate runs via CLI**.



☑ GET Test 1
☑ POST Test 2
☑ DEL Test 3

Choose how to run your collection

○ Run manually
Run this collection in the Collection Runner.

○ Schedule runs
Periodically run collection at a specified time on the Postman Cloud.

● Automate runs via CLI
Configure CLI command to run on your build pipeline.

Run on Postman CLI
Automate using Postman's command-line tool.
Download Postman CLI ↗

Copy this command and run it in your local terminal.

```
postman login --with-api-key  Add API Key
postman collection run 16724969[redacted] -e 16724969-
[redacted]
```

ⓘ You can view all your runs for this collection under the runs tab.

Run on CI/CD
Configure command to run collection on CI/CD pipeline.

   > If you're running a collection linked to an API that's connected to a Git repository, replace the collection ID with the path to the collection file in your remote repository.

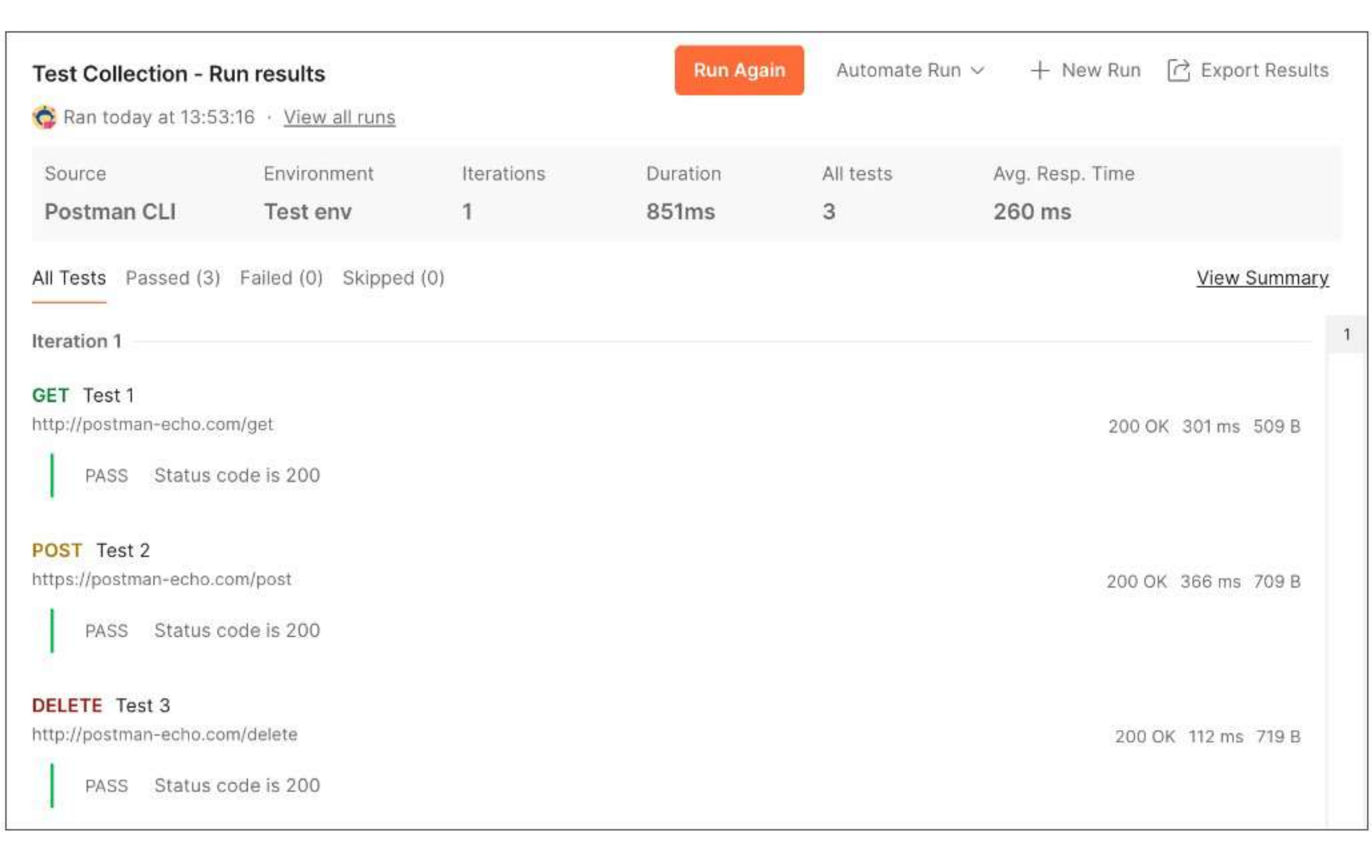
5. Under **Run on Postman CLI**, click **Add API Key**. Do one of the following:

