Unknown date | **Unknown author**

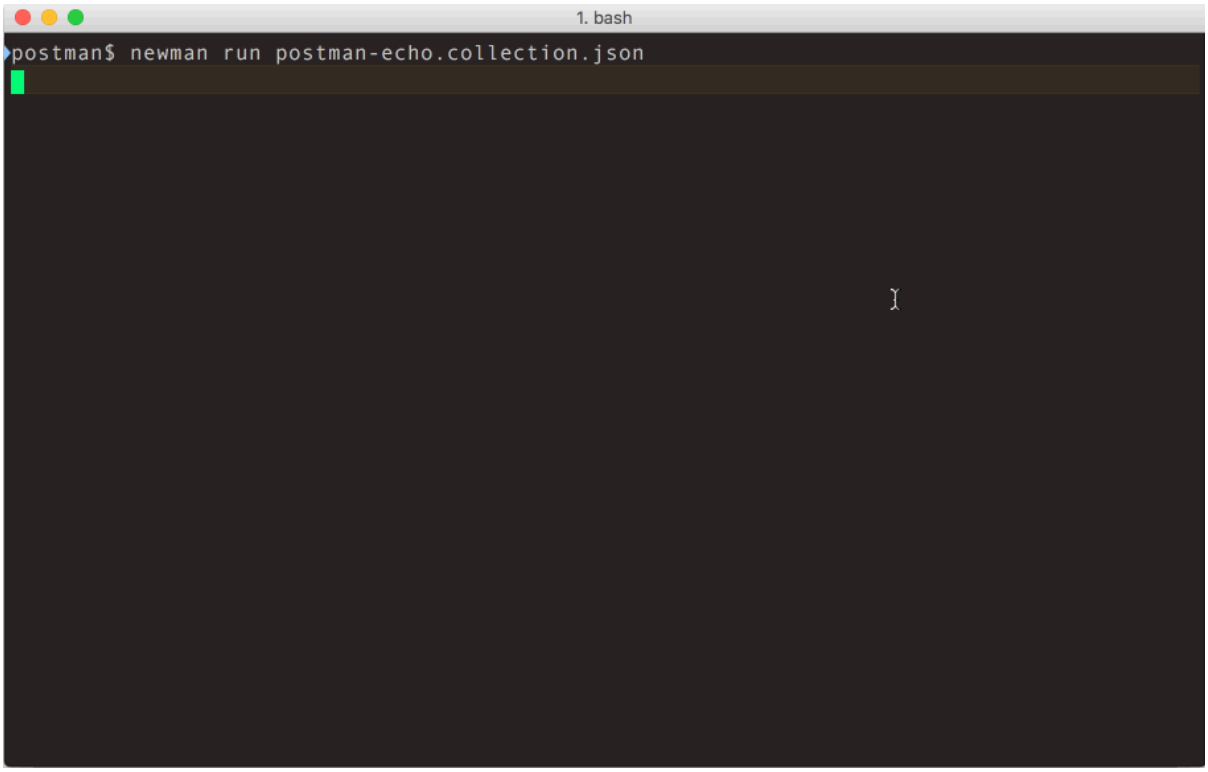
Command line integration with Newman

- Newman

Newman is a command line collection runner for Postman. It allows you to run and test a Postman Collection directly from the command line. It is built with extensibility in mind so that you can easily integrate it with your continuous integration servers and build systems.

Newman maintains feature parity with Postman and allows you to run collections just the way they are executed inside the collection runner in the Postman app.

Newman resides in the NPM registry and on GitHub.



Getting Started on Linux, Windows, or Mac

Newman is built on Node.js. To run Newman, make sure you have Node.js installed. Node.js can be downloaded and installed on Linux, Windows, and

Mac OSX.

