

# 目錄

Introduction	1.1
介绍	1.2
安装	1.3
安装和更新	1.3.1
发送请求	1.3.2
创建集合	1.3.3
导航	1.3.4
账号	1.3.5
同步	1.3.6
设置	1.3.7
发送 API 请求	1.4
请求	1.4.1
响应	1.4.2
历史	1.4.3
请求异常	1.4.4
调试及日志	1.4.5
授权	1.4.6
Cookies	1.4.7
证书	1.4.8
捕获HTTP请求	1.4.9
拦截器	1.4.10
代理	1.4.11
生成代码片段	1.4.12
SOAP 请求	1.4.13
集合	1.5
创建集合	1.5.1
分享集合	1.5.2
管理集合	1.5.3
描述使用 Markdown	1.5.4
示例	1.5.5

数据格式	1.5.6
脚本	1.6
脚本介绍	1.6.1
请求前脚本	1.6.2
测试脚本	1.6.3
测试示例	1.6.4
分支和循环	1.6.5
Postman Sandbox	1.6.6
Postman Sandbox API	1.6.7
环境和全局	1.7
变量	1.7.1
管理环境	1.7.2
管理全局	1.7.3
Collection runs	1.8
开始	1.8.1
使用环境	1.8.2
Working with data files	1.8.3
Running multiple iterations	1.8.4
创建工作流	1.8.5
分享	1.8.6
调试	1.8.7
Command line integration with Newman	1.8.8
Integration with Jenkins	1.8.9
Integration with Travis CI	1.8.10
Newman with Docker	1.8.11
Team Library	1.9
Setting up a Team Library	1.9.1
Sharing	1.9.2
Activity feed and restoring collections	1.9.3
Searching	1.9.4
Conflicts	1.9.5
API documentation	1.10
Intro to API documentation	1.10.1
Viewing documentation	1.10.2

---

Environments and environment templates	1.10.3
How to document using Markdown	1.10.4
Publishing public docs	1.10.5
Adding and verifying custom domains	1.10.6
Adding team name and logo	1.10.7
Monitors	1.11
Intro to monitors	1.11.1
Setting up a monitor	1.11.2
Viewing monitor results	1.11.3
Monitoring APIs and websites	1.11.4
Set up integrations to receive alerts	1.11.5
Pricing for monitors	1.11.6
Troubleshooting monitors	1.11.7
FAQs for monitors	1.11.8
Mock servers	1.12
Setting up a mock server	1.12.1
Mocking with examples	1.12.2
Mocking with the Postman Pro API	1.12.3
Matching algorithm	1.12.4
Postman API	1.13
Intro to the Postman API	1.13.1
Continuous Integration	1.13.2
Notifications	1.14

---

# Postman 中文文档

Postman 功能其实很强悍的，但是由于使用文档较少，很少有人去更深层次的使用，只是用来简单的访问请求。

本人利用闲暇时间来翻译官方文档，旨在让人们更容易理解使用，Postman 不再大材小用。

[Postman 官方文档](#)



*Supercharge your API workflow*

*Modern software is built on APIs. Postman helps you develop APIs faster.*

## Postman 应用支持

如果你有任何建议和问题，请在 [Github](#) 上告诉我们 ([Github issue tracker](#))。在提问题前，我建议你先去 [Github](#) 看看是否已经有人提了这个问题，以及是否有解决方案。

如果你要提交问题，请注明重现 Bug 的步骤、Postman 版本号、Chrome 浏览器版本号和操作系统版本。如果有 collections、数据转储、错误信息、截图等其他信息对我们找到问题更有帮助。[We have compiled a quick set of guidelines for reporting issues.](#)

账户特定查询：如果您有任何账单或账户特定查询，请通过 [help@getpostman.com](mailto:help@getpostman.com) 与我们联系。

在 Twitter 上关注我们 [@postmanclient](#)，关注 Postman 的最新消息、功能和发布信息。

## 社区



如果你有兴趣与团队交谈，我们就在 Slack。随意下来，打个招呼。与世界各地好友共同讨论 Postman 即将发布的功能和测试版本。

从[这里](#)获取 Postman Slack 社区的邀请。

已经是会员？从[这里](#)登录。

## 产品路线图

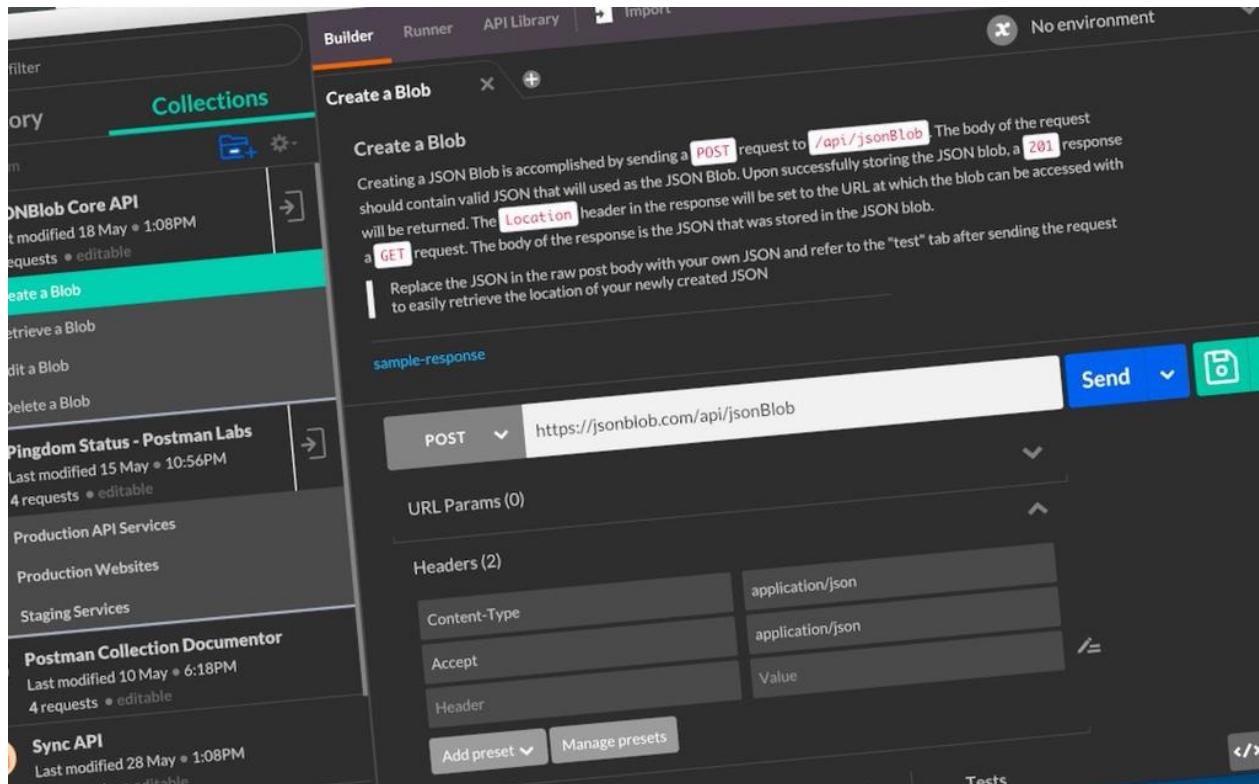
如果您对即将发布的版本计划的功能和增强功能感到好奇，可以访问

<https://trello.com/b/4N7PnHAz/postman-roadmap-for-developers> 查看 Postman 的开发人员路线图。

想早点体验这些功能？我们最新的 Canary 版本提供了一些增强功能。下载：[OSX \(x64\)](#) / [Windows \(x86 或 x64\)](#) / [Linux \(x86 或 x64\)](#).

## 文档和教程

如果你想了解更多有关应用功能、安装和使用的信息，请访问我们网站的[文档部分](#)。你可以访问我们的[博客](#)了解有趣的教程、开发案例和平台更新。



## 问题报告指南

We have put together a short set of guidelines you can follow while adding an issue to our [GitHub issue tracker](#). Following this should help speedy resolution of issues.

1. This issue tracker is only for Postman App related issues, along with other services accessible from the app. If you have Newman specific issues, a better place to report them would be the Newman issue tracker at  
<https://github.com/postmanlabs/newman/issues>
2. If you are facing [Postman Cloud](#) related issue (such as sync, cloud-api, documenter, etc) and you want to include personal information such as your username or collection names, then mail us at [help@getpostman.com](mailto:help@getpostman.com).

3. Answer to questions along the lines of "How do I... in Postman" should be in our online documentation at <http://getpostman.com/docs>. If you are unable to find a how-to guide in our online documentation, feel free to ask your question on our [Slack Community](#).
4. Before reporting an issue use the search feature on the issues page to check if there are issues similar to yours. A lot of issues are duplicates, and it is hard to keep track of them or respond when the issues are solved. If you find your issue already reported, feel free to add "+1" reaction and we will keep a note of it.
5. When reporting a new issue in the issue tracker, check whether it helps to answer the following questions:
  - Which Postman App, Chrome and Operating System version you are using. You can check this out in *Settings -> About* section.
  - Is the bug reproducible every time, or do you see it occasionally?
  - Did you first encounter it recently, or has it always been there?
  - If it is a UI issue, a screenshot or GIF will help tremendously. (Tip: For quick gifs, check out <http://www.cockos.com/licecap/>)
  - Do you have the [Postman Interceptor](#) switched on? (applicable for the Chrome app)
  - If the request/response flow is not working, be sure to check and include details from the DevTools window (More on this at <https://www.getpostman.com/docs/errors>).

## 关于 Postman

Postman 是一个功能强大的 API 测试套件，已经成为许多开发人员必备的工具。我们制作漂亮的产品来构建令人惊叹的 API 和提升开发者的生产力。Postman 现在已有一百多万的开发者在使用，这个数字还在持续增长。随着更多产品的投入使用，我们的目标是为开发人员提供最全面的API开发和测试解决方案。

Postman 官网: <https://www.getpostman.com/>

---

如果您遇到与 Postman 旧版本有关的问题或程序贡献，请访问 [postmanlabs/postman-chrome-extension-legacy](https://github.com/postmanlabs/postman-chrome-extension-legacy)。

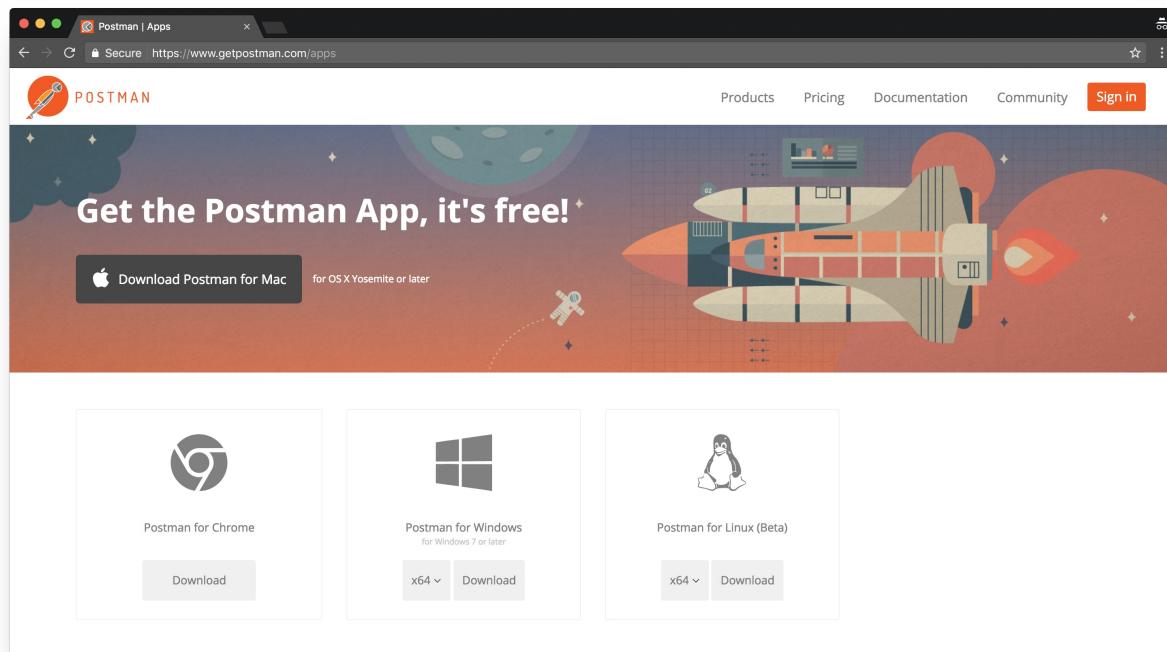


## 安装 Postman

### Postman 客户端

Postman 客户端支持 Mac、Windows 和 Linux 系统。

前往[应用页](#)点击 **Download** 下载适合自己的版本。



#### macOS 安装

Once you've downloaded app, you can drag the file to the “Applications” folder. Double click on Postman to open the application.

#### Windows 安装

- 下载安装文件
- 运行安装程序

#### Linux 安装

- Installation on Linux can vary between distributions. Check out this guide for installing the [Postman app on Ubuntu 16.04](#).

### Postman Chrome 应用

虽然我们支持谷歌浏览器插件，但是推荐使用原生客户端。了解更多关于[为什么 Postman 将弃用谷歌应用](#)。

Postman 谷歌应用只能在谷歌浏览器上运行。在使用前你必需先[安装谷歌浏览器](#)。

如果你已经安装了 谷歌浏览器，前往谷歌应用商店搜索 [Postman](#)，然后点击“Add to Chrome”下载

下载所需时间取决于你的网络环境。下载完成后你可以[运行 Postman](#)。

## 原生应用和谷歌应用的区别

Postman's native apps are built on [Electron](#), and overcome a number of restrictions of the Chrome platform.

A few features exclusive to the native apps are listed here:

### Cookies

The native apps let you work with [cookies](#) directly. Unlike the Chrome app, no separate extension ([Interceptor](#)) is needed.

### Built-in proxy

The native apps come with a built-in proxy that you can use to [capture network traffic](#).

### Menu bar

The native apps are not restricted by the Chrome standards for the menu bar. With the native apps, you can create collections, switch to history requests, and more.

### Restricted headers

The latest versions of the native apps let you send headers like Origin and User-Agent. These are [restricted](#) in the Chrome app.

### Don't follow redirects option

This option exists in the native apps to prevent requests that return a 300-series response from being automatically redirected. Previously, users needed to use the Interceptor extension to do this in the Chrome app.

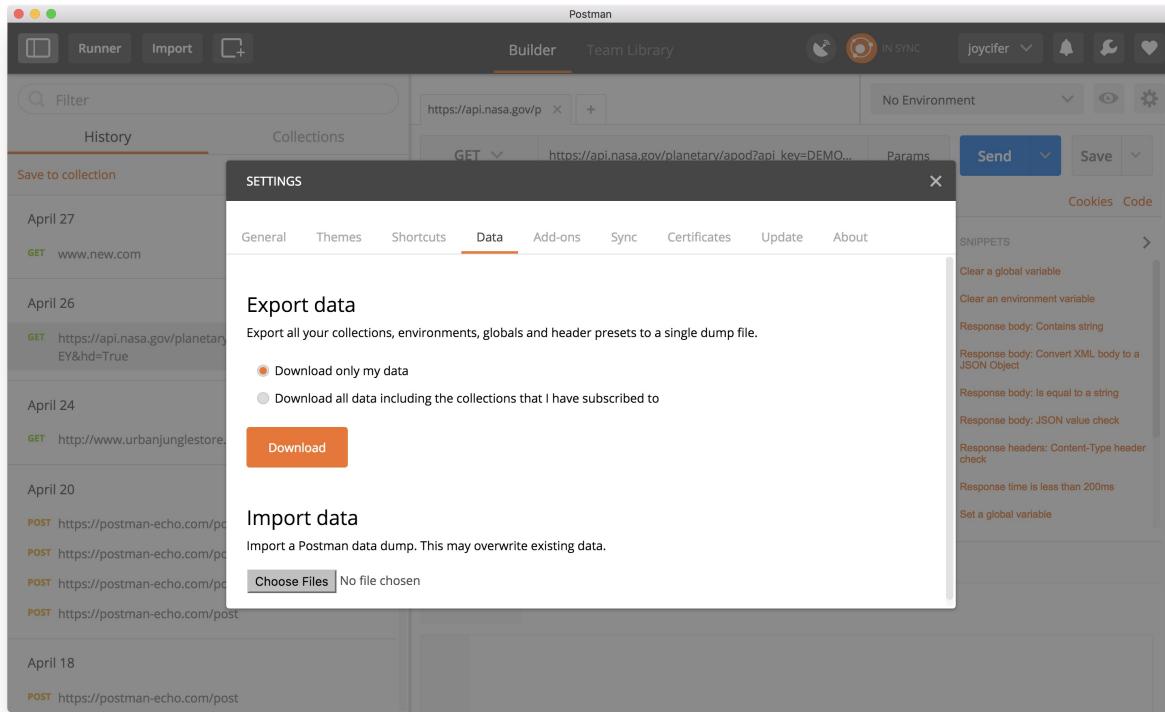
### Postman console

The latest version of the native app also has a built-in [console](#), which allows you to view the network request details for API calls.

## Migrating to the native app

It's simple. Sign in to your Postman account after you [download](#) and start the new native app, and all your history and collections will be automatically synced.

Alternatively, if you don't want to sign in to your Postman account, you can bulk export your Postman data from the Chrome app, and then bulk import into the new native app.



## Bulk export

From the Postman settings, select the **Data** tab and click the **Download** button to export all your collections, environments, globals and header presets to a single dump file.

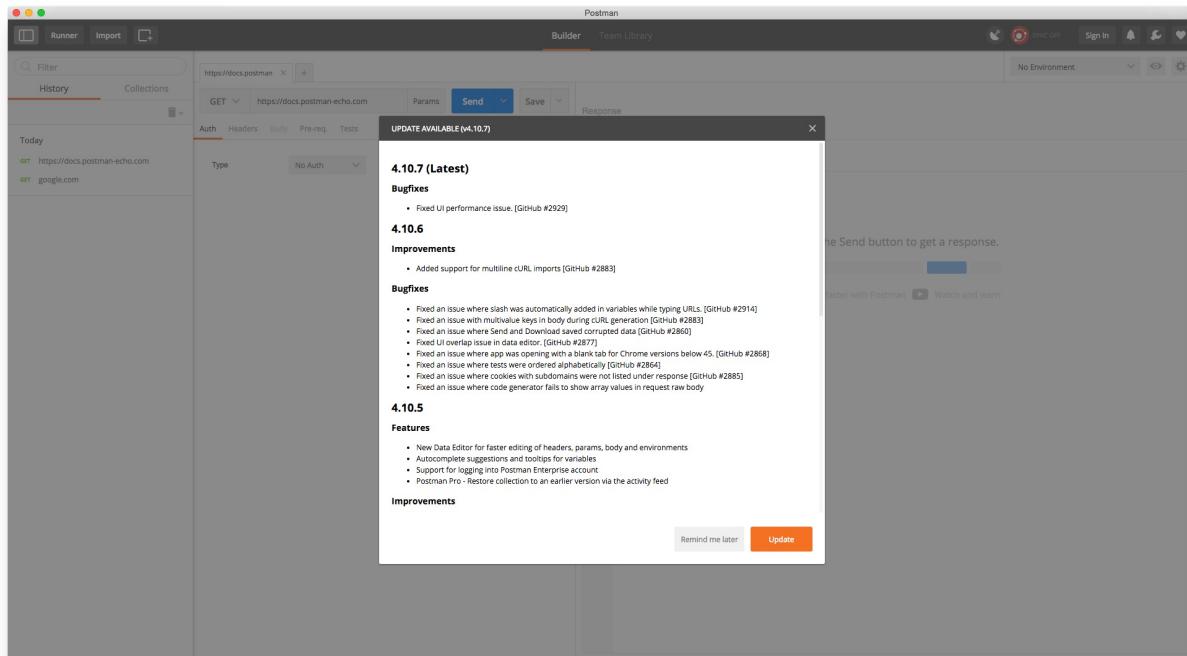
## Bulk import

From the same area in Postman settings, you can initiate a bulk import from a Postman data dump file. This will overwrite your existing data so be a little careful.