   - Click **Generate Key** to create a new API key. Enter a name for the API key and click **Generate**. Click **Copy** to copy the key and save it somewhere safe.
   - Click **Use Existing Key** and enter a valid API key.

6. Click **Insert Key**.
7. Click **Copy** to copy the commands.
8. Paste and run the commands in your terminal. After running the commands, the Postman CLI outputs a run report and a link to the run results in Postman.

   > In the run report, `test-scripts` refers to post-response scripts.

9. Follow the link to check the results in Postman.



Test Collection - Run results
Ran today at 13:53:16 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Postman CLI	Test env	1	851ms	3	260 ms

All Tests Passed (3) Failed (0) Skipped (0)

Iteration 1

GET Test 1
http://postman-echo.com/get — 200 OK 301 ms 509 B
PASS Status code is 200

POST Test 2
https://postman-echo.com/post — 200 OK 366 ms 709 B
PASS Status code is 200

DELETE Test 3
http://postman-echo.com/delete — 200 OK 112 ms 719 B
PASS Status code is 200

Run a collection in CI/CD

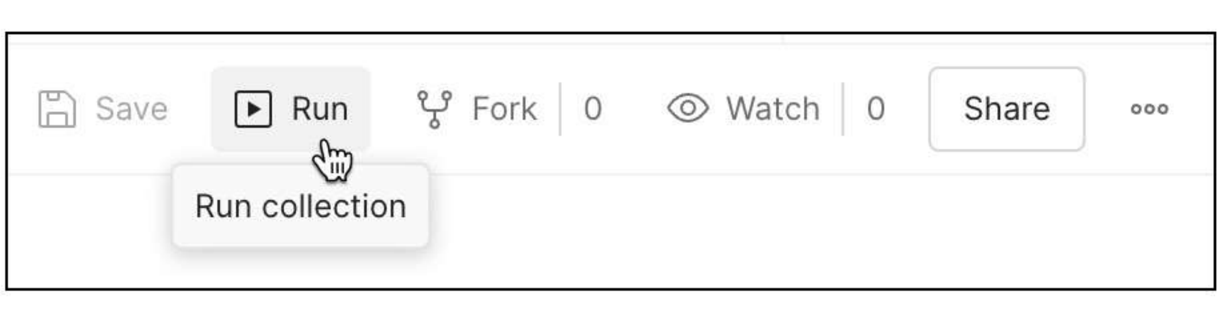
When the collection runs to your satisfaction, you can copy the commands into your CI/CD script to integrate them into your workflows. When adding the commands to your CI/CD script, you may want to replace the API key with a variable.

To run a collection in CI/CD, do the following:

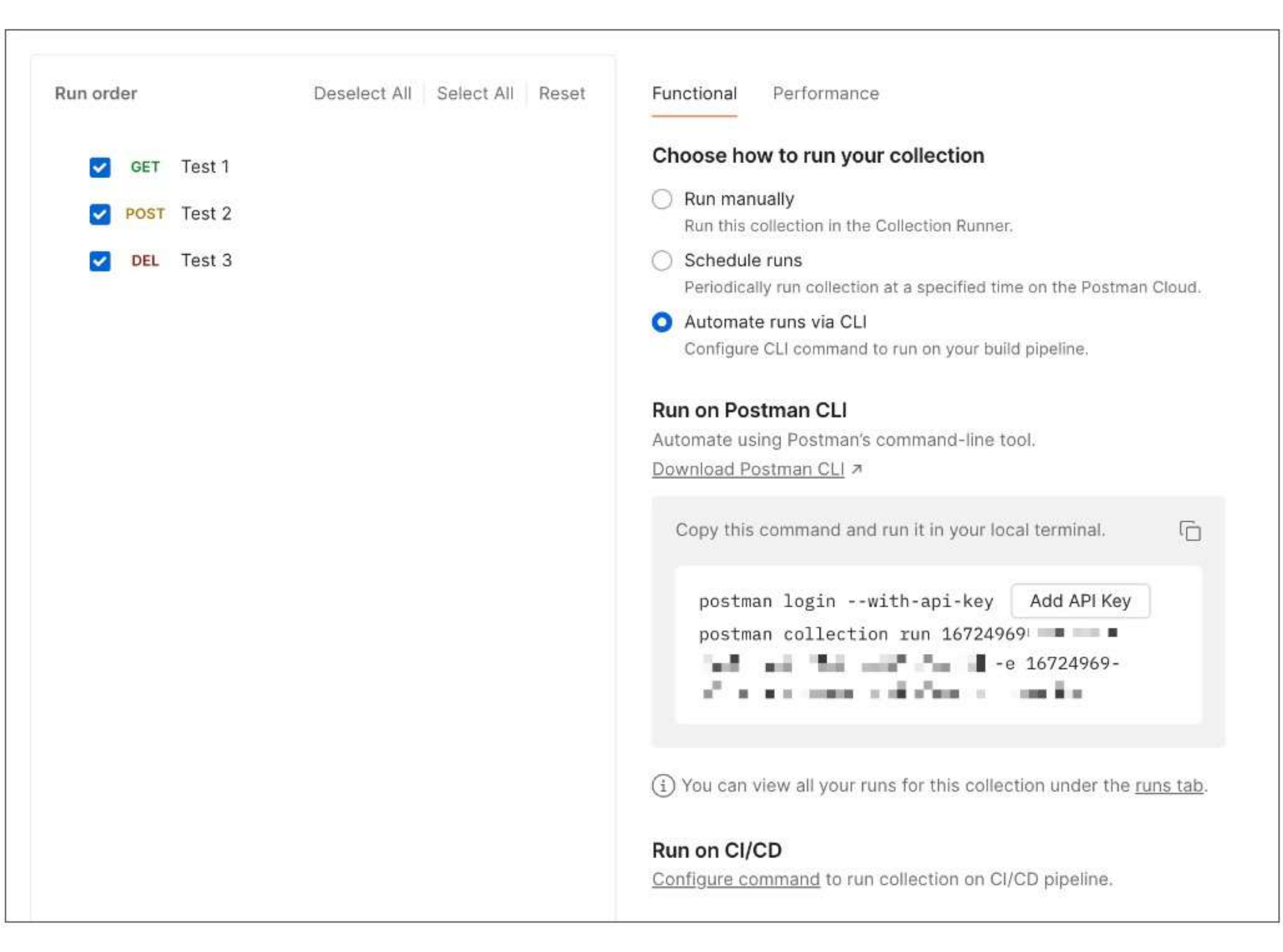
1. Click **Collections** in the sidebar and select the collection or folder you want to run.

   > You can also run a collection or folder that's linked to an API. Learn more about adding a collection to an API in the Postman API Builder.

2. Click **Run**.



3. On the **Functional** tab, click **Automate runs via CLI**.



☑ GET Test 1
☑ POST Test 2
☑ DEL Test 3

Choose how to run your collection

○ Run manually
Run this collection in the Collection Runner.

○ Schedule runs
Periodically run collection at a specified time on the Postman Cloud.

● Automate runs via CLI
Configure CLI command to run on your build pipeline.