Once Node.js is installed, Newman is just a command away. Install Newman from npm globally on your system allowing you to run it from anywhere.

```
$ npm install -g newman
```

The easiest way to run Newman is to run it with a collection. You can run any collection file from your file system. Refer to the collection documentation to learn how to export collections to share as a file.

```
$ newman run mycollection.json
```

You can also pass a collection as a URL. Refer to the collection documentation to learn how to share a file as a URL. Your collection probably uses environment variables. To provide an accompanying set of environment variables, export the template from Postman and run them with the -e flag.

```
$ newman run https://www.getpostman.com/collections/cb208e7e64056f5294e5 -e
```

Options

Newman provides a rich set of options to customize a run. A list of options can be retrieved by running it with the -h flag.

```
$ newman run -h
```

```
Options:

Utility:
-h, --help                      output usage information
-v, --version                   output the version number

Basic setup:
--folder [folderName]           Specify a single folder to run from a collection.
-e, --environment [file|URL]    Specify a Postman environment as a JSON [file]
-d, --data [file]               Specify a data file to use either json or csv
```

```
-g, --global [file]             Specify a Postman globals file as JSON [file]
-n, --iteration-count [number]  Define the number of iterations to run

Request options:
--delay-request [number]        Specify a delay (in ms) between requests [number]
--timeout-request [number]      Specify a request timeout (in ms) for a request

Misc.:
--bail                          Stops the runner when a test case fails
--silent                        Disable terminal output
--no-color                      Disable colored output
-k, --insecure                  Disable strict ssl
-x, --suppress-exit-code        Continue running tests even after a failure, but exit wi
--ignore-redirects              Disable automatic following of 3XX responses
```

Use the -n option to set the number of iterations to run the collection.

```
$ newman run mycollection.json -n 10  # runs the collection 10 times
```

To provide a different set of data, i.e. variables for each iteration, you can use the -d to specify a JSON or CSV file. For example, a data file such as the one shown below will run *2* iterations, with each iteration using a set of variables.

```
[{
    "url": "http://127.0.0.1:5000",
    "user_id": "1",
    "id": "1",
    "token_id": "123123",
},
{
    "url": "http://postman-echo.com",
    "user_id": "2",
    "id": "2",
    "token_id": "899899",
}]
```

```
$ newman run mycollection.json -d data.json
```

The CSV file for the above set of variables would look like:

```
url, user_id, id, token_id
http://127.0.0.1:5000, 1, 1, 123123123
http://postman-echo.com, 2, 2, 899899
```

Newman, by default, exits with a status code of 0 if everything runs well i.e. without any exceptions. Continuous integration tools respond to these exit codes and correspondingly pass or fail a build. You can use the ``–bailflag to tell Newman to halt on a test case error with a status code of 1 which can then be picked up by a CI tool or build system.

```
$ newman run PostmanCollection.json -e environment.json --bail newman
```

Example collection with failing tests

```
→ Status Code Test
  GET https://echo.getpostman.com/status/404 [404 Not Found, 534B, 1551ms]
  1\. response code is 200

┌─────────────────────────┬────────────────────┬───────────────────┐
│                         │ executed │ failed │
├─────────────────────────┼────────────────────┼───────────────────┤
│              iterations │ 1 │ 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│                requests │ 1 │ 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│            test-scripts │ 1 │ 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│      prerequest-scripts │ 0 │ 0 │
├─────────────────────────┼────────────────────┼───────────────────┤
│              assertions │ 1 │ 1 │
├─────────────────────────┴────────────────────┴───────────────────┤
│ total run duration: 1917ms                                       │
├──────────────────────────────────────────────────────────────────┤
│ total data received: 14B (approx)                                │
├──────────────────────────────────────────────────────────────────┤
```

```
| average response time: 1411ms                |
└──────────────────────────────────────────────┘


 #  failure        detail

1\.  AssertionFai...  response code is 200
                      at assertion:1 in test-script
                      inside "Status Code Test" of "Example Collection with
                      Failing Tests"
```

The results of all tests and requests can be exported into a file and later imported into Postman for further analysis. Use the JSON reporter and a file name to save the runner output into a file.

```
$ newman run mycollection.json --reporters cli,json --reporter-json-export outputfile
```

Note: Newman allows you to use all libraries and objects that Postman supports to run tests and pre-request scripts.