## Updating Postman

### Native app (Mac, Windows and Linux)

Postman's native apps will check for updates whenever the app reloads, or is launched. The app will display the changelog prompting you to update the app.



## Mac and Windows

Click **Update** to download the latest update. You will be notified when the download is complete prompting you to restart the Postman app to apply the updates. If you're not ready to update yet, click **Remind me later** to prompt you again after the next app reload or launch.

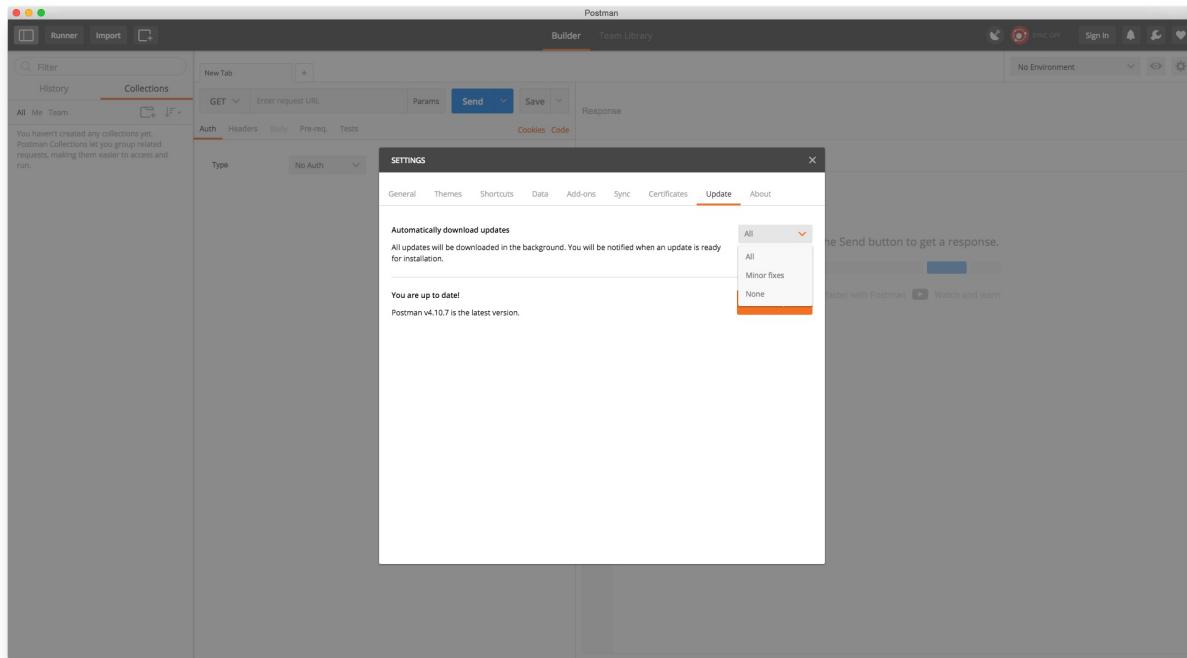
You can also configure your preferences to automatically download updates under the **Update** tab within the **SETTINGS** modal.

- All
- Minor fixes
- None

**All** - Downloads all updates automatically and will show a small notification at the top prompting you to restart the app to apply the updates.

**Minor fixes** - You will be notified of all major updates, and other minor fixes will automatically download prompting you to restart the app to apply the updates.

**None** - This will show up the update version every time it finds a update for your current version.



## Troubleshooting updates in macOS Sierra

We have received user feedback that the Mac update does not complete successfully, even after downloading the update for macOS Sierra.

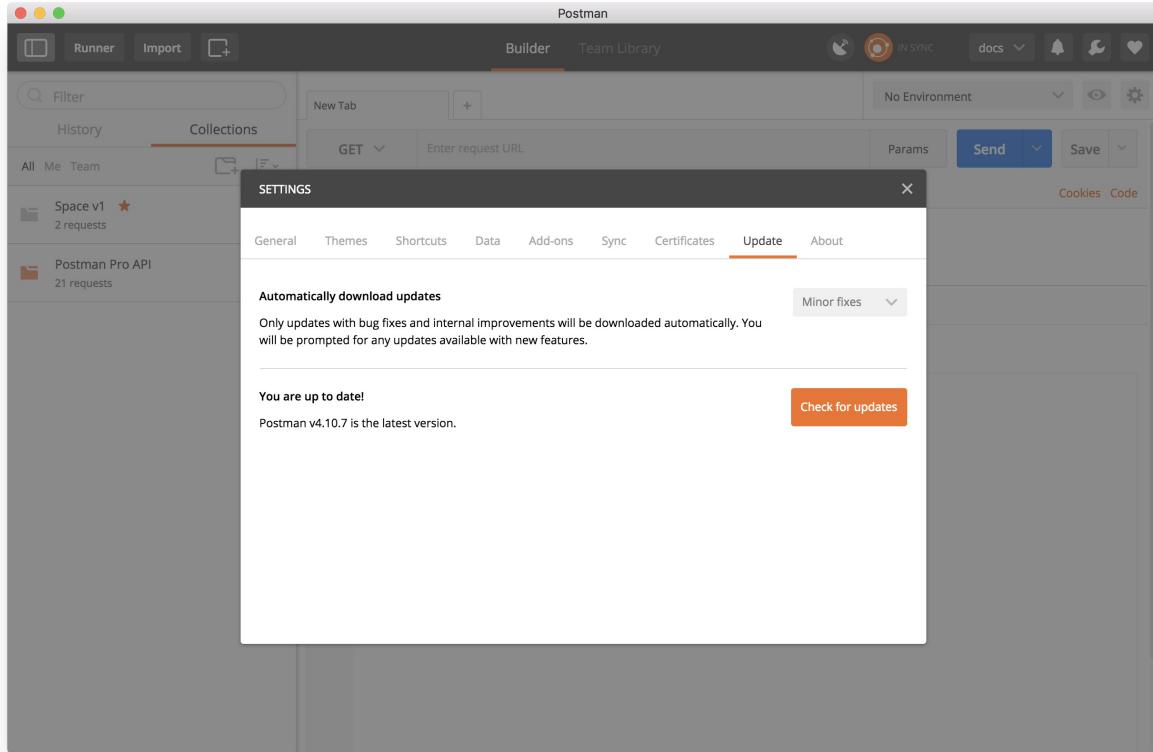
This can be solved by moving the app out of the Downloads directory. The Postman updater is unable to switch the downloaded version on the read-only memory, initially assigned for the downloaded apps by macOS Sierra.

If you continue experiencing difficulty with the update, fetch the logs from `~/Library/Caches/com.postmanlabs.mac.ShipIt` in your system and let us know.

## Linux

Postman's native app on Linux will notify you whenever an update is available. If an update is available, you need to download the [latest version](#) of the application, and replace the current application directory with the new version. Postman stores all user data outside of the application directory, so you can safely replace the current application directory with the new version.

Since Postman's native apps check for updates only on app reload or launch, at any time, you can force a check for updates under the **Update** tab in the **SETTINGS** modal of the app.



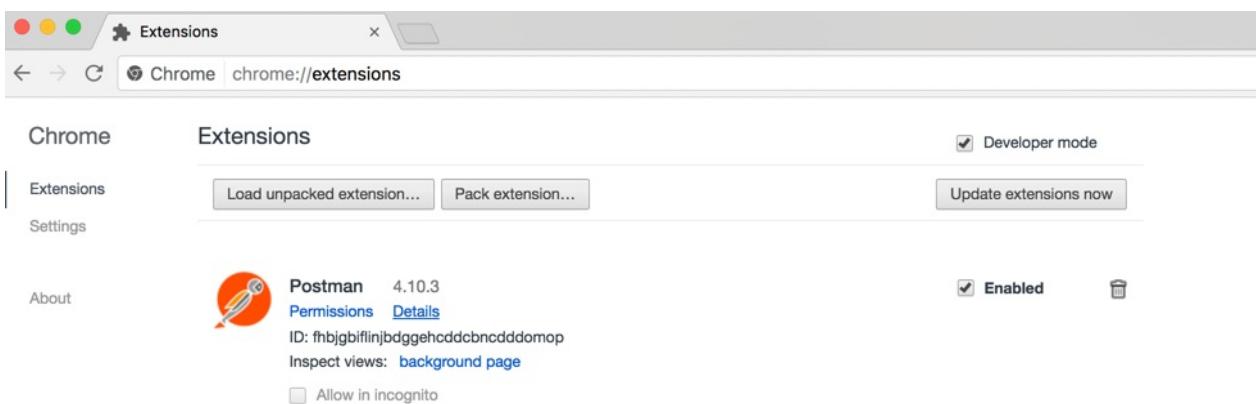
## Chrome

Postman's Chrome app is usually updated automatically. However, Postman doesn't control the Chrome app update flow, and Chrome sometimes doesn't update the app for long periods of time.

The latest version is visible on [Postman's Web Store listing](#).

To manually force an update, here's what you need to do in Chrome:

1. In the address bar, type chrome://extensions.
2. At the top of the page, check and enable **Developer Mode**.
3. Click the **Update extensions now** button beneath **Developer Mode**.





# 发送第一个请求

让我们发送我们的第一个API请求！

- 将 [postman-echo.com/get](http://postman-echo.com/get) 填入 URL 栏.
- 点击 **Send** 按钮发送你的请求, 你将在底部看到服务返回的一些 JSON 数据. 注意 Postman 已经将 [postman-echo.com/get](http://postman-echo.com/get) 添加到侧边栏的 **History** 选项卡下.

The screenshot shows the Postman application window. The top navigation bar includes 'Runner', 'Import', 'Builder' (which is selected), 'Team Library', 'OFFLINE', and user information 'joyce+test'. The left sidebar has 'History' (selected) and 'Collections'. The main workspace shows a request for 'postman-echo.com/get' with a 'GET' method. The 'Authorization' tab is selected, showing 'No Auth'. The 'Body' tab is selected under the response pane, which displays a JSON response with various headers and a URL. The status bar at the bottom indicates 'Status: 200 OK'.

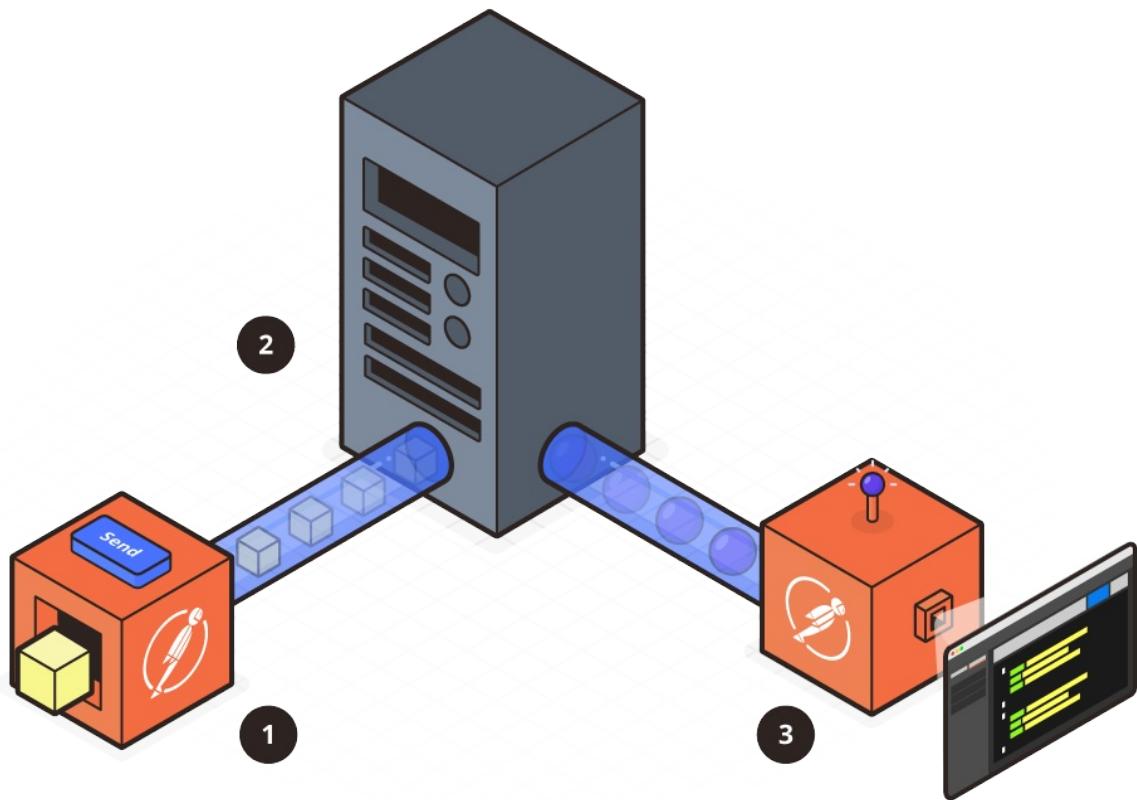
```

1  {
2   "args": {},
3   "headers": {
4     "host": "postman-echo.com",
5     "cache-control": "no-cache",
6     "postman-token": "fdad91085-e2e5-4af4-a813-98c629b2cb01",
7     "user-agent": "PostmanRuntime/3.0.11-hotfix.2",
8     "accept": "*/*",
9     "cookie": "sails.sid=s%3AG1B72G7GttxSBvgmr9dQGAX0kx00JCYJ
10 .SozhMKdVXx2Fsmz08hoJ2c9Qo%2FZ0Qab1J0xq4Mi62yTQI",
11   },
12   "url": "http://postman-echo.com/get"
13 }

```

## 怎么用

下面我们用一个简单的例子来说明这个过程：



1. 在 Postman 中输入你的请求信息 (URL: postman-echo.com/get), 然后点击 **Send** 按钮.
2. 该请求由 API 服务器 (postman-echo.com) 接收，并且它返回一个响应.
3. Postman 收到请求，然后在界面显示响应.

## Postman Echo

[postman-echo.com](http://postman-echo.com) 是 Postman 为您提供各种类型的请求的示例API. 它将返回包含你的请求信息的响应.

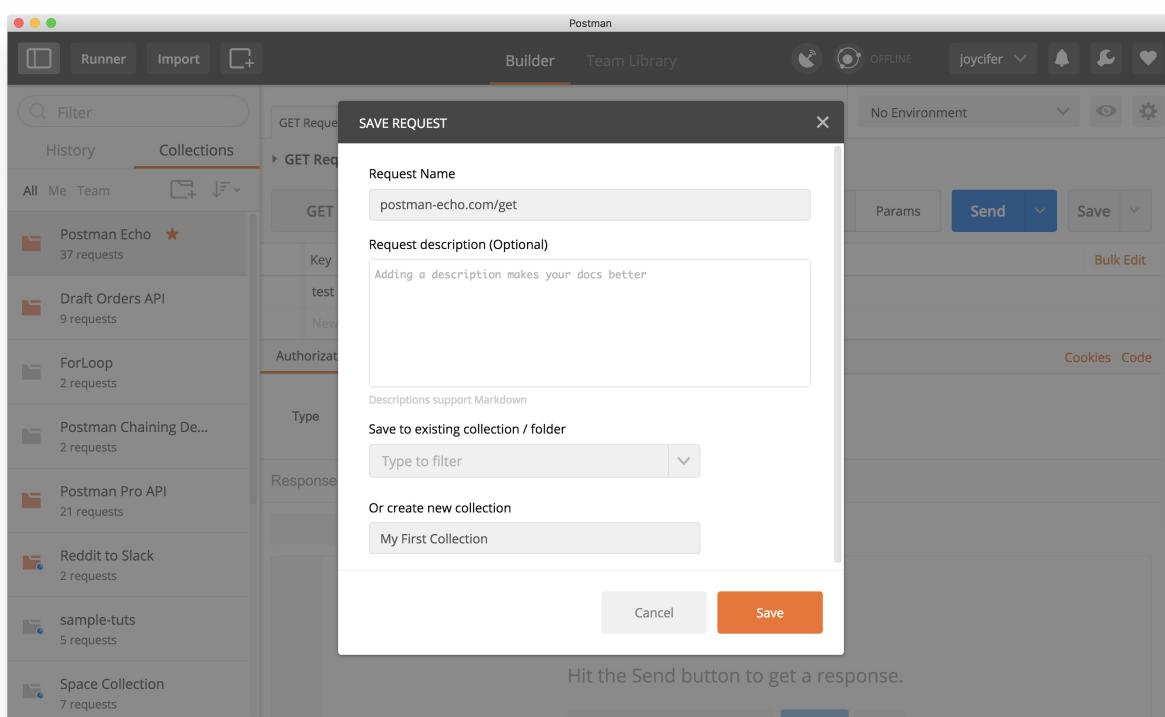
导入此示例集合并参考 [此示例 API 的文档](#).

# 创建第一个 Collection

你在 Postman 发送的每个请求都显示在侧边栏的 **History** 选项卡下。在小规模上，通过历史部分重复使用请求很方便。但是，随着 Postman 使用规模的扩大，在您的历史记录中查找特定请求可能会耗费大量时间。这就是 Postman 收集的地方。Collections 是保存请求的组合，这是 Postman 众多功能中的一个基础功能。

我们从 [发送第一个请求](#) 中断的地方继续，然后创建一个新的集合。

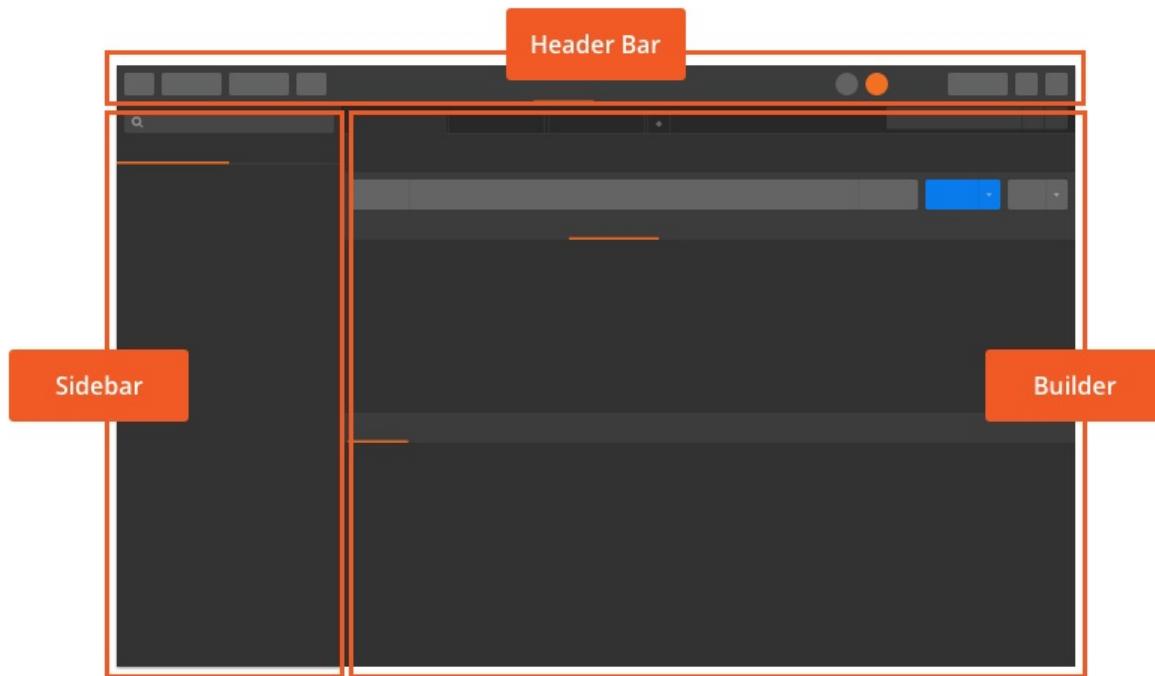
- 通过刚刚在请求生成器中创建的请求，单击 **Save** 按钮打开 **SAVE REQUEST** 模式。
- 作为可选步骤，输入一个新的请求名称。否则，默认名称将是请求URL。
- 作为可选步骤，请以纯文本或使用 [Markdown](#) 输入请求说明。
- 现在，将此请求保存到现有集合中，或者通过输入集合名称来创建新集合，然后单击 **Save**。



恭喜！您可以在左侧边栏的 **Collections** 选项卡下看到所有集合。

## 浏览 Postman

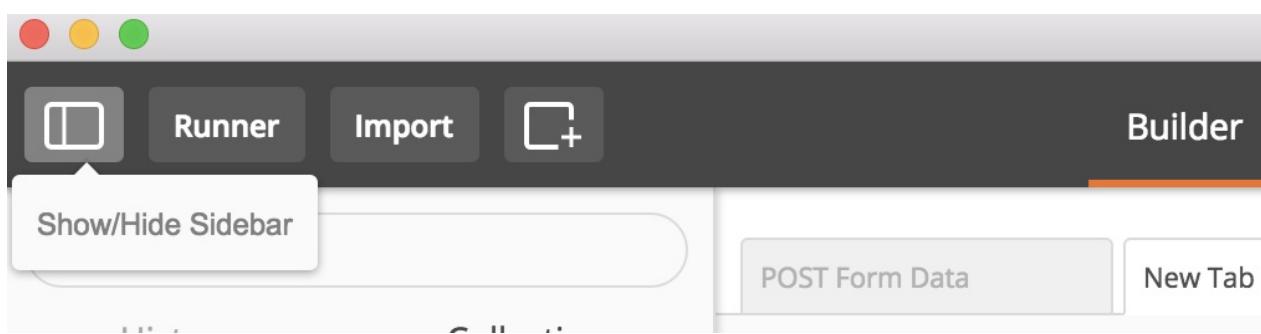
Postman 提供了一个多窗口和多标签界面让你使用 API??。这种界面设计为您的 API 提供了尽可能多的空间。



### 侧边栏

侧边栏方便你查找和管理请求和集合。侧边栏有两个标签：[History](#) 和 [Collections](#).