Run on Postman CLI
Automate using Postman's command-line tool.
Download Postman CLI ↗

Copy this command and run it in your local terminal.

```
postman login --with-api-key  Add API Key
postman collection run 16724969[redacted] -e 16724969-
[redacted]
```

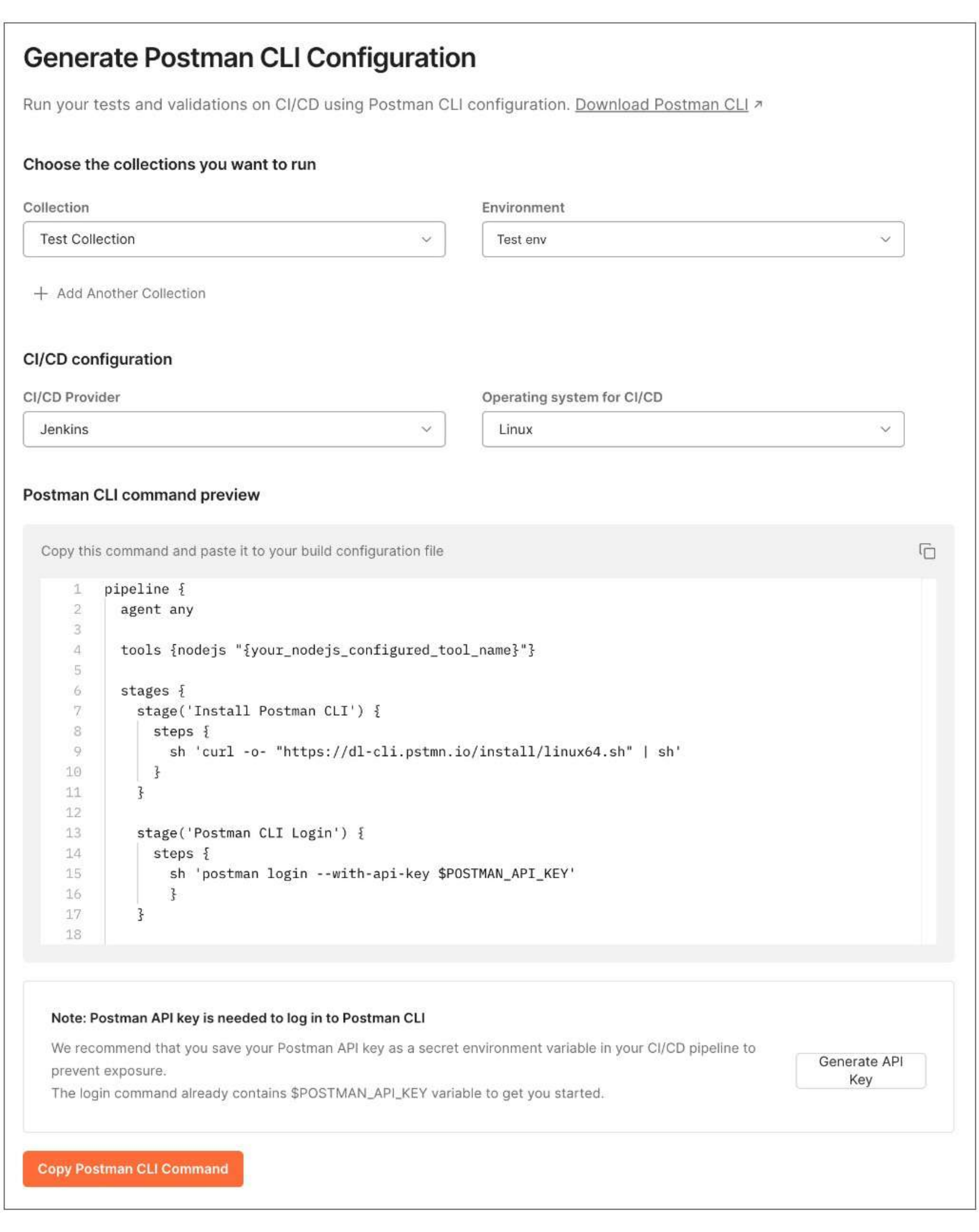
ⓘ You can view all your runs for this collection under the runs tab.

Run on CI/CD
Configure command to run collection on CI/CD pipeline.