File uploads

Newman also supports file uploads. For this to work correctly, the file to be uploaded must be placed in the relative location specified within the collection. For instance, for the following collection:

```
{
	"variables": [],
	"info": {
		"name": "file-upload",
		"_postman_id": "9dbfcf22-fdf4-f328-e440-95dbd8e4cfbb",
		"description": "A set of `POST` requests to upload files as form data fiel
		"schema": "https://schema.getpostman.com/json/collection/v2.0.0/collec
	},
	"item": [
		{
			"name": "Form data upload",
			"event": [
```

```
                {
                    "listen": "test",
                    "script": {
                        "type": "text/javascript",
                        "exec": [
                            "var response = JSON.parse(responseBody)
                            "",
                            "tests[\"Status code is 200\"] = responseCo
                            "tests[\"File was uploaded correctly\"] = /^
                            ""
                        ]
                    }
                }
            ],
            "request": {
                "url": "https://echo.getpostman.com/post",
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "formdata",
                    "formdata": [
                        {
                            "key": "file",
                            "type": "file",
                            "enabled": true,
                            "src": "sample-file.txt"
                        }
                    ]
                },
                "description": "Uploads a file as a form data field to `https://
            },
            "response": []
        }
    ]
}
```

The file sample-file.txt must be present in the same directory as the collection. The collection can the be run as usual.

```
$ newman run file-upload.postman_collection.json
```

Library

Newman has been built as a library from the ground up so that it can be extended and used in various ways. You can use it as follows in your Node.js code:

```
var newman = require('newman'); // require newman in your project

// call newman.run to pass `options` object and wait for callback
newman.run({
    collection: require('./sample-collection.json'),
    reporters: 'cli'
}, function (err) {
        if (err) { throw err; }
    console.log('collection run complete!');
});
```

Custom reporters

Custom reporters come in handy when one would want to generate collection run reports that cater to very specific use cases. For instance, logging out the response body when a request (or it's tests) fail, and so on.

Building custom reporters

A custom reporter is a Node module with a name of the form `newman-reporter-<name>`. To create a custom reporter:

1. Navigate to a directory of your choice, and create a blank npm package with `npm init`.
2. Add an `index.js` file, that exports a function of the following form:

```
function (emitter, reporterOptions, collectionRunOptions) {
  // emitter is is an event emitter that triggers the following events: https://git
  // reporterOptions is an object of the reporter specific options. See usage ex
  // collectionRunOptions is an object of all the collection run options: https://g
};
```

3. Publish your reporter using npm publish, or use your reporter locally see usage instructions.

Scoped reporter package names like @myorg/newman-reporter-<name> are also supported. Working reporter examples can be found in working reporter examples.

Using custom reporters

In order to use the custom reporter, it will have to be installed first. For instance, to use the Newman teamcity reporter:

Install the reporter package.

```
npm install newman-reporter-teamcity
```

Note that the name of the package is of the form newman-reporter-<name>, where <name> is the actual name of the reporter. The installation should be global if Newman is installed globally, local otherwise. Run npm install ... with the -g flag for a global installation.

To use local (non-published) reporters, run the command npm install <path/to/local-reporter-directory> instead.

Use the installed reporter, either via the CLI, or programmatically. Here, the newman-reporter prefix is not required while specifying the reporter name in the options.

Scoped reporter packages must be specified with the scope prefix. For instance, if your package name is @myorg/newman-reporter-name, you must specify the reporter with @myorg/name.

CLI:

```
newman run /path/to/collection.json -r myreporter --reporter-myreporter-<option-n
```

Programmatically:

```
var newman = require('newman');
```

```
newman.run({
   collection: '/path/to/collection.json',
   reporters: 'myreporter',
   reporter: {
     myreporter: {
       'option-name': 'option-value' // this is optional
     }
   }
}, function (err, summary) {
  if (err) { throw err; }
  console.info('collection run complete!');
});
```

In both cases above, the reporter options are optional.

For the complete list of details, see the Newman README.

Debugging a collection run Integration with Jenkins
Integration with Jenkins