您可以拖动右边缘来调整侧边栏的宽度。您还可以最小化小屏幕的侧边栏，并在状态栏中显示或隐藏侧边栏。



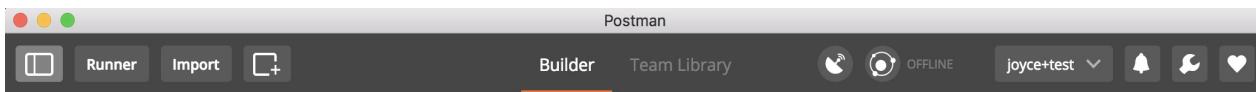
### History 选项卡

Postman 将您发送的每个请求保存在侧边栏的 **History** 选项卡中。学习了解 [history](#).

## Collections 选项卡

在侧边栏的 **Collections** 选项卡中创建和管理集合. 学习了解 [collections](#) 以及它们如何加速 API 的速度.

### 标题栏



顶部的标题栏包含以下选项:

- **New** 按钮: 创建请求，集合，环境，文档，模拟服务器和监视器。
- **Import** 按钮 - 使用文件，链接或者文本将 Postman 集合，环境，WADL，Swagger，RAML，or cURL 导入 Postman.
- **Runner** 按钮 - 查看 [collection runner](#).
- 新窗口图标 - 打开一个新的“Tab”，“Postman Window”或“Runner Window”.
- **Builder** 和 **Team Library** 选项卡 - 在请求生成器和团队库之间切换.
- 拦截器/代理 图标 - 管理代理或拦截器设置。
- 同步状态图标 - 更新 Postman 账户的状态.
- 公共 API 库 - 显示公共API网络.
- 设置图标 - 管理 Postman 应用程序设置并查找其他支持资源.
- 通知图标 - 接收通知或广播.
- 心形图标 - 喜欢 Postman 吗？点击这个按钮分享爱！
- 用户下单菜单 - 显示当前用户并提供以下选项：“配置文件”，“帐户设置”，“通知首选项”，“活动会话”和“添加新帐户”.

### 生成器

Postman 生成器为您提供了一个选项卡式的布局来发送和管理构建器中的API请求. 上半部分是请求生成器，下半部分是响应查看器.

- **Cookies** - 你可以使用 **Save** 按钮下的 **Cookies** 链接来访问 **MANAGE COOKIES** 模式. 该功能可让您管理与请求相关联的Cookie. 详细了解 [使用 cookies](#).
- **Code** - 您可以使用 **Save** 按钮下最右侧的 **Code** 链接来访问 **GENERATE CODE SNIPPETS** 模式. 通过此功能，您可以使用20多种语言生成与请求关联的代码片段。

### 控制台

Postman 有两个控制台可以看到幕后发生了什么. 了解更多 [使用控制台日志排除问题](#).

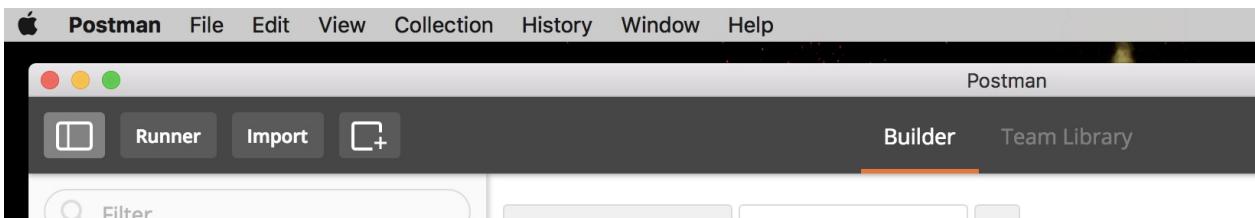
- Postman 控制台 - 包含 HTTP 请求和响应的运行日志. 你可以使用脚本记录消息(例如 console.log). 此功能仅适用于 Postman 的本机应用程序.

- DevTools 控制台 - 在开发过程中提供诊断信息。要了解如何访问 DevTools 控制台日志，请参阅 [调试和日志](#)。

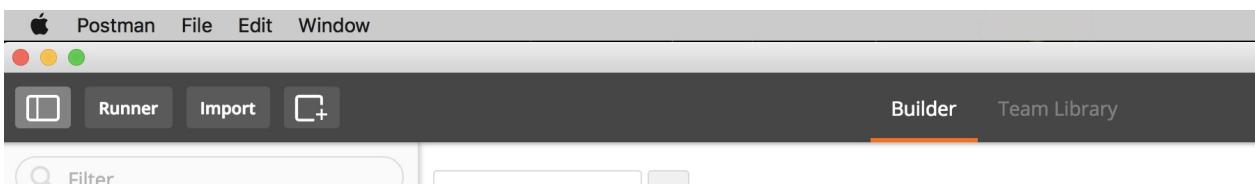
## 菜单栏

Postman 应用程序菜单栏提供对其他功能的访问。请注意，Postman 原生应用程序的菜单栏和 Postman 的 Chrome 应用程序之间有一些区别。

**Postman 原生应用程序：**显示更多菜单选项以访问 Postman 特定的功能。



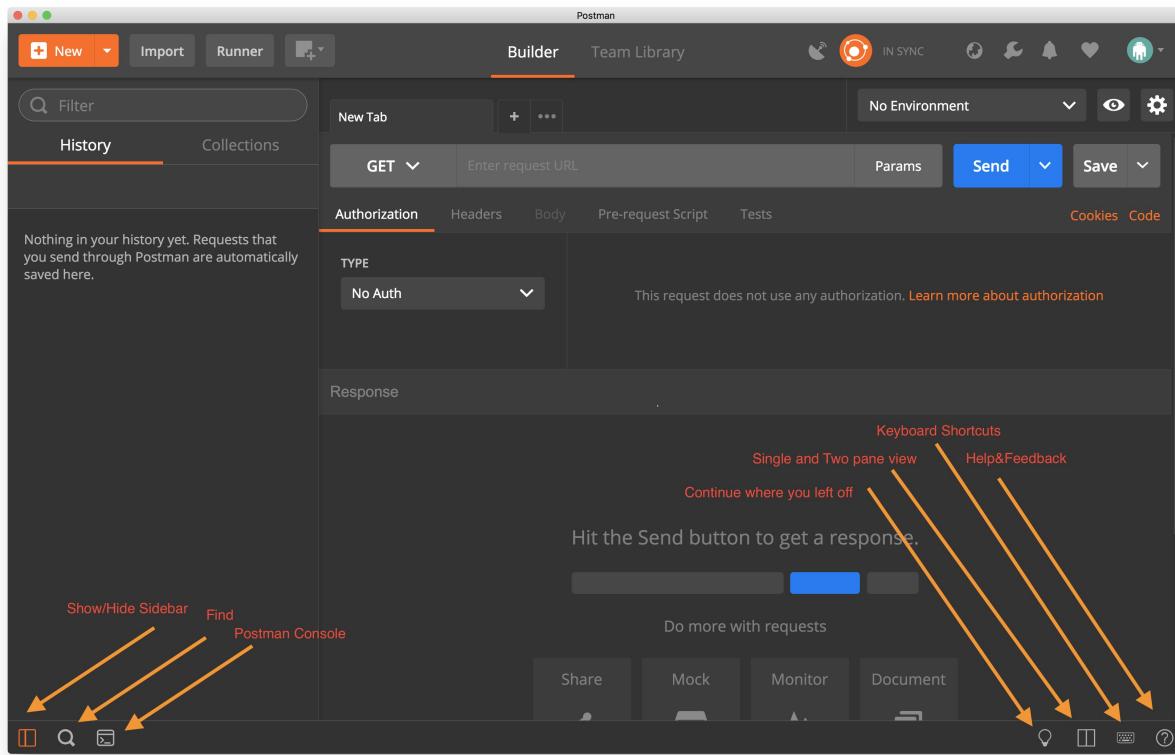
**Postman Chrome 应用程序：**由于 Chrome 标准限制，显示较少的菜单选项。



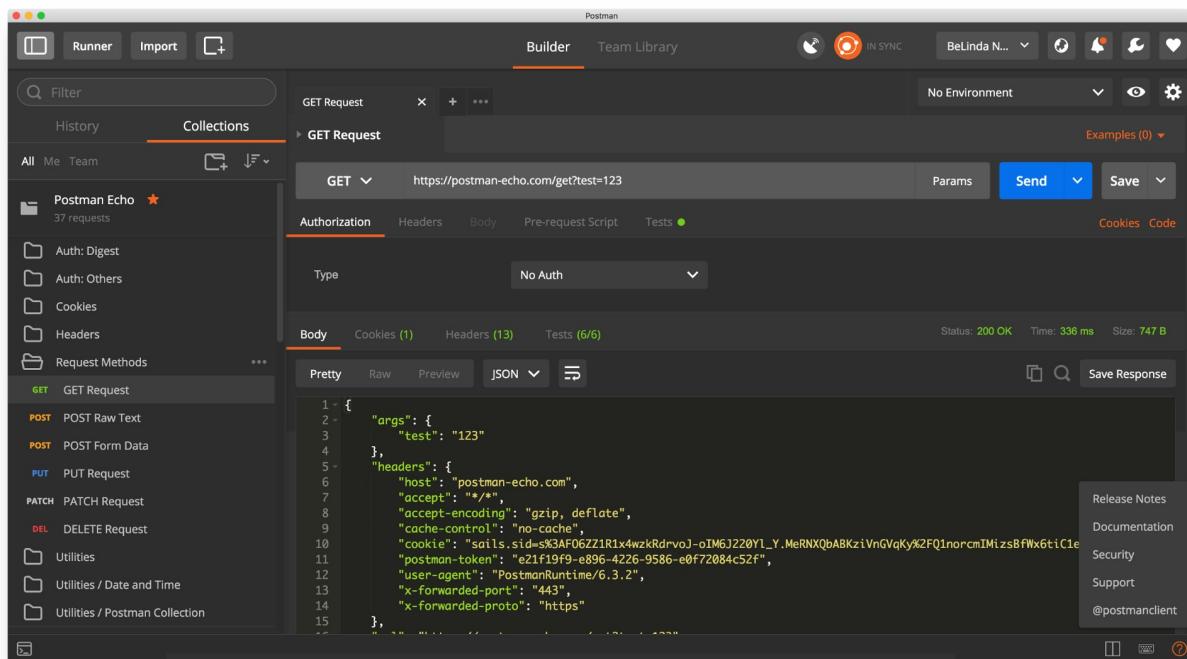
## 状态栏

Postman 界面底部的状态栏为您提供了一种便捷的方式：

- 显示或隐藏侧边栏。
- 搜索集合、环境和全局。
- 打开 Postman [控制台](#)。
- 选择一个或两个窗格布局。
- 打开键盘 [快捷键](#)。
- 获得帮助或提供反馈。



当你单击 **Help & Feedback** 图标时, 你可以看到一个菜单, 提供最新发布说明, 文档, 安全性, 支持, and 社交媒体的访问.



## 选项卡和窗口

Postman 让你使用多选项卡和对窗口配置来处理请求. 甚至可以同时进行多个请求.

要在 Postman 中打开新选项卡，请按下生构建器中的 + 图标或使用 **CMD/CTRL + T** 快捷键。从菜单栏中，您也可以从 **File** 菜单中选择“New Tab”来创建一个新选项卡。

当您右键单击一个选项卡名称时，该菜单允许您复制或关闭选项卡。如果任何选项卡在尝试关闭选项卡时有未保存的更改，Postman 会提示您保存更改。

## Busy tabs

The idea behind busy tabs is to make sure people don't lose the request they've been working on, even if it isn't in a collection. When you're on a busy tab, opening a new request from the sidebar will open the request in a new tab. It does not replace or interfere with the request in the previous busy tab.

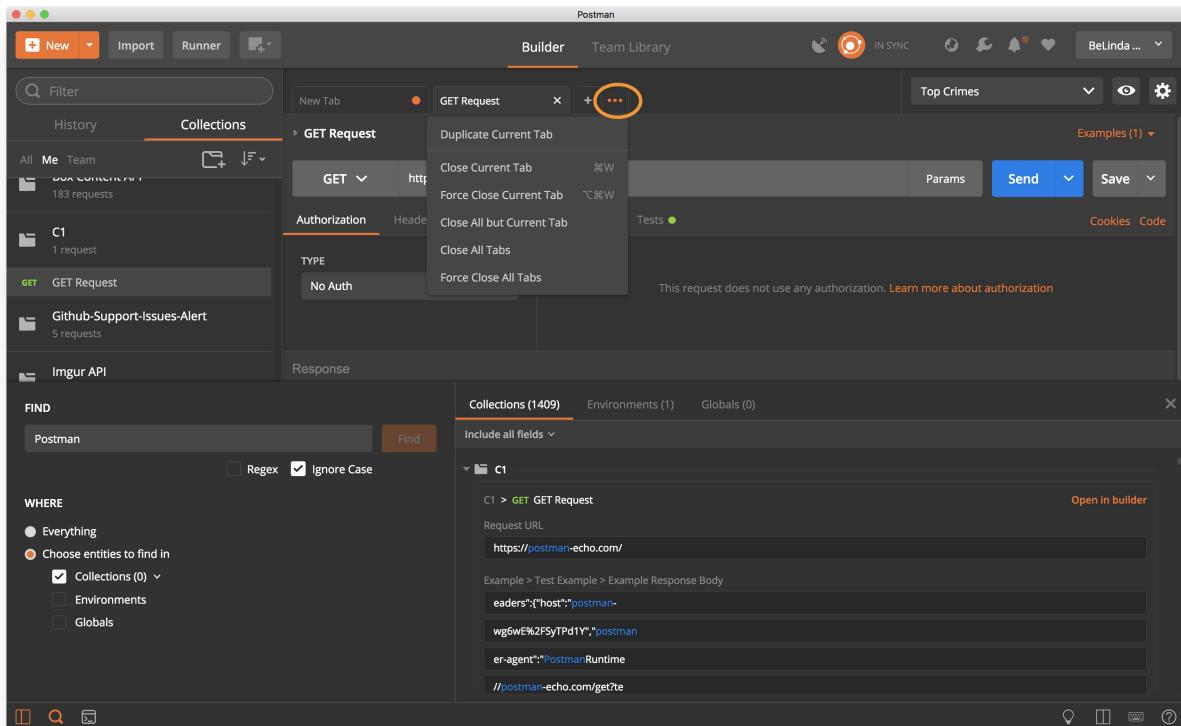
What makes a tab busy? Working on a tab moves the tab into a busy state. For example, receiving a response or making changes that are not yet saved (reflected by an orange dot on the tab) will all result in making a tab busy.

### 选项卡和侧边栏行为

默认情况下，Postman 假定您想在一个选项卡中处理一个请求集合。当您从侧边栏打开请求时，Postman 在现有选项卡未保存更改时打开新选项卡，如果不是，请求将在当前选项卡打开。在侧边栏的 **Collections** 选项卡中，右键单击一个请求并选择“Open in New Tab”，这样你可以始终在新选项卡中打开一个请求。

### 选项卡菜单

邮差提供了几个选项卡操作，以帮助您管理您的工作。

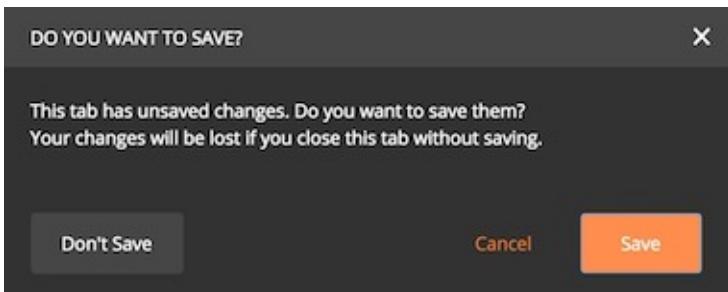


要访问选项卡菜单，请单击标签右侧的三个点。出现下拉菜单，其中包含管理选项卡的选项。

- 复制当前选项卡
- 关闭当前选项卡
- 强制关闭当前选项卡
- 关闭除当前选项卡外其他所有选项卡
- 关闭所有选项卡
- 强制关闭所有选项卡

当您“强制关闭当前选项卡”或“强制关闭所有选项卡”时，选项卡会立即关闭，而不会提示您将工作保存在选项卡中。

当您“关闭当前选项卡”，“关闭所有当前选项卡”或“关闭所有选项卡”时，将出现一个对话框，其中包含保存所做工作的选项。



移动请求

在构建器中，你可以拖拽选项卡重新排序。在新的选项卡或者在新的 Postman 窗口中打开一个请求。你可以使用标题栏上的 **New Window** 图标或快捷键打开多个窗口。

## 键盘快捷键

Keyboard usability is high on the priority list for any dev tool. For most developers, it's a more efficient input method, requiring minimum movement and effort compared to a mouse or other pointing device. It also saves time, and for repetitive or well-frequented tasks, this can bring about a huge improvement in speed over the long run.

When we approached assigning keyboard shortcuts in Postman, we broadly categorized them into three buckets based on their function: navigation, manipulation and global. Navigational shortcuts help you move around the interface, manipulation shortcuts allow you to manipulate the current selection and global shortcuts can be accessed from anywhere.

### Navigational Shortcuts

We want to make it easier for a user to navigate quickly between elements. Let's take the simple example of opening and sending a series of saved requests. Without a shortcut, a user selecting a request in the sidebar would use their pointer every time to open it in a tab. With the **CMD/CTRL + ALT + 1** shortcut, you can now focus the sidebar from wherever you are in interface. You can then navigate to the desired request using the arrow keys. Combine this with **CMD/CTRL + Enter** to send a number of requests in the matter of a few seconds.