4. Under **Run on CI/CD**, click **Configure command**.
5. Click a **Collection** to run during pipeline builds. You can also select an **Environment** to use.

   > If needed, click + **Add Another Collection** to select more collections to run.

6. Click the **CI/CD Provider** and **Operating system** for your CI/CD pipeline.
7. To copy the Postman CLI configuration, click **Copy** or **Copy Postman CLI Command**.



Generate Postman CLI Configuration
Run your tests and validations on CI/CD using Postman CLI configuration. Download Postman CLI ↗

Choose the collections you want to run

Collection: Test Collection
Environment: Test env

CI/CD configuration

CI/CD Provider: Jenkins
Operating system for CI/CD: Linux

Postman CLI command preview

Copy this command and paste it to your build configuration file

```
pipeline {
  agent any

  tools {nodejs "{your_nodejs_configured_tool_name}"}

  stages {
    stage('Install Postman CLI') {
      steps {
        sh 'curl -o- "https://dl-cli.pstmn.io/install/linux64.sh" | sh'
      }
    }

    stage('Postman CLI Login') {
      steps {
        sh 'postman login --with-api-key $POSTMAN_API_KEY'
        }
    }

```

Note: Postman API key is needed to log in to Postman CLI
We recommend that you save your Postman API key as a secret environment variable in your CI/CD pipeline to prevent exposure.
The login command already contains $POSTMAN_API_KEY variable to get you started.

8. Add the Postman CLI configuration to your CI/CD script. This process depends on your CI tool.

Run a collection in a specific order

By default, when you generate the command to run a collection from the Collection Runner, a single Collection ID for the collection is specified. This runs the folders and requests in that collection in the sequence they're listed in the collection.

If you need to change the request order, select a request in the Collection Runner and drag it to its new location in the order. You can also remove an individual request from the run by clearing the checkbox next to its name.

When you change the folder and request sequence, the generated command also changes. In addition to the Collection ID, the generated command specifies a number of folder and request UIDs with the `-i` option. This runs each of the folders or requests in that specified order.

Run a collection that uses test data files

The Postman CLI can't run requests that use files in your local working directory to send body data. If your collection has requests that use files, upload your test data files to make them available to the Postman CLI.

Run a collection that uses packages

> Using the Postman CLI to run packages from your team's Package Library is available with Postman Professional and Enterprise plans.

You can use the Postman CLI to run collections with scripts that import packages from your team's Package Library. Learn how to add packages to the Package Library, and import packages into your scripts.

You can also use the Postman CLI to run collections that import external packages from npm or JSR package registries.

Generate collection run reports using the Postman CLI

The Postman CLI has built-in reporters you can use to generate reports for your collection runs. The following reporters are available: CLI, JSON, JUnit, and HTML. You can use more than one reporter for a run, and you can customize the report output to meet your needs.

About built-in reporters

Each Postman CLI reporter generates a local report with details about a collection run, including the requests sent, response codes and times, and the number of passed and failed tests. JSON and HTML reports also include details about failed requests ran from scripts at the collection and folder levels. The following reporters are available:

- **CLI** - Displays a report in the terminal. The CLI report is displayed by default if no reporter options are specified.
- **JSON** - Creates a JSON file containing the report. By default, the report structure follows the Postman CLI JSON reporter schema. You can optionally generate a report using the Newman JSON reporter schema (example).
- **JUnit** - Creates an XML file containing the report. To learn more about JUnit reporting, see the JUnit documentation.
- **HTML** - Creates an HTML file containing the report. You can filter your collection run iterations by test failures or errors, or by test failures only. Then select the iteration you'd like to view from the filtered results. You can also filter the requests within each iteration to show requests that encountered test failures or errors, or errors only.

Use a built-in reporter

To generate a report for a collection run, use the `-r` or `--reporters` option. Then specify the reporter you want to use to generate the report: `cli`, `json`, `junit`, or `html`. Use the following syntax:

```
postman collection run <collection-id> -r <reporter>
```

> You can find the collection ID in Postman. Click **Collections** in the sidebar and select a collection. Then click **Info** ⓘ in the right sidebar to view or copy the collection ID.