### Manipulation Shortcuts

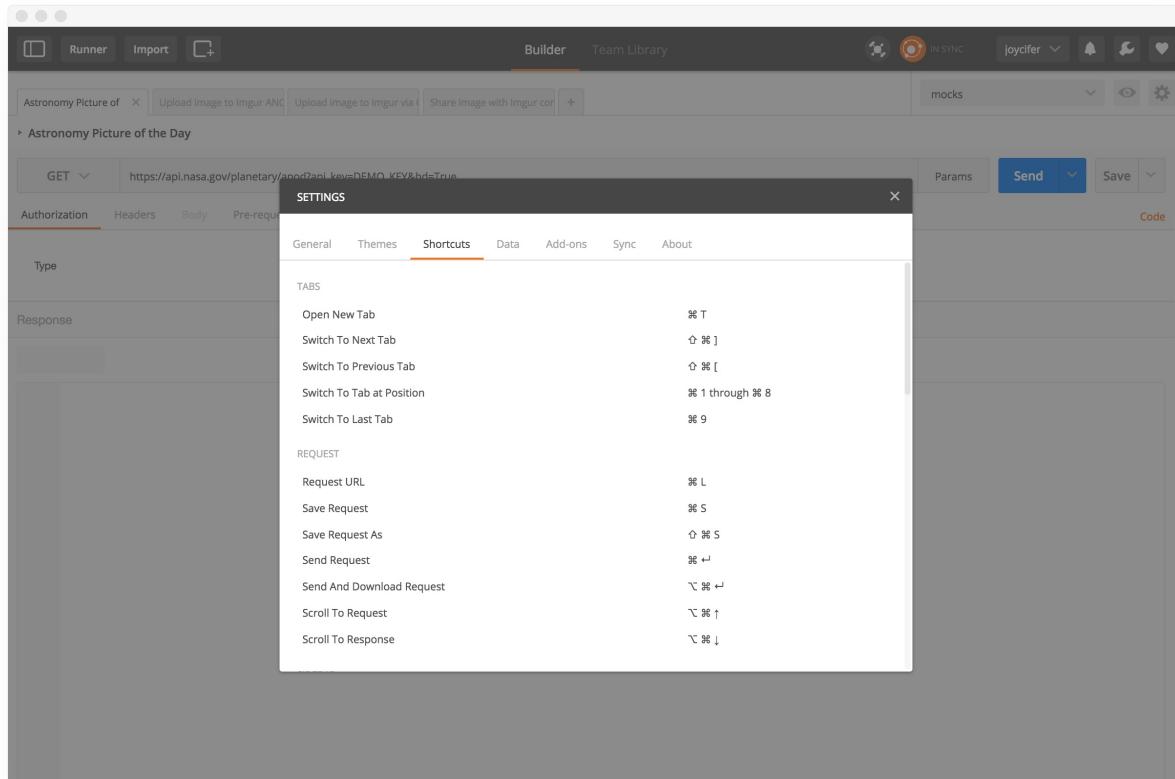
Manipulation shortcuts allow you to quickly work on your current selection and perform actions such as edit, delete and duplicate. Combined with navigational shortcuts, this makes creating and organizing collections in your sidebar a breeze. These shortcuts can also be used in other list views such as our new data editor.

### Global Shortcuts

Global shortcuts are used for important actions so that they can be accessed irrespective of the element in focus. Actions on your selected tab, like Save (**CMD/CTRL + S**), Save As (**CMD/CTRL + Shift + S**), and Send Request (**CMD/CTRL + Enter**) all follow this pattern. Global shortcuts can also be used to perform UI actions like toggling the sidebar (**CMD/CTRL +** ), Jump to URL (**CMD/CTRL + L**), and Open Console (**CMD/CTRL + ALT + C**).

### View keyboard shortcuts for your OS

Different operating systems will have different shortcuts. You can always view a complete list of your operating system's shortcuts under the **Shortcuts** tab in the **SETTINGS** modal.



## Reference for all shortcuts

TABSmacOSWindows / Linux shortcutsOpen New Tab? TCtrl + TClose Tab? WCtrl +  
WSwitch To Next Tab? ? ]Ctrl + Shift + ]Switch To Previous Tab? ? [Ctrl + Shift + [Switch To  
Tab at Position? 1 through ? 8Ctrl + 1 through Ctrl + 8Switch To Last Tab? 9Ctrl + 9Open  
Request from Sidebar in New Tab? ? (click)Ctrl + Shift + (click)**REQUEST** Request URL?  
LCtrl + LSave Request? SCtrl + SSave Request As? ? SCtrl + Shift + SSend Request? ?Ctrl  
+ EnterSend And Download Request? ? ?Ctrl + Alt + EnterScroll To Request? ? ↑Ctrl + Alt +  
↑Scroll To Response? ? ↓Ctrl + Alt + ↓Beautify raw Request body? BCtrl + B**SIDE BAR**  
Search Sidebar? FCtrl + FToggle Sidebar? \Ctrl + \Next Item↓\Previous Item↑↑Expand  
Item→→Collapse Item←←Select Item?EnterRename Item? ECtrl + EGroup Items? GCtrl +  
GCut Item? XCtrl + XCopy Item? CCtrl + CPaste Item? VCtrl + VDuplicate Item? DCtrl +  
DDelete Item? Del**INTERFACE** Zoom In? +Ctrl + +Zoom Out? -Ctrl + -Reset Zoom? 0Ctrl +  
0Toggle Two-Pane View? ? VCtrl + Alt + VSwitch To Sidebar? ? 1Ctrl + Alt + 1Switch To  
Builder? ? 2Ctrl + Alt + **2WINDOWS AND MODALS** New Requester Window? NCtrl +  
NNew Runner Window? ? NCtrl + Shift + NNew Console Window? ? CCtrl + Alt + CImport?  
OCtrl + OManage Environments? ? ECtrl + Alt + ESettings? ,Ctrl + ,Submit Modal? ?Ctrl +  
EnterOpen Shortcut Help? /Ctrl + /

## Data editor

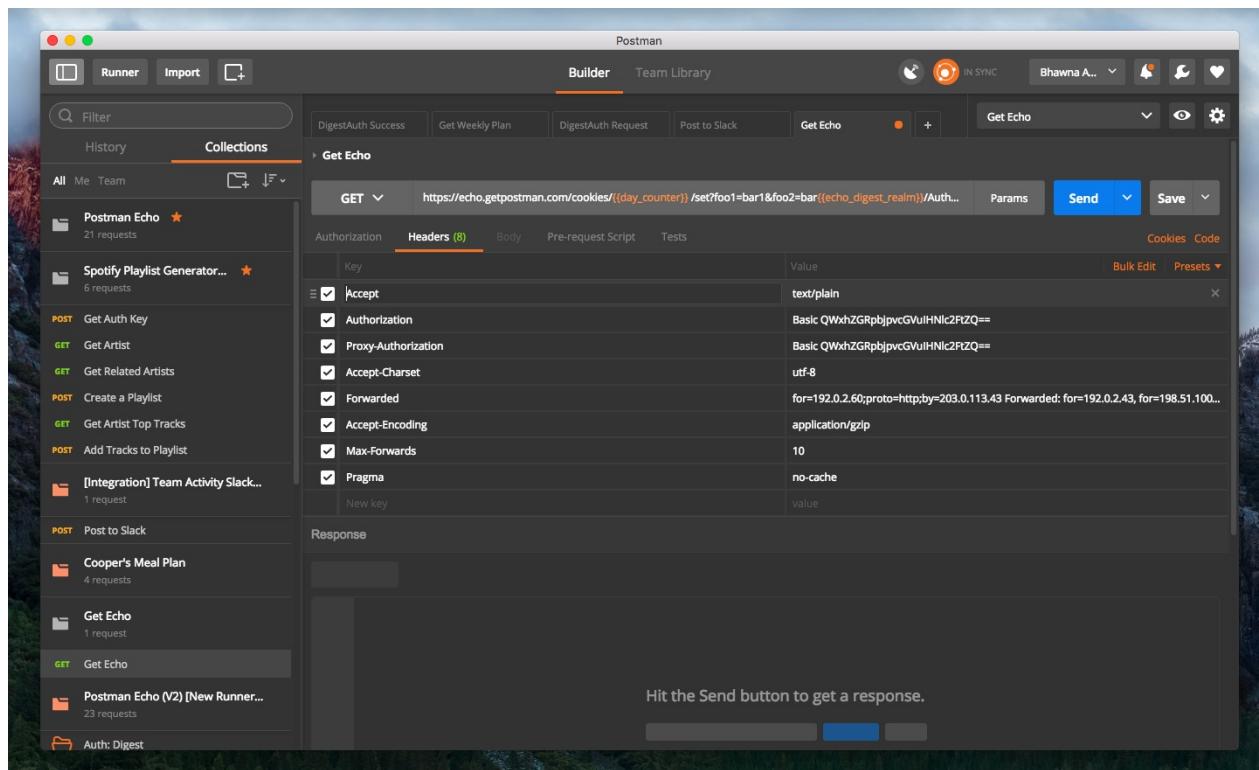
Working with large amounts of data can be cumbersome and time consuming. Postman's data editor lets you view and manipulate data in a fast, effective, and elegant manner.

Spreadsheet features are familiar to most people working with data, so we used them as inspiration for the design of the data editor.

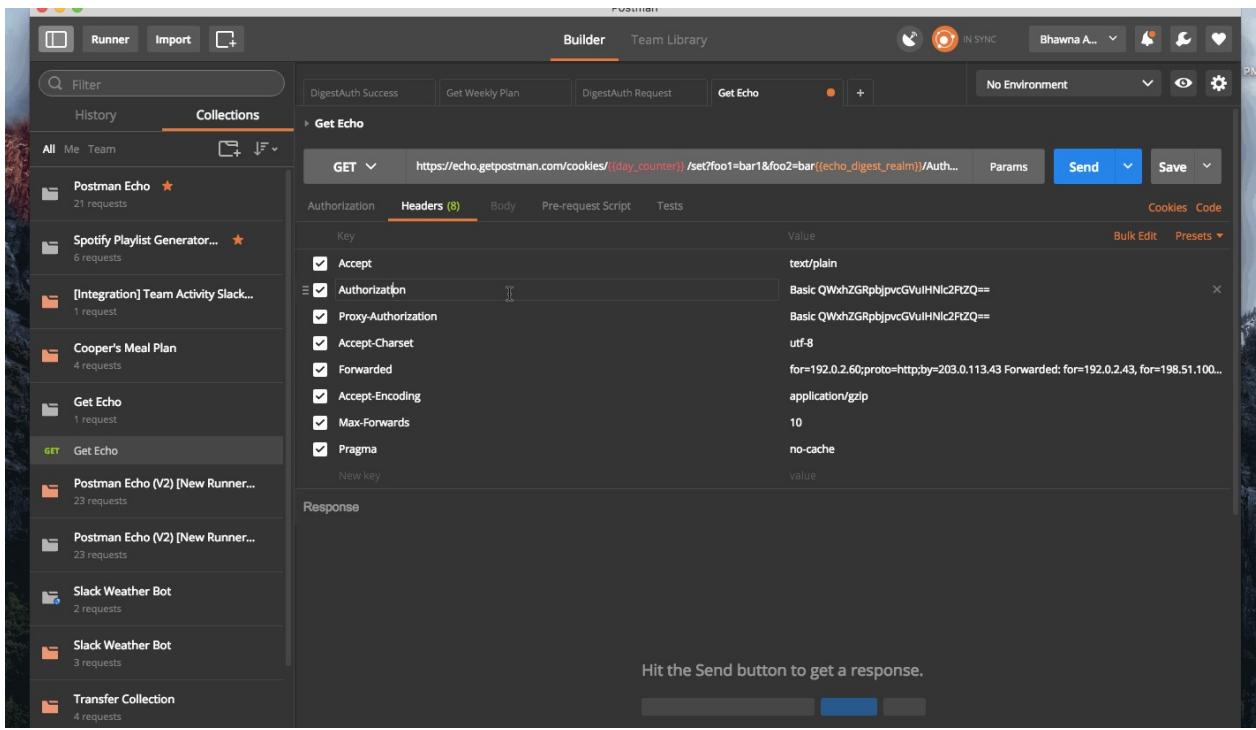
### Visual layout

We've configured the available horizontal and vertical space more effectively to optimize the data that can be displayed up front.

Relevant features will display for a specific row on hover. This reduces clutter in the interface and helps the user focus on the most relevant data.

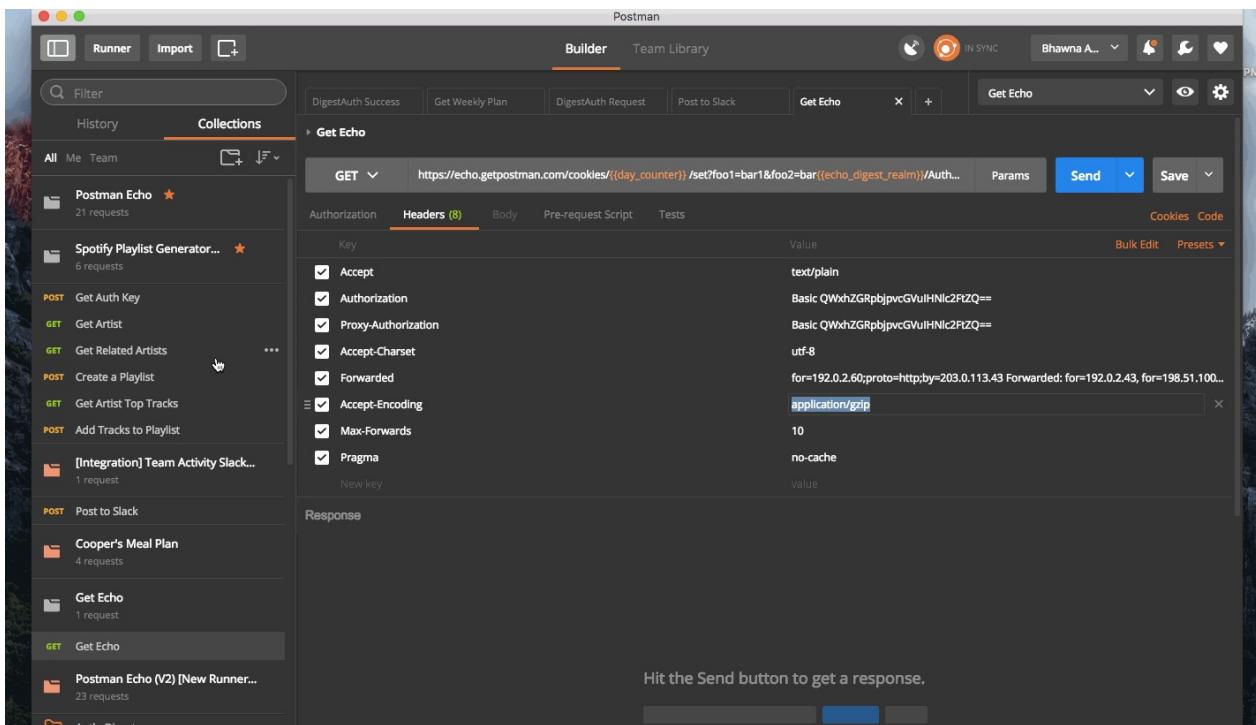


**Select multiple rows by simply dragging your mouse**



## Keyboard shortcuts for the data editor

Data editormacOSWindows / Linux shortcutsNavigation  
 arrow keys ( $\uparrow, \rightarrow, \downarrow, \leftarrow$ ) + Tab  
 keys ( $\uparrow, \rightarrow, \downarrow, \leftarrow$ ) + Tab  
 Duplicate row? DCtrl + D  
 Select specific rows? (click)Ctrl + (click)  
 Select previous rows?  $\uparrow$ Shift +  $\uparrow$ Select next rows?  $\downarrow$ Shift +  $\downarrow$ Select current row?  $\rightarrow$ Shift +  $\rightarrow$ Select current row?  $\leftarrow$ Shift +  $\leftarrow$ Move row(s) up? ?  $\uparrow$ Ctrl + Shift +  $\uparrow$ Move row(s) down? ?  $\downarrow$ Ctrl + Shift +  $\downarrow$ Copy - can multiselect and copy rows? CCtrl + C  
 Cut - can multiselect and cut rows? XCtrl + X  
 Paste? VCtrl + V  
 Delete - can multiselect and delete rows? Del  
 Deselect rows? Esc



## Support for bulk actions

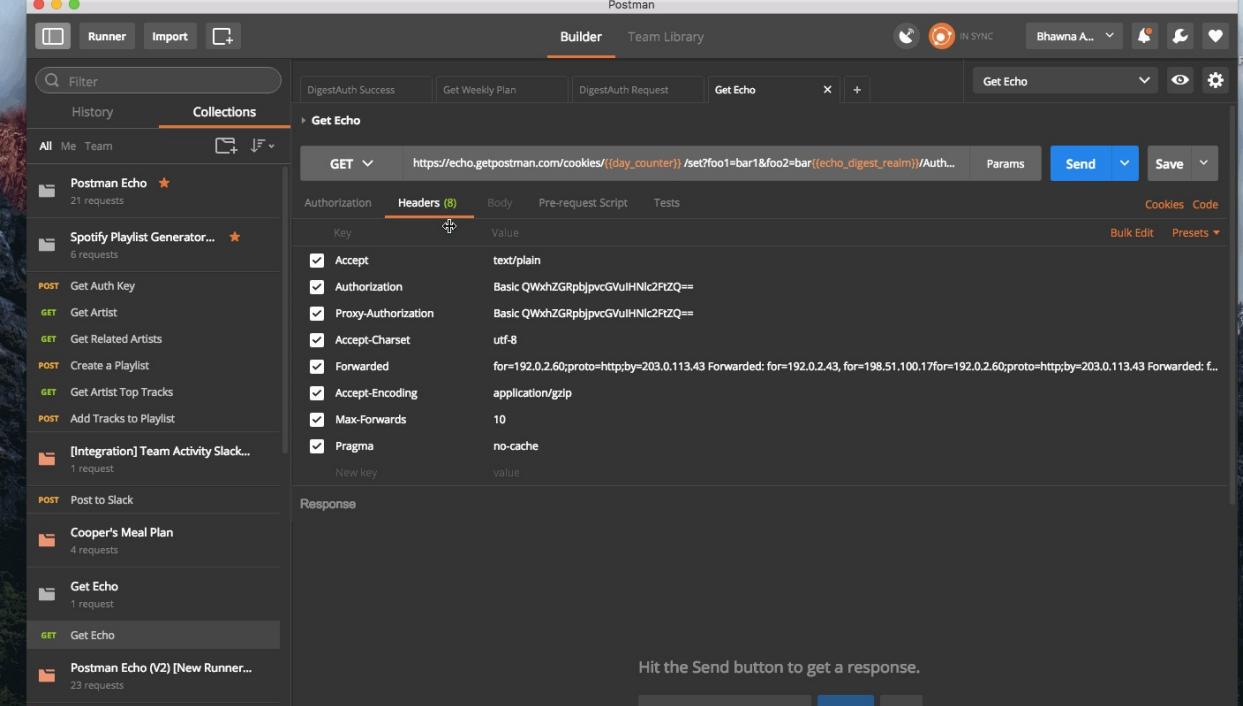
You can select and copy multiple rows. Then go ahead and paste them in a different place (e.g. params) and check out what happens for yourself!

## View information up front

We want to display the information you want to see up front. If you navigate to a place with a large amount of data, the UI element auto expands to show the complete information. This is implemented in the data editor as well as the URL bar.

## Ability to resize columns

You can change the width of the key and value columns by dragging the boundary on the right side of the ‘key’ column header.



Key	Value
Accept	text/plain
Authorization	Basic QWxhZGRpbjpvGVuIHNlc2RZQ==
Proxy-Authorization	Basic QWxhZGRpbjpvGVuIHNlc2RZQ==
Accept-Charset	utf-8
Forwarded	for=192.0.2.60;proto=http;by=203.0.113.43 Forwarded: for=192.0.2.43, for=198.51.100.17for=192.0.2.60;proto=http;by=203.0.113.43 Forwarded: f...
Accept-Encoding	application/gzip
Max-Forwards	10
Pragma	no-cache

## Multiline support

The data editor supports sending multiline values. Pressing ‘Enter’ in a key or value field expands it and takes the cursor to a new line.

# Postman account

## Why sign up for a Postman account

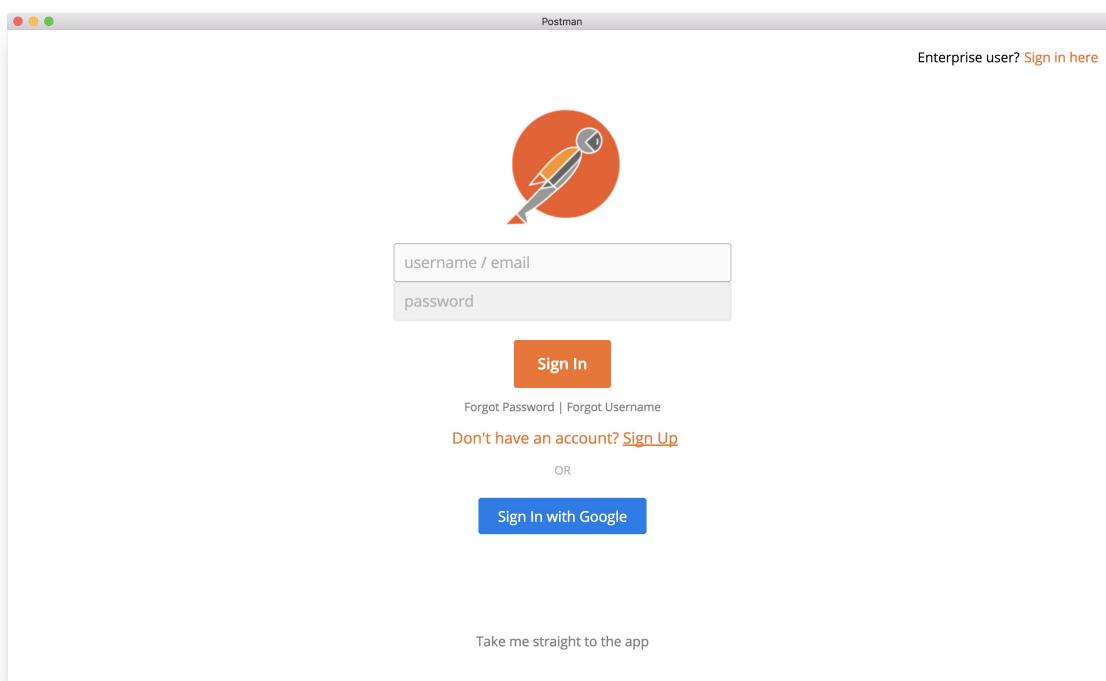
Sign up for a free Postman account and get the following benefits:

- [Sync](#) and back up your history, collections, environments and header presets.
- Easily work on multiple Postman instances from different machines.
- Create [collection links](#) to send to other developers.

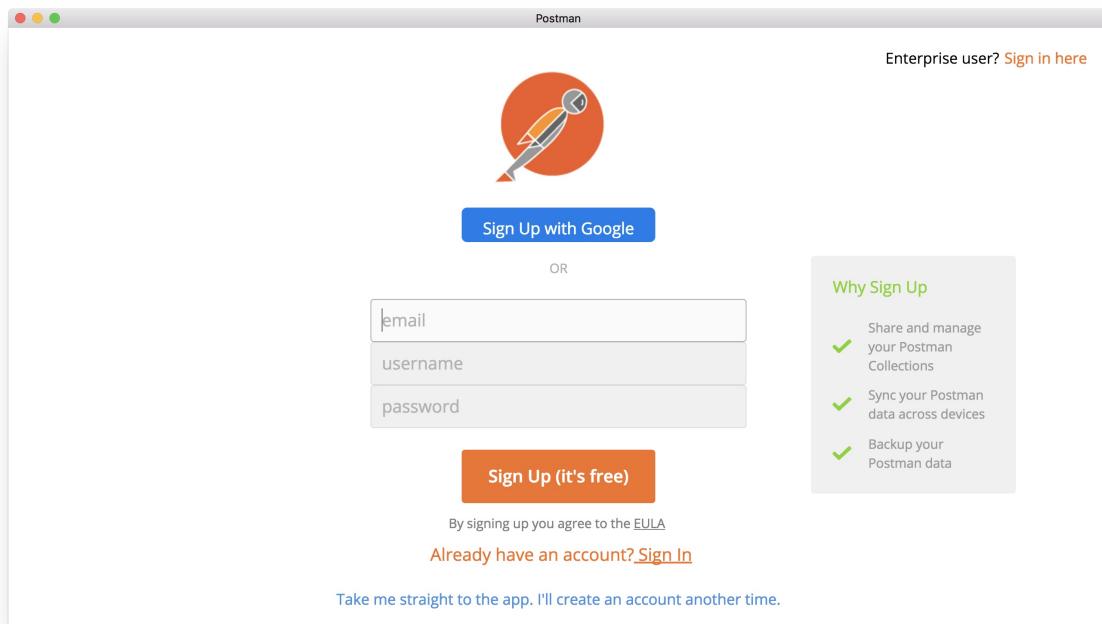
Signing up is completely optional. Read our [EULA](#) and our [security page](#) and [privacy page](#) to know more.

## Signing up for a Postman account

- If you haven't already, download the Postman app.
- Launch the app, and see a prompt to log in or sign up.

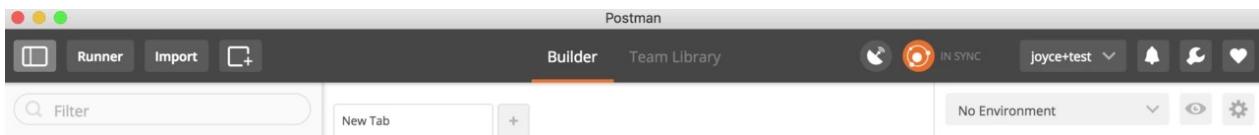


- Sign up with your email address or your Google account.



- Confirm your email and you are good to go.

Once you are logged in, you will see the **IN SYNC** icon at the top telling you that you are connected to our servers. Postman uses WebSockets for real-time [syncing](#). If you are experiencing issues with syncing, you can file an issue via our [support center](#).



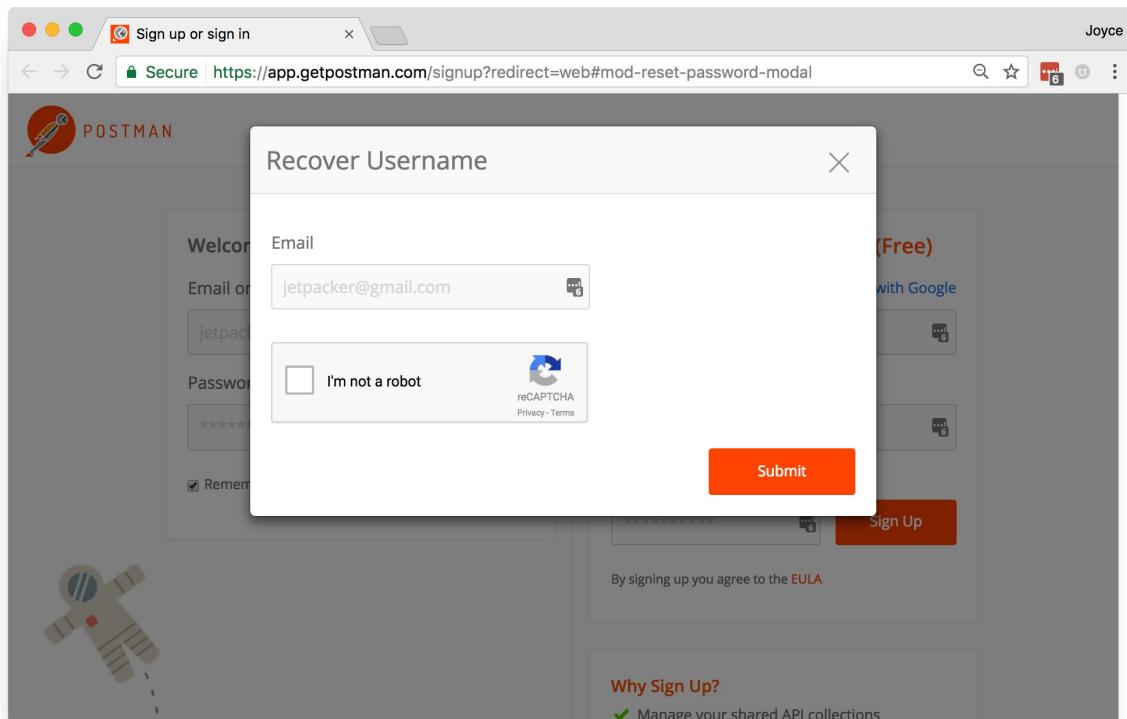
## Security policies and standards

Postman uses several technologies to ensure that your data is safe and secure. For more details, visit our [security page](#).

- Communication with Postman servers only happens through HTTPS and Secure WebSockets.
- Environment and global variables are encrypted so that only you can view them.
- You can turn off sync any time and still use Postman.

## Forgot your username or password?

Clicking on the Forgot Username or Password links near the **Sign In** prompt will allow you to [recover your username or reset a password](#).



# Syncing

## What is Syncing?

Syncing is a process that makes all your Postman data available wherever you're signed in to your Postman account.

Any changes (edits, additions, deletions) you make will be synced across all devices linked to your account.

The following entities are synced with our server and saved to the cloud:

- Collections
- Folders
- Requests
- Responses
- Header Presets
- Environments
- Environment variables
- Global variables
- Collection run results

## How do I Sync between computers?

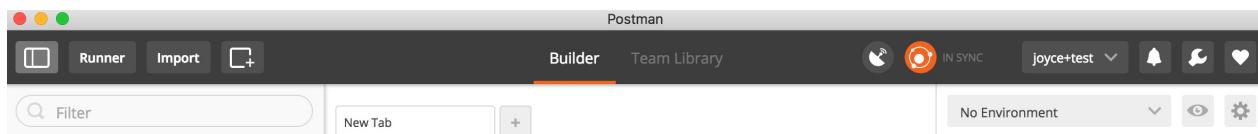
Install the [Postman app](#) and sign in using the same email address, or username, on all your devices. If you forget your username or password, you can recover your username and reset your password by clicking on the links near the sign in prompt. If you have sync enabled, all the data you now create (or have created in the past) will be synced across all your devices.

Postman automatically makes sure your data is the same no matter where you access it. No further settings are required.

**Note:** Postman restricts parallel usage to 3 apps per account.

### States

If your app is in sync with our server, the icon to the left of your name in the top navigation will be orange, and say **IN SYNC**. If a sync operation is underway, it will say **SYNCING**. If you are not signed in to the app, you will see **OFFLINE**. When you open the app or if your connection is dropped, it will say **CONNECTING** before it starts syncing.



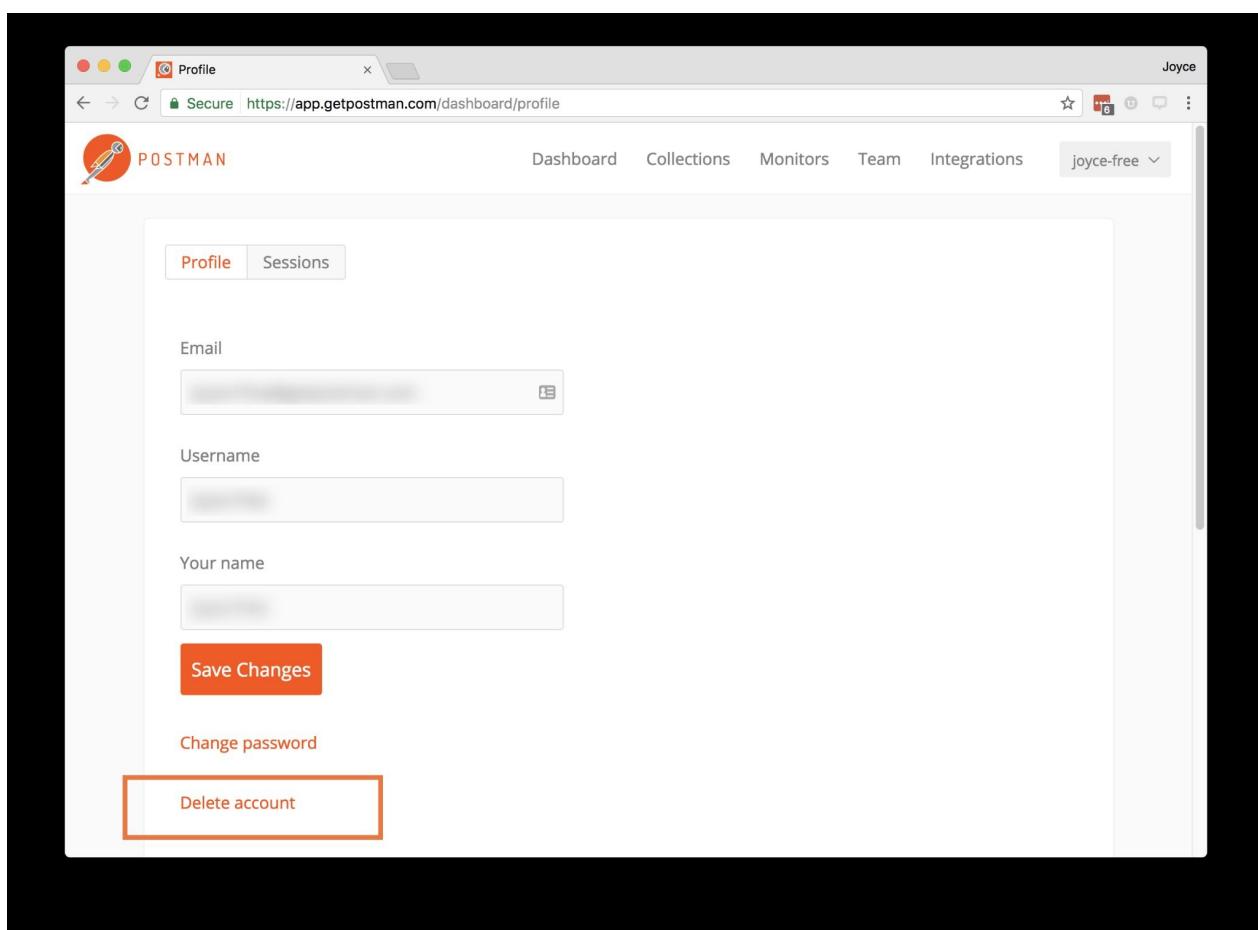
## Sign Out

If you choose to sign out, your data will be deleted from the local instance of the Postman app. Don't worry, all your data will be restored from the cloud when you sign in. This is just meant to enable other users to use the app without your data clashing.

When you reload the app, Postman will automatically retrieve the most recent and up-to-date version of your collections.

## Delete your Postman account

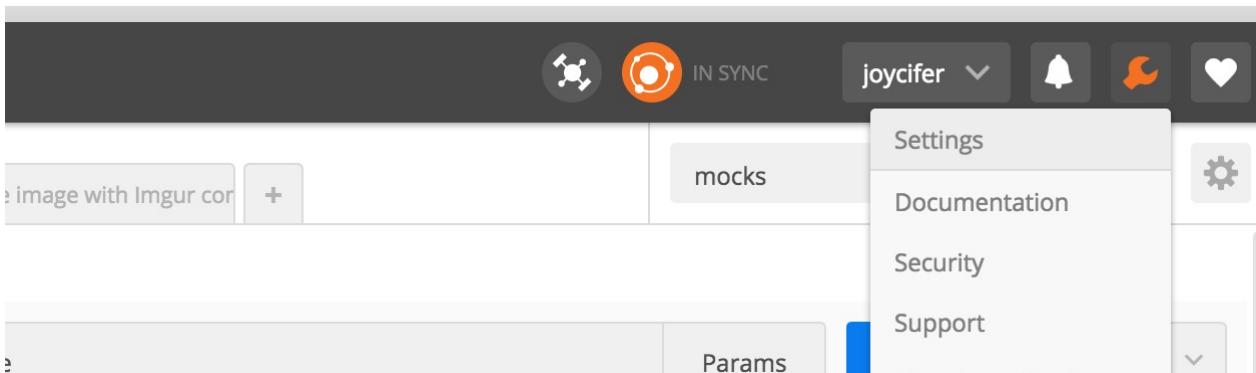
If you have a Postman account, are not currently part of a paid Postman Pro or Enterprise team, and have never taken part in a Postman Pro trial team, you can [delete your account here](#). Otherwise, you can contact us at [help@getpostman.com](mailto:help@getpostman.com).



# Settings

## Getting to the Settings

In the header toolbar of the Postman app, click the wrench icon and select “Settings” to open the **SETTINGS** modal. You can also use the keyboard shortcut (**CMD/CTRL + ,**) to open the modal.



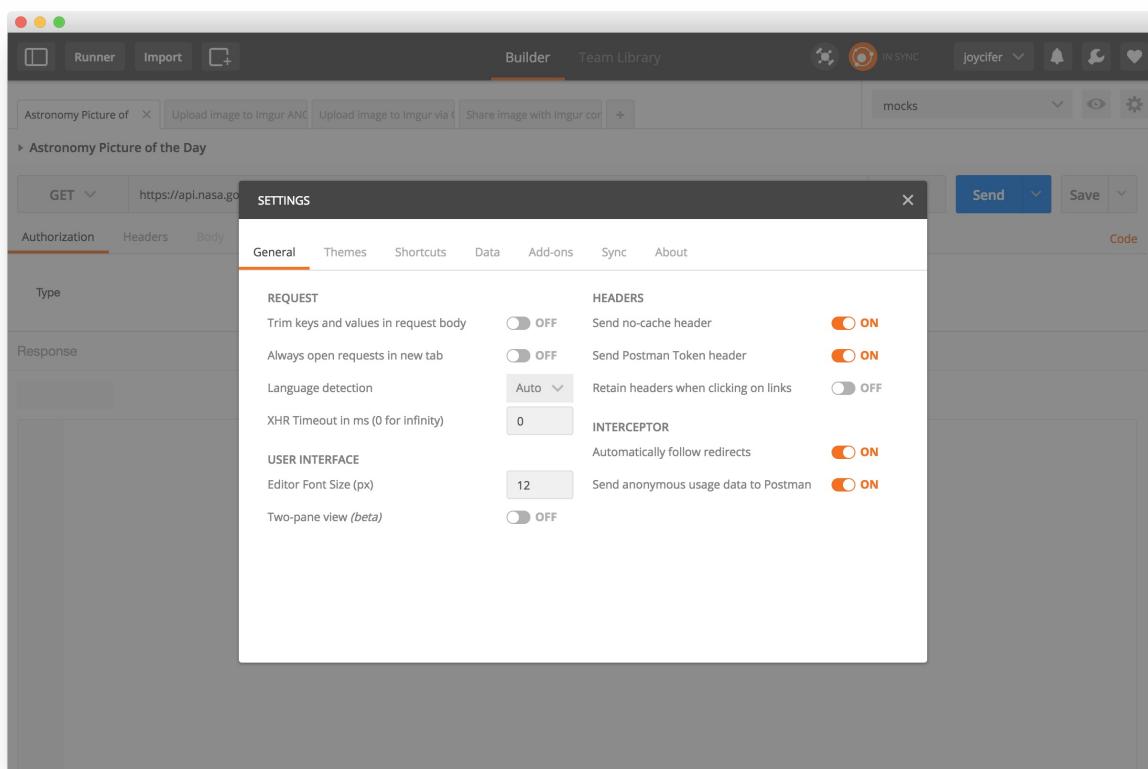
## General Settings

Postman tries to minimize the number of settings you have to change, so we established some defaults. However, given the diversity of use cases, if you need to make adjustments, here's how:

- **Trim keys and values in request body:** If you're using the form-data or url-encoded modes to send data to the server, switching this to “ON” will cause any parameters to be trimmed.
- **SSL certificate verification** (native apps only): Prevents the app from checking validity of SSL certificates while making a request. Read more about [managing client certificates](#).
- **Language detection:** Setting this to JSON will force a JSON rendering, irrespective of the response Content-Type header.
- **XHR Timeout in ms:** Set how long the app should wait for a response before saying that the server isn't responding. A value of 0 indicates infinity - Postman will wait for a response forever.
- **Editor Font Size:** Adjust the font size in pixels for the text that appears in Postman.
- **Two-pane view:** Toggle between showing the response below, or beside, the request.
- **Send no-cache header** (recommended): Sending a no-cache header makes sure you get the freshest response from your server.
- **Send Postman Token header:** This is primarily used to bypass a bug in Chrome. If an XMLHttpRequest is pending and another request is sent with the same parameters then

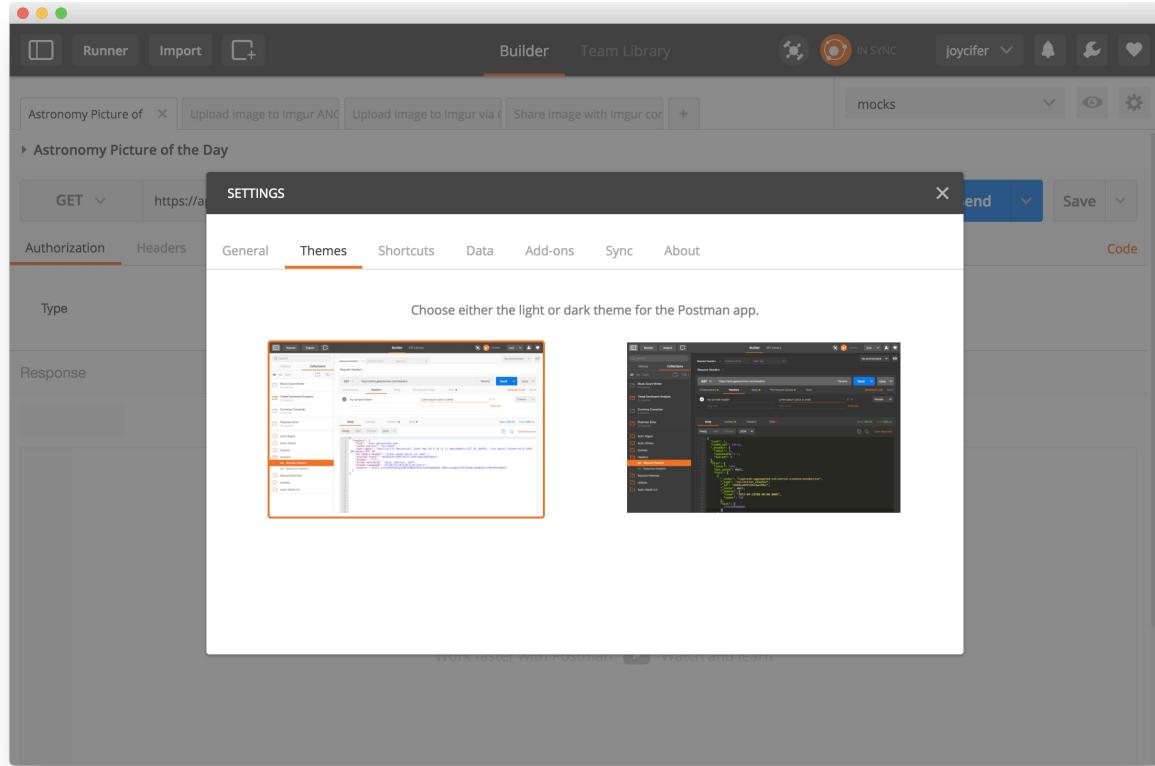
Chrome returns the same response for both of them. Sending a random token avoids this issue. This can also help you distinguish between request on the server side.