By default, reports are created inside the directory `postman-cli-reports` in the current working directory. If the `postman-cli-reports` directory doesn't exist, it's automatically created. You can also specify an output directory. The filename includes the collection name and a system timestamp in 24-hour format: `collection-name-yyyy-mm-dd-hh-mm-ss`.

Use more than one reporter

You can specify one or more reporters for a collection run. If you specify more than one reporter, separate the reporter names in a comma-separated list, for example, `-r json,junit`.

By default, CLI reporter output is shown in the terminal when you run a collection. If you specify one or more reporters, for example `-r json`, then CLI reporter output isn't shown. In this case, to show the CLI reporter, you must explicitly specify it along with the other reporters, for example: `-r cli,json`.

The following example runs the `cli` and `json` reporters:

```
postman collection run <collection-id> -r cli,json
```

Configure built-in reporters

You can customize the output of each reporter by including one or more options when running the command. Use the following syntax:

```
postman collection run <collection-id> -r json <--reporter-json-option1> <--reporter-json-option2>
```

The following example runs the `json` reporter with the omit headers and omit response bodies options:

```
postman collection run <collection-id> -r json --reporter-json-omitHeaders --reporter-json-omitResponseBodies
```

Configure the CLI reporter

The CLI reporter is turned on by default when running a collection with the Postman CLI and prints the report to the terminal. Use the following options to configure the CLI reporter.

Option	Details
`--reporter-cli-silent`	Turns off the CLI reporter, and no output is displayed in the terminal.
`--reporter-cli-show-timestamps`	Prints the local time that each request was made.
`--reporter-cli-no-summary`	Doesn't print the statistical summary table.
`--reporter-cli-no-failures`	Doesn't print details for run failures.
`--reporter-cli-no-assertions`	Turns off output for assertions as they happen.
`--reporter-cli-no-success-assertions`	Turns off output for successful assertions as they happen.
`--reporter-cli-no-console`	Turns off `console.log()` output (and other console methods) from pre-request and post-response scripts.
`--reporter-cli-no-banner`	Turns off the banner shown at the beginning of each collection run.

Configure the JSON, JUnit, and HTML reporters

The JSON, JUnit, and HTML reporters create a report file in the specified format in your working directory. Use the following options to configure the reporter. Replace `<reporter>` with the reporter you are using: `json`, `junit`, or `html`.

Option	Details
`--reporter-<reporter>-export <path>`	Specify a path to save the report. By default, reports are saved to the `/postman-cli-reports` directory in your current working directory. If the directory doesn't exist, it will be created automatically. If the specified path is an existing directory, the report file will be saved within it.
`--reporter-<reporter>-omitRequestBodies`	(JSON and HTML reporters only) Remove all request bodies from the report.
`--reporter-<reporter>-omitResponseBodies`	(JSON and HTML reporters only) Remove all response bodies from the report.
`--reporter-<reporter>-omitHeaders`	(JSON and HTML reporters only) Remove all request and response headers from the report.
`--reporter-<reporter>-omitAllHeadersAndBody`	(JSON and HTML reporters only) Remove all request and response headers, and all request and response bodies, from the report.
`--reporter-json-structure newman`	(JSON reporter only) Generate a JSON report using the Newman schema. By default, JSON reports use the native structure of the Postman CLI.

Configure options for more than one reporter

If you're using more than one reporter, you can specify an option for a single reporter using the syntax `--reporter-<reporter>-<option>`. Replace `<reporter>` with the reporter you are using: `cli`, `json`, `junit`, or `html`. Replace `<option>` with the reporter option you want to apply to the reporter.

```
postman collection run <collection-id> -r json,html --reporter-json-omitHeaders
```

If you want all reporters to accept the same option, you can specify a reporter option for all reporters using the syntax `--reporter-[option]`.

```
postman collection run <collection-id> -r json,html --reporter-omitHeaders
```

> When using more than one reporter, options specified for a single reporter take precedence over options specified for all reporters.