- **Retain headers when clicking on links:** If you click on a link in a response, Postman creates a new GET request with that URL. If you want to retain the headers that you set in the previous request set “ON” here. This is useful if you are accessing mainly protected resources.
- **Automatically follow redirects:** Prevent requests that return a 300-series response from being automatically redirected.
- **Send anonymous usage data to Postman:** Option to disable sending basic anonymous usage data (button clicks and app events) to Postman. We use usage data to make Postman a better product.



## Themes

Pick your pleasure: choose a light or dark theme for the Postman app.



## Keyboard Shortcuts

This is where you can view keyboard shortcuts available for your operating system here. Learn more about using these [shortcuts to increase your productivity](#).

## Data Import / Export

Import and export data in bulk inside Postman. This will overwrite your existing collections and environments so be a little careful. It always helps to take a backup before you are importing other files. Learn more about [importing and exporting data](#) in Postman.

## Add-ons

Download Newman, Postman's command line companion, to integrate Postman collections with your build system, or run automated tests for your API through a cron job. Learn more about [Newman](#).

## Sync

If you are signed in to Postman, your data is synced with our server, making sure you have it all next time you use the app (and not just locally). You can force re-sync or disable it under **Settings**. Learn more about [syncing](#).

## Certificates

Add and view client certificates on a per domain basis. Learn more about [setting certificates](#).

## Update

Postman's native apps will notify you whenever a version update is available. To force a check for updates, head to the **Update** tab of the **SETTINGS** modal. Learn about [updating the Postman app](#).

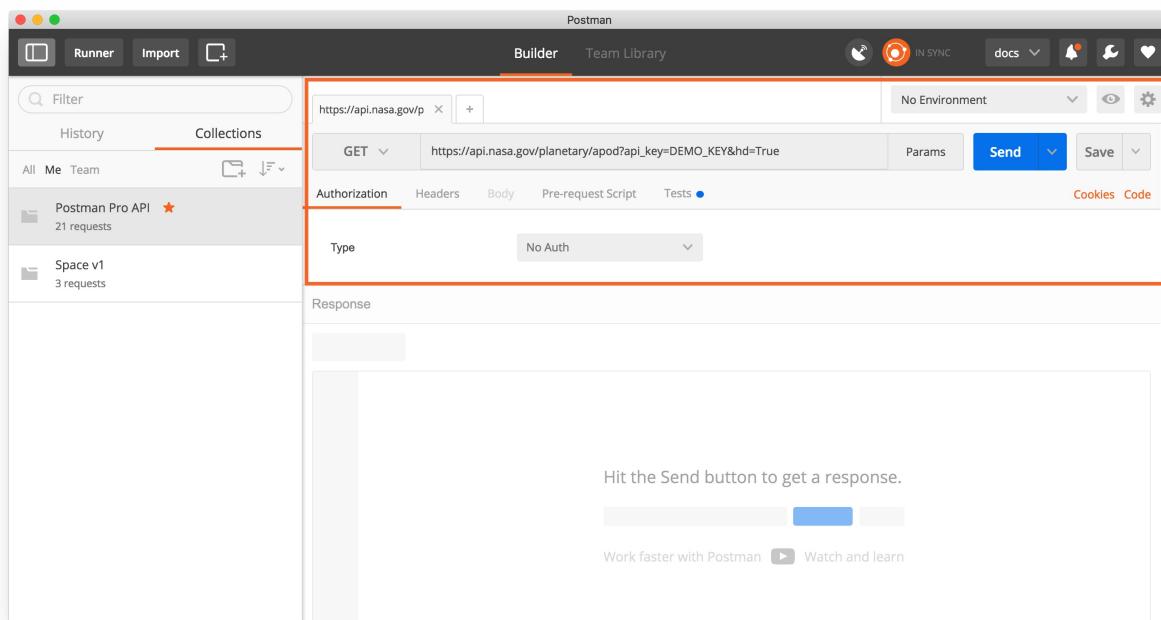
## About

This is where you can verify your current version of the Postman app. There are also some helpful support links to reference.

# Requests

## Request builder

Under the **Builder** tab, the request builder lets you create any kind of HTTP request quickly. The four parts of an HTTP request are the URL, method, headers, and the body. Postman gives you tools to work with each of these parts.



## URL

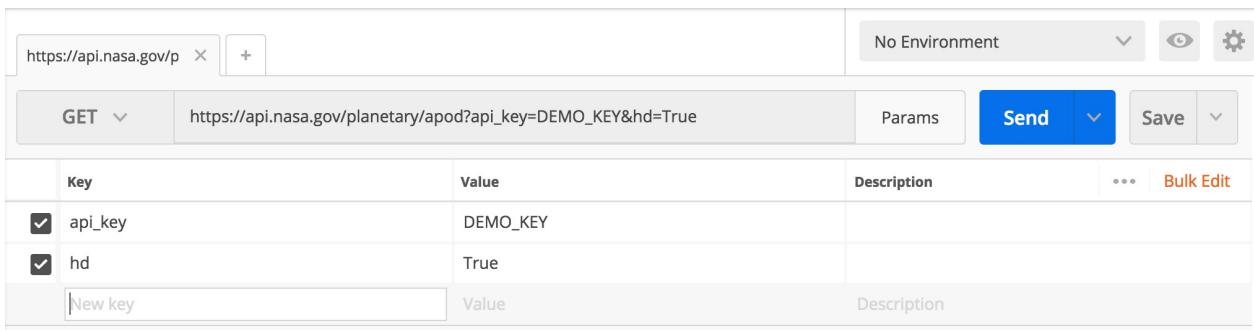
The URL is the first thing that you would be setting for a request. The URL input field stores previously-used URLs and will show an autocomplete dropdown as you begin entering your URL.

Clicking on the **Params** button opens up the [data editor](#) for entering URL parameters. You can individually add key-value pairs and Postman will combine everything in the query string above. If your URL already has parameters - for example, if you are pasting a URL from some other source, Postman will split the URL into pairs automatically.

**Note:** Parameters you enter in the URL bar or in the data editor will not automatically be URL-encoded. Right click on a piece of selected text, and select “EncodeURIComponent” to manually encode the parameter value.

**Note:** Postman will automatically add http:// to the beginning of the URL if no protocol is specified.

## 发送 API 请求

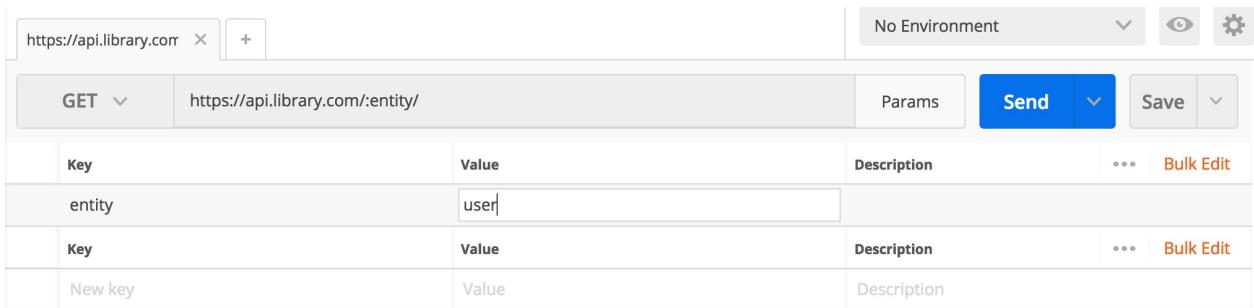


The screenshot shows the Postman interface with a GET request to `https://api.nasa.gov/planetary/apod`. The URL bar shows the full URL with parameters `api_key=DEMO_KEY&hd=True`. The 'Params' tab is selected, displaying two entries: `api_key` with value `DEMO_KEY` and `hd` with value `True`. There is also a placeholder row for a new key.

Some API endpoints use path variables. You can work with those in Postman. Below is an example of a URL with a path variable:

```
https://api.library.com/:entity/
```

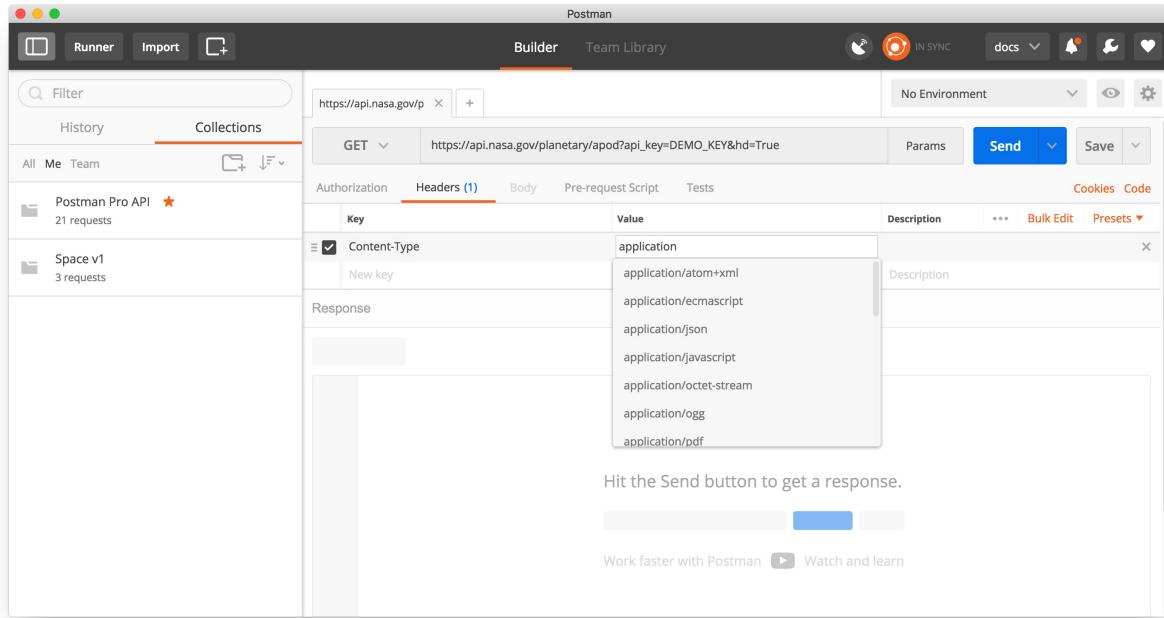
To edit the path variable, click on **Params** to see it already entered as the key. Update the value as needed. For example, `:entity` can be “user” in this specific case. Postman will also give you suggestions to autocomplete the URL.



The screenshot shows the Postman interface with a GET request to `https://api.library.com/:entity/`. The URL bar shows the full URL with a path variable `:entity` set to `user`. The 'Params' tab is selected, displaying one entry: `entity` with value `user`. There is also a placeholder row for a new key.

## Headers

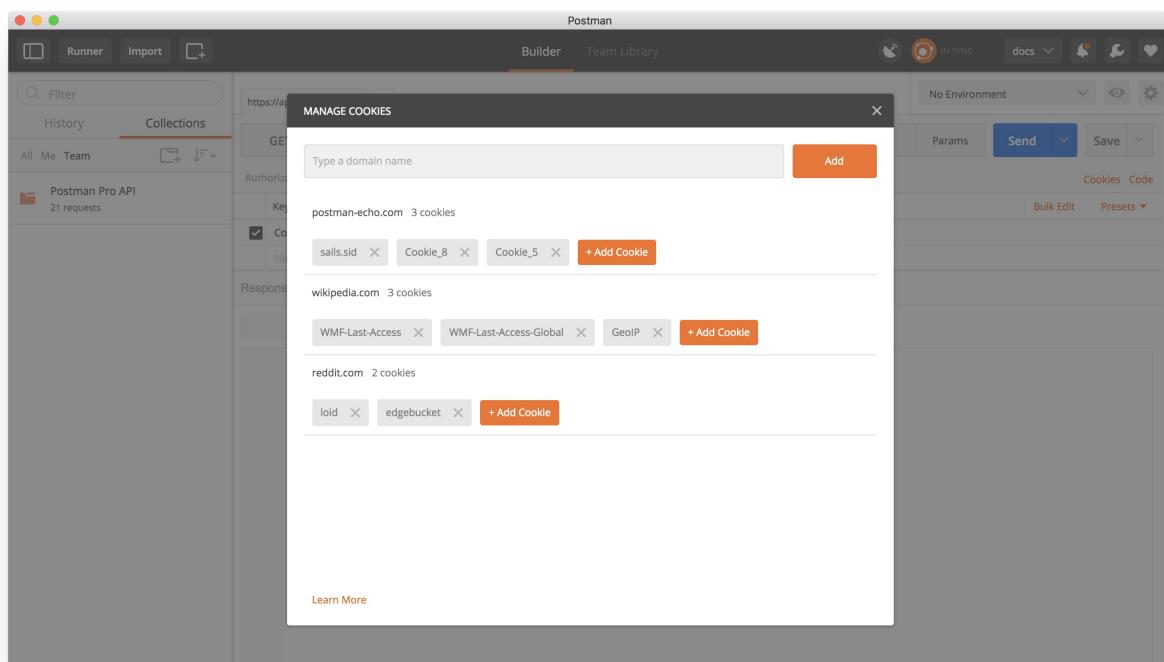
Clicking on the **Headers** tab will show the headers key-value editor. You can set any string as the header name. The autocomplete dropdown will provide suggestions of common HTTP headers as you type in the fields. Values for the “Content-Type” header are also available in an auto-complete drop down.



**Note on restricted headers:** If you're using the Postman Chrome app, some headers are restricted by Chrome and the XMLHttpRequest specification. However, sending restricted headers is simple using the [Interceptor extension](#).

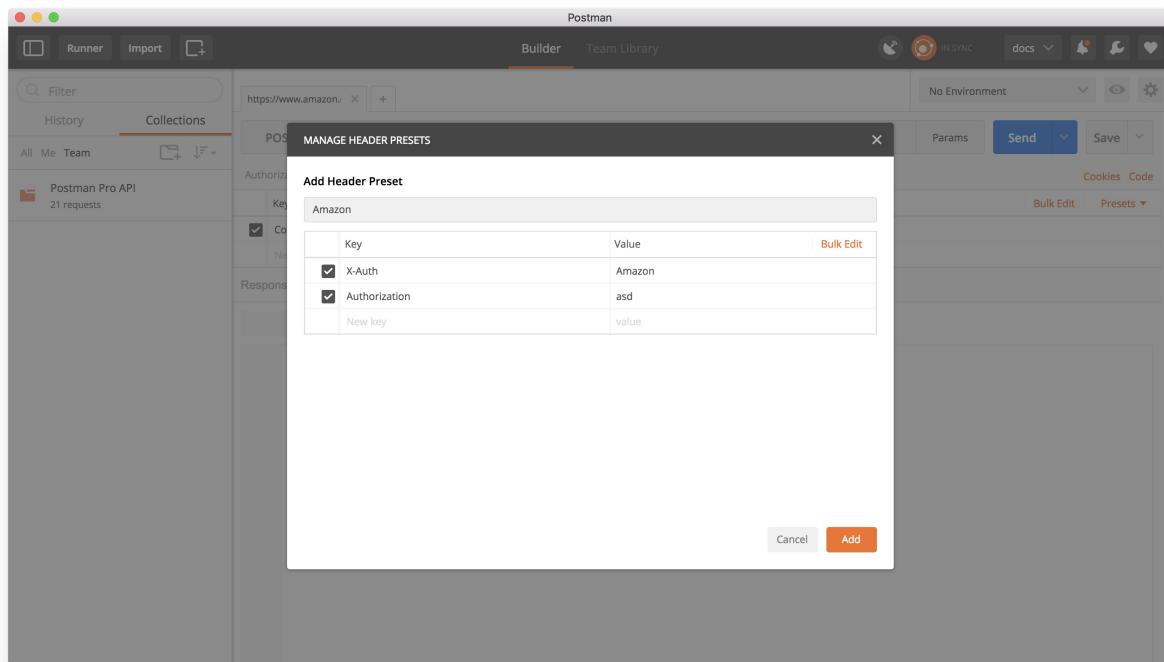
## Cookies

Cookies can be managed in native apps by using the cookie manager to edit cookies associated with each domain. To open the modal, click the **Cookies** link under the **Send** button. Learn more about [managing cookies](#).



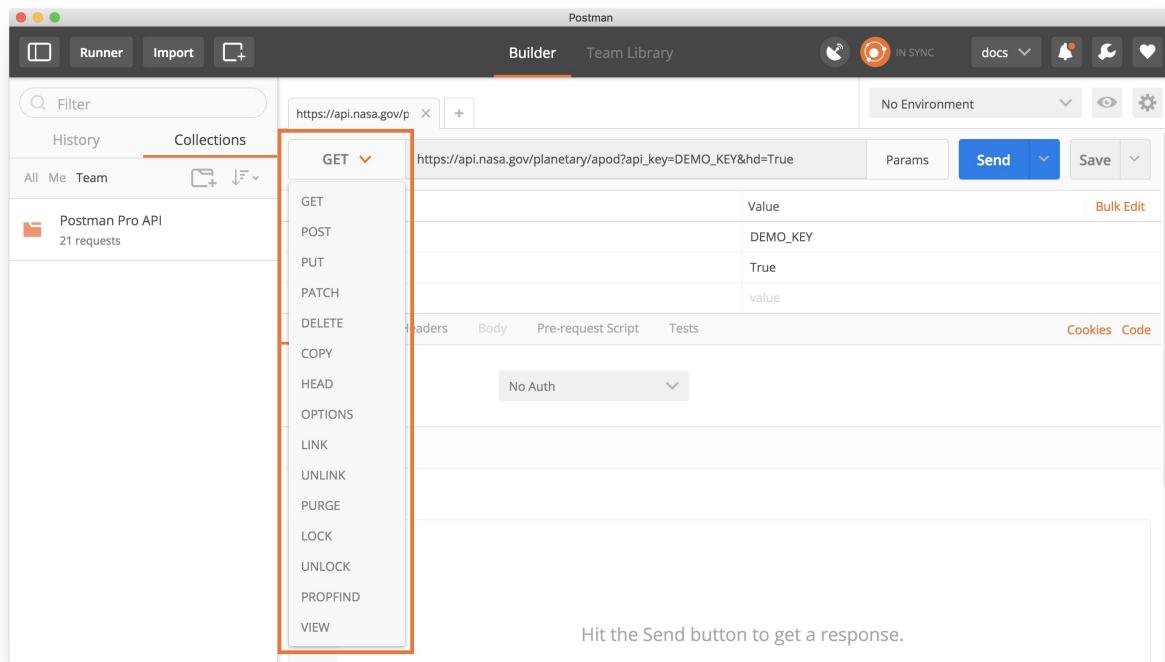
## Header presets

You can save commonly used headers together in a header preset. Under the **Headers** tab, you can add a header preset to your request by selecting “Manage Presets” from the **Presets** dropdown on the right.



## Method

Changing the request method is straightforward, using the control dropdown. The request body editor area will change depending on whether the method can have a body attached to it or not.



## Request Body

While constructing requests, you will be working with the request body editor a lot. Postman lets you send almost any kind of HTTP request. The body editor is divided into 4 areas and has different controls depending on the body type.

**Note about Headers:** When you are sending requests through the HTTP protocol, your server might expect a Content-Type header. The Content-Type header allows the server to parse the body properly. For form-data and urlencoded body types, Postman automatically attaches the correct Content-Type header so you don't have to set it. The raw mode header is set when you select the formatting type. If you manually use a Content-Type header, that value takes precedence over what Postman sets. Postman does not set any header type for the binary body type.

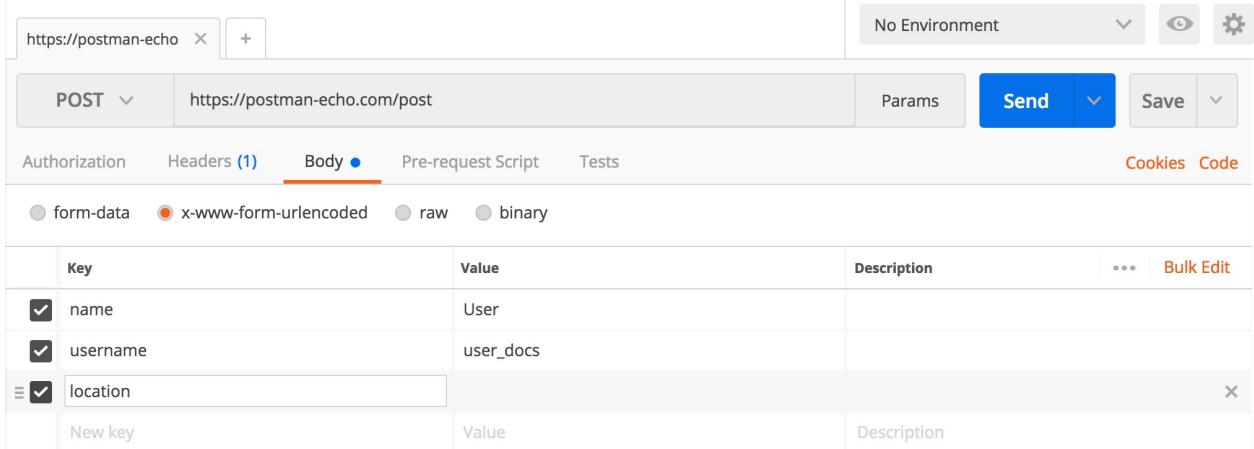
### Form-data

Key	Value	Description	...	Bulk Edit
size	original			
file	Choose Files No file chosen			

multipart/form-data is the default encoding a web form uses to transfer data. This simulates filling a form on a website, and submitting it. The form-data editor lets you set key-value pairs (using the [data editor](#)) for your data. You can attach files to a key as well. Note: due to restrictions of the HTML 5 spec, files are not stored in history or collections. You will need to select the file again the next time you send the request.

Uploading multiple files each with their own Content-Type is not supported yet.

## Urlencoded

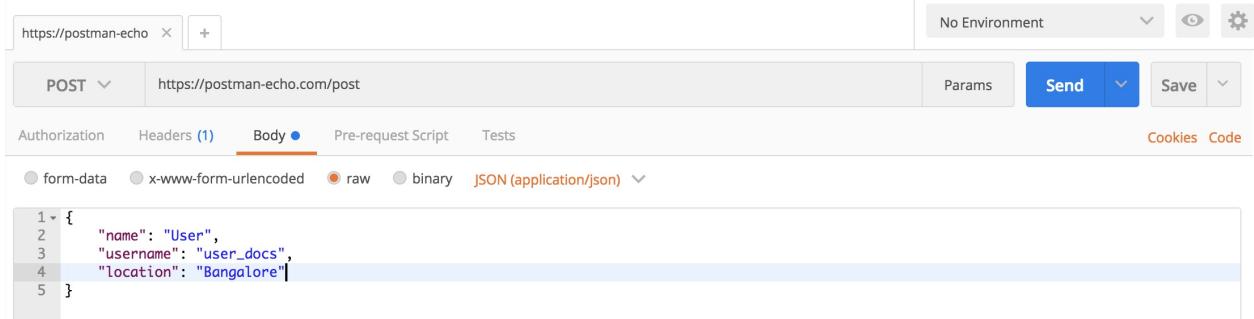


The screenshot shows the Postman interface with a POST request to <https://postman-echo.com/post>. The 'Body' tab is selected, showing the following key-value pairs:

Key	Value	Description
<input checked="" type="checkbox"/> name	User	
<input checked="" type="checkbox"/> username	user_docs	
<input checked="" type="checkbox"/> location		
New key		Description

This encoding is the same as the one used in URL parameters. You just need to enter key-value pairs and Postman will encode the keys and values properly. Note that you cannot upload files through this encoding mode. There might be some confusion between form-data and urlencoded so make sure to check with your API first.

## Raw



The screenshot shows the Postman interface with a POST request to <https://postman-echo.com/post>. The 'Body' tab is selected, showing the following JSON content:

```

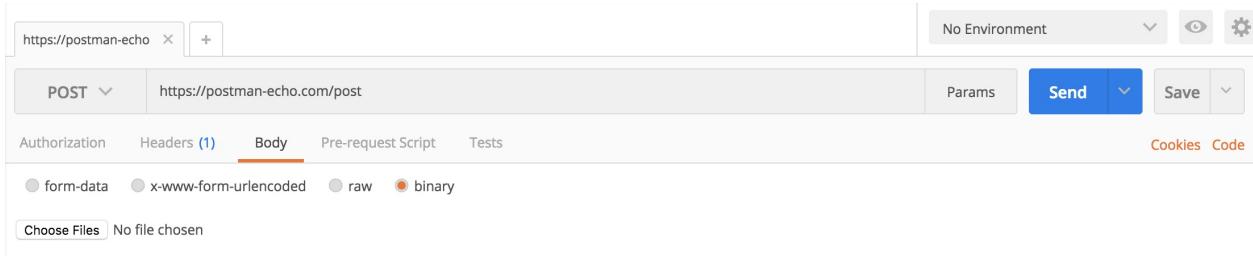
1 {
2   "name": "User",
3   "username": "user_docs",
4   "location": "Bangalore"
5 }

```

A raw request can contain anything. Postman doesn't touch the string entered in the raw editor except replacing [environment variables](#). Whatever you put in the text area gets sent with the request. The raw editor lets you set the formatting type along with the correct header that you should send with the raw body. You can set the Content-Type header manually too and this will override the Postman defined setting. Selecting XML/JSON in the editor type enables syntax highlighting for your request body and also sets the Content-Type header.

**Tip:** Selecting text in the editor and pressing **CMD/CTRL + B** can beautify the XML/JSON content automatically.

## Binary



The screenshot shows the Postman application interface. At the top, there is a header bar with a URL field containing "https://postman-echo.com/post", a "No Environment" dropdown, and several icons for settings and sharing. Below the header is a toolbar with "POST" (selected), "Params", "Send" (highlighted in blue), and "Save". The main workspace is divided into sections: "Authorization", "Headers (1)", "Body" (which is selected and underlined in orange), "Pre-request Script", and "Tests". Under the "Body" section, there are four radio button options: "form-data", "x-www-form-urlencoded", "raw", and "binary". The "binary" option is selected, indicated by a red dot. Below these options is a "Choose Files" button with the placeholder text "No file chosen".

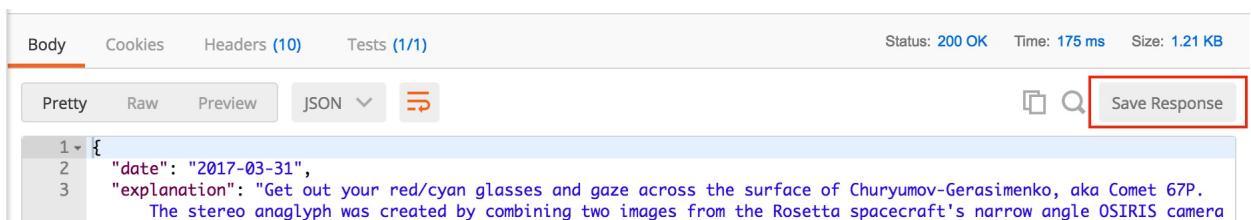
Binary data allows you to send things which you can not enter in Postman, for example, image, audio, or video files. You can send text files as well. As mentioned earlier in the form-data section, you would have to reattach a file if you are loading a request through the history or the collection.

# Responses

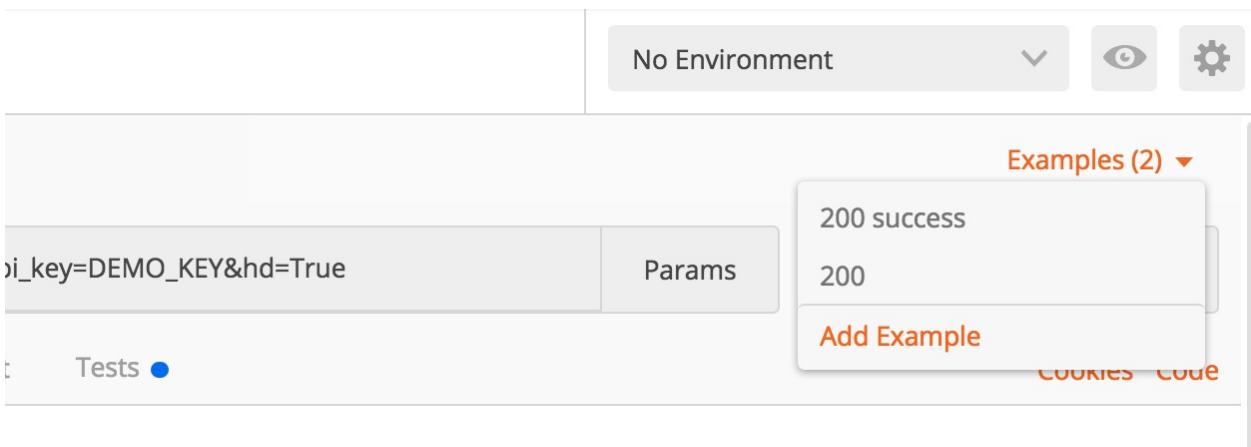
Ensuring that the API response is correct is something that you will be doing a lot when working with APIs. The Postman response viewer will make this task much easier for you.

An API response consists of the body, headers, and the status code. Postman organizes body and headers in different tabs. The status code with the time taken to complete the API call is displayed next to the tabs. You can hover over the status code to get more details about the code. Mostly it will be the default description as mandated by the HTTP specification, however, API authors can also add custom messages.

## Saving responses



If a request has been saved in a collection, you can save responses for that request. Once the response has been returned, click the **Save Response** button. Enter a name to call your saved response. All responses saved for a request will be available as an [example](#) whenever you load the request. Click the **Examples** dropdown in the top right to view and select the saved examples.



## Viewing responses

The Postman **Body** tab gives you several tools to help you make sense of things quickly. The body can be viewed in one of three views - pretty, raw, and preview.

### Pretty

The screenshot shows the Postman interface with the 'Body' tab selected. Below it, there are three tabs: 'Pretty' (which is highlighted with a red box), 'Raw', and 'Preview'. To the right of these tabs are buttons for 'JSON' (with a dropdown arrow) and a copy icon. On the far right, there are icons for saving and deleting the response, along with a search bar and a 'Save Response' button. The main content area displays a JSON object with several properties. The 'date' property has a value of "2017-03-31". The 'explanation' property contains a detailed description of a comet's surface, mentioning the Rosetta spacecraft and its mission to Comet 67P. A URL is also provided: "http://apod.nasa.gov/apod/image/1703/AP00-67P-Seth.jpg". The JSON object ends with a closing brace '}'.

```

1 {
2   "date": "2017-03-31",
3   "explanation": "Get out your red/cyan glasses and gaze across the surface of Churyumov-Gerasimenko, aka Comet 67P. The stereo anaglyph was created by combining two images from the Rosetta spacecraft's narrow angle OSIRIS camera taken on September 22, 2014. Stark and jagged, the 3D landscape is found along the Seth region of the comet's double-lobed nucleus. It spans about 985 x 820 meters, pocked by circular ridges, depressions, and flattened areas strewn with boulders and debris. The large steep-walled circular pit in the foreground is 180 meters in diameter. Rosetta's mission to the comet ended in September 2016 when the spacecraft was commanded to a controlled impact with the comet's surface.",
4   "hdurl": "http://apod.nasa.gov/apod/image/1703/AP00-67P-Seth.jpg"
}

```

The pretty mode formats JSON or XML responses so that they are easier to look at. Nobody wants to scroll through a minified single line JSON response looking for that elusive string! Links inside the pretty mode are highlighted and clicking on them can load a GET request in Postman with the link URL. For navigating large responses, click on the down-pointing triangles (▼) on the left to collapse large sections of the response.

For Postman to automatically format the body, make sure the appropriate Content-Type header is returned. If the API does not do this, then you can force formatting through JSON or XML. You can force JSON formatting under the **General** tab within the **SETTINGS** modal by selecting “JSON” from the “Language detection” dropdown.

**Finding items in responses:** You can use **CMD/CTRL + F** to open the search bar, and **CMD/CTRL + G** to scroll through results. See complete set of [keyboard shortcuts](#).

## Raw

The screenshot shows the Postman interface with the 'Body' tab selected. Below it, there are three tabs: 'Pretty' (highlighted with a red box), 'Raw' (which is highlighted with a red box), and 'Preview'. To the right of these tabs are buttons for 'JSON' (with a dropdown arrow) and a copy icon. On the far right, there are icons for saving and deleting the response, along with a search bar and a 'Save Response' button. The main content area displays the same JSON object as the previous screenshot, but in raw mode. The JSON is presented as a single block of text, making it difficult to read the explanation text due to line wrapping.

```
{
  "date": "2017-03-31",
  "explanation": "Get out your red/cyan glasses and gaze across the surface of Churyumov-Gerasimenko, aka Comet 67P. The stereo anaglyph was created by combining two images from the Rosetta spacecraft's narrow angle OSIRIS camera taken on September 22, 2014. Stark and jagged, the 3D landscape is found along the Seth region of the comet's double-lobed nucleus. It spans about 985 x 820 meters, pocked by circular ridges, depressions, and flattened areas strewn with boulders and debris. The large steep-walled circular pit in the foreground is 180 meters in diameter. Rosetta's mission to the comet ended in September 2016 when the spacecraft was commanded to a controlled impact with the comet's surface.",
  "hdurl": "http://apod.nasa.gov/apod/image/1703/AP00-67P-Seth.jpg"
}
```

The raw view is just a big text area with the response body. It can help to tell whether your response is minified or not.

## Preview

The screenshot shows the Postman interface with the 'Preview' tab highlighted by a red box. The response body contains the content of the Wikipedia homepage, including the title 'WIKIPEDIA', the tagline 'The Free Encyclopedia', and language links for English, Español, Deutsch, and 日本語.

The preview tab renders the response in a sandboxed iframe. Some web frameworks by default return HTML errors and the preview mode is especially helpful there. Due to iframe sandbox restrictions, JavaScript and images are disabled in the iframe.

You can maximize the body to occupy the whole Postman window. In case you plan on spending a lot of time with the response, this is the way to go.

If your API endpoint returns an image, Postman will detect and render it automatically. For binary response types, you should select “Send and download” which will let you save the response to your hard disk. You can then view it using the appropriate viewer. This gives you the flexibility to test audio files, PDFs, zip files, or anything that the API throws at you.

## Headers

The screenshot shows the Postman interface with the 'Headers' tab highlighted by a red box. The response headers listed are:

- Age → 0
- Connection → keep-alive
- Content-Length → 949
- Content-Type → application/json

Headers are displayed as key-value pairs under the **Headers** tab. Hovering over the header name can give you a description of the header according to the HTTP spec. If you are sending a HEAD request, Postman will show the headers tab by default.

## Response time

Postman automatically calculates the time it took for the response to arrive from the server. This is useful for some preliminary testing for performance.

## Response size

Postman breaks down the response size into body and headers. The response sizes are approximate.

## Cookies

Cookies sent by the server are visible in a dedicated tab. To [manage cookies](#) in Postman the native apps, use the **MANAGE COOKIES** modal. If you're working in the Postman Chrome app, you can use the [Interceptor extension](#) to help manage cookies.

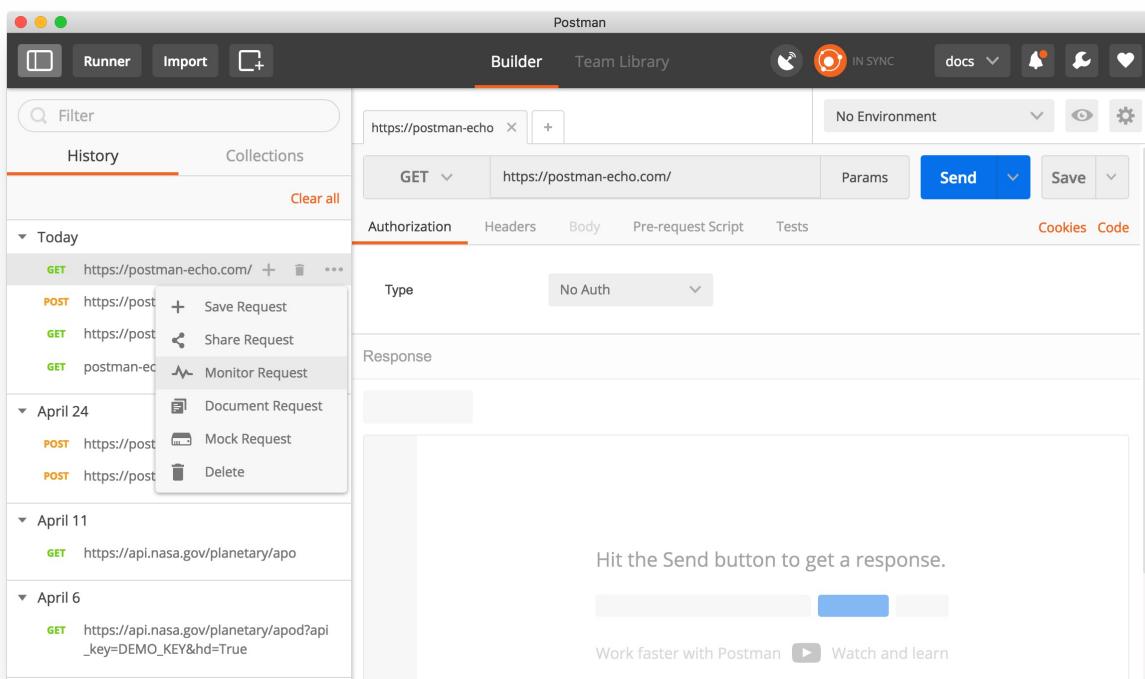
## Tests

Along with everything that you get from the server for the request, you can also see the results of the tests that were run against the request. Learn more about [testing](#) in Postman.

# History

All requests you send using Postman are stored in the history which you can access using the left sidebar. The history lets you experiment with variations of requests quickly without wasting time building a request from scratch. You can load a previous request by clicking on the request name.

If you create an account and sign in to Postman, your history will be synced with our server, backed up in realtime, and retrievable across your devices. If you sign out of your Postman account, and then log back in, the last 10 requests will remain in your history. Postman Pro and Enterprise users will have access to the last 100 requests. The same policy holds for collection runs. Remember that you can always save as many requests in [collections](#) as you want.



From the **History** tab within the sidebar:

## Navigate through requests

Click on a request in this tab to load the exact request configuration in the builder. You can use the up and down arrow keys on your keyboard to navigate through the requests. Postman will load the request in a preview state. Pressing **Enter** will ensure that the request is not replaced by another one.

## Find requests

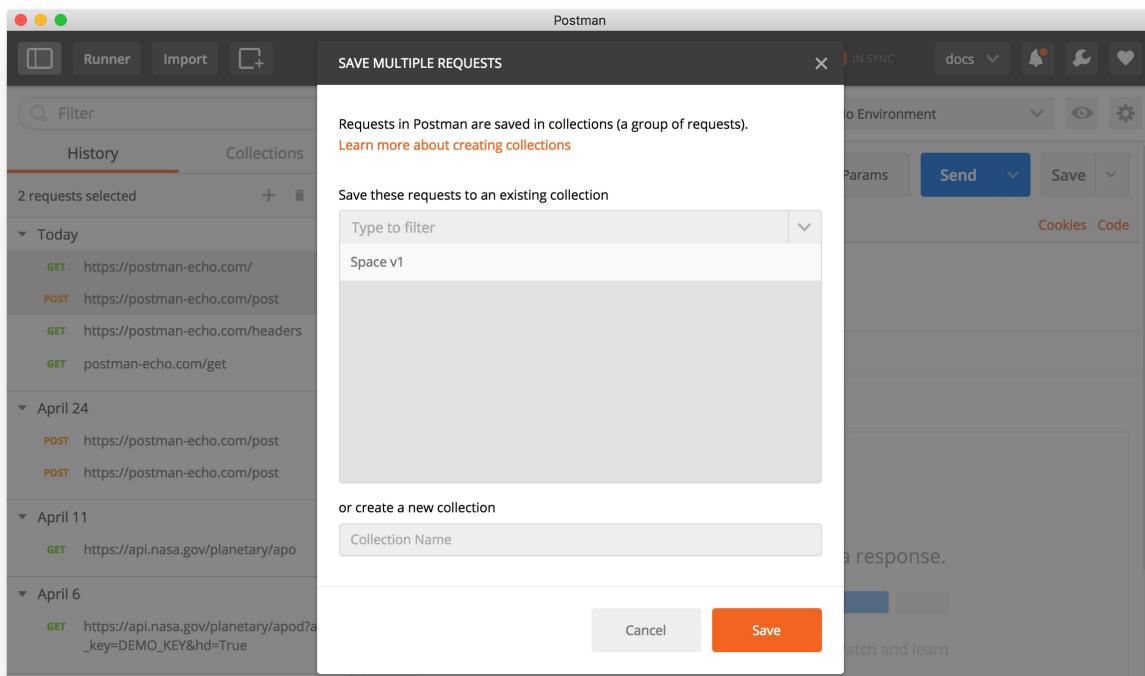
The requests are ordered by time, with the most recent requests displayed at the top. Postman avoids duplicating GET requests with the same URL in the history. If your request history becomes large, filter requests in the sidebar using the search input field. Postman matches the request URL in your history with the search term.

## Multi-selecting requests

Hold down the **CMD** key (or **CTRL** on Windows), and click on each request that you want to select. You can initiate actions like [saving](#), [sharing](#), [documenting](#), [mocking](#), [monitoring](#), or deleting on these requests through actions at the top of the list.

## Save requests to a collection

To organize commonly used requests, you can save them from your history to a collection. To save a single request to a collection, hover over it and click the plus icon (+) that displays next to it. When selecting multiple requests, the plus icon (+) will display at the top of the sidebar. Click on the plus icon (+). Select an existing collection, or create a new collection, to save the requests to a collection.



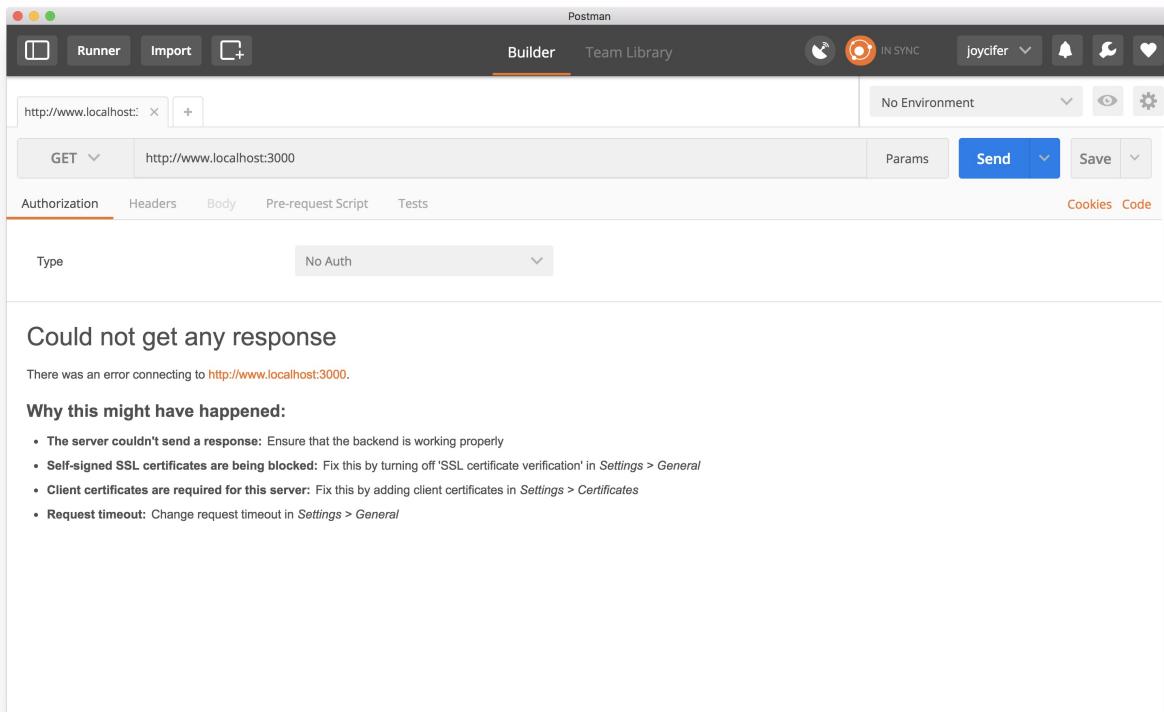
## Delete requests

If you want to get rid of all the requests in the history, select **Clear all** link at the top of the sidebar. To delete a single request, hover over it and click the trash icon that displays next to it. You can also multi-select and delete requests that you don't want to see in your history by

clicking the trash icon that will display at the top of the sidebar once the requests are selected.

# Troubleshooting API requests

There might be cases when your API doesn't work, or exhibits unexpected behavior. If you're not getting any response, Postman will display a message that there was an error connecting to the server.



For more details about the possible causes of the error, open [Postman Console](#), which has detailed information about the failure, which can substantially reduce the time required to troubleshoot. Consider the following issues as you're troubleshooting API requests.

## Connectivity issues

If Postman is unable to connect to your server, it shows the message above. Usually, the easiest way to check if there are connectivity issues is to open your server address in a browser, such as Chrome or Firefox. If opening it in the browser works, then the possible causes could be:

### Firewall issues

Some firewalls may be configured to block non-browser connections, in this case, you should talk to your network administrators in order for Postman to work.

### Proxy Configuration

If you are using a proxy server to make requests, make sure you configure it correctly. By default, Postman uses the proxy settings configured in your Operating System's network settings. Postman Console will provide debug information about the proxy server.

## SSL Certificate issues

When using HTTPS connections, Postman may show the error above. In this case, you can try turning off SSL verification in the Postman Settings. If that does not help, your server might be using a client-side SSL connection. This too can be configured in [Postman Settings](#). Use the Postman Console to ensure that the correct SSL certificate is being sent to the server.

## Client Certificate issues

Client certificates may be required for this server. Fix this by [adding a client certificate](#) in the [Postman Settings](#).

## Incorrect Request URLs

If you use variables in your request, make sure they are defined in your environment or globals. Unresolved request variables may result in invalid server addresses.

## Using incorrect protocol

Check whether you're accidentally using "https://" instead of "http://" in your URL (or vice versa).

## Invalid Postman behavior

Very rarely, it is possible that Postman might be making invalid requests to your API server. You can confirm this by checking your server logs (if available). We're always watching out for these cases, so get in touch with us if you believe Postman is misbehaving. Let us know on our [GitHub issue tracker](#) if you feel that Postman is not working as intended.

## Very short timeouts

If you configure a very short timeout in Postman, the request may timeout before completing, resulting in the error block above. Try increasing the timeout to avoid this issue.

## Invalid Responses

If your server sends incorrect response encoding errors, or invalid headers, Postman will fail to interpret the response, causing the error above.

If you still can't get your API working, help can frequently be found in the [Postman community](#) or [Stack Overflow](#).

If you've tried unsuccessfully troubleshooting the issue, search the [Postman issue tracker](#) on GitHub to check if someone has already reported the issue and whether there is a known solution that you can use. If you're reporting a new issue, follow these [guidelines](#). If you wish to include confidential data, you can file a ticket via our [support center](#) and include the app's [console logs](#) in your report to provide some helpful data for troubleshooting.

# Debugging and logs

Postman apps go through extensive testing and beta builds before we ship. That said, there might be cases when the app crashes, or exhibits unexpected behavior. If you've been unable to [troubleshoot](#) the issue on your own, you can file an issue in the [GitHub tracker](#), or visit our [support center](#) if you wish to include confidential data. Including the app's console logs in your report will provide some helpful data for troubleshooting.

The Postman Console is analogous to a browser's developer console, except that it's tuned for API development. If an API or API test is not behaving as you expect, this would be the place where you will go to deep dive while debugging. As long as the console window is open, all your API activities will be logged here to see what's going on under the hood.

The Postman Console logs the following information:

- The actual request that was sent, including all underlying request headers and variable values, etc.
- The exact response sent by the server before it is processed by Postman
- The proxy configuration and certificates used for the request.
- Error logs from test or pre-request scripts
- `console.log()` from inside scripts.

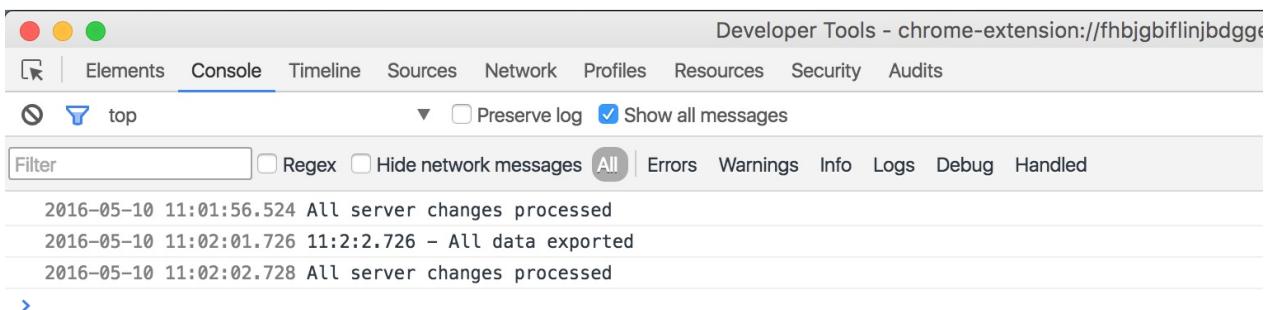
Using `console.info()` or `console.warn()` at appropriate locations in the scripts will help extract the exact line of code that is acting up. If you know your way around `console.log()` in JavaScript, this is similar.

## DevTools Console Logs

To access the console logs, follow these steps:

### For the native app for Mac / Windows / Linux

- Head to View in the application menu, and click on "Show DevTools".
- In the DevTools window, clicking on the top level Console tab should show the app's debug logs.



### For the Chrome app

- Type chrome://flags/#debug-packed-apps in the URL bar in your Chrome browser window.
- Search for “packed” or try to find the “Enable debugging for packed apps” setting.
- Enable the setting.

**NaCl Socket API.** Mac, Windows, Linux, Chrome OS

Allows applications to use NaCl Socket API. Use only to test NaCl plugins. [#allow-nacl-socket-api](#)

[Enable](#)

**Enable debugging for packed apps.** Mac, Windows, Linux, Chrome OS

Enables debugging context menu options such as Inspect Element for packed applications. [#debug-packed-apps](#)

[Disable](#)

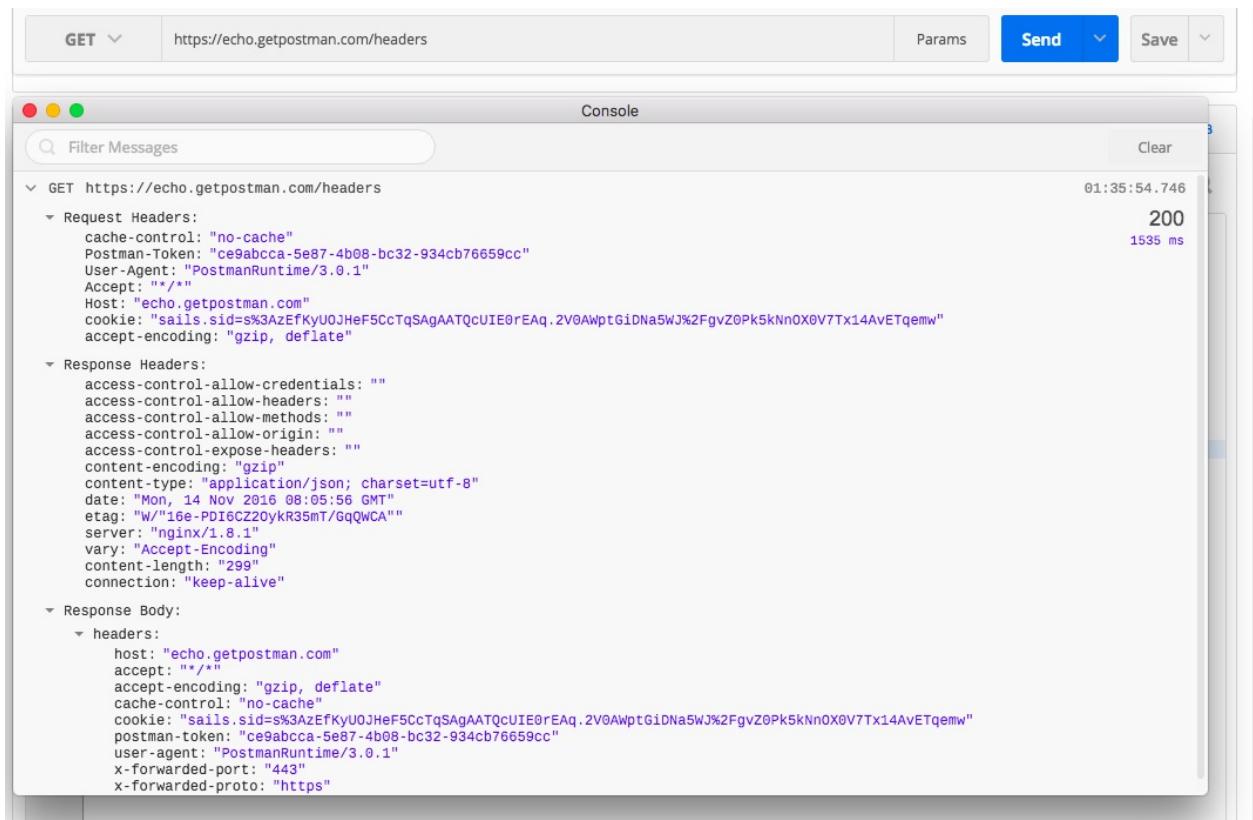
**Enable password generation.** Mac, Windows, Linux, Chrome OS

- Restart Chrome. Once this is done, you can access the Developer Tools window by right clicking anywhere inside Postman and selecting “inspect element”. You can also go to chrome://inspect/#apps and then click “inspect” just below requester.html under the Postman heading.

## Network Calls with Postman Console

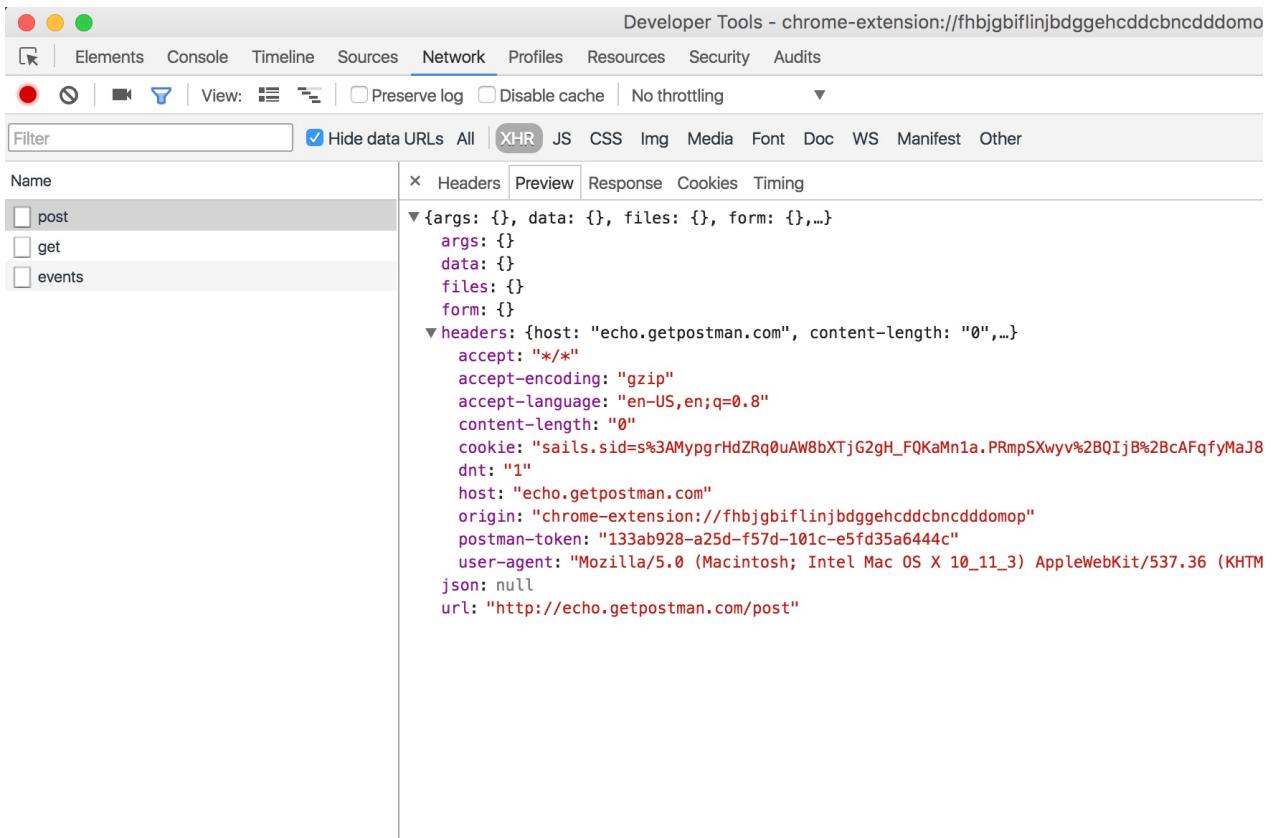
### For the native app for Mac / Windows / Linux

Head to View in the application menu, and click on “Show Postman Console” or use the keyboard shortcut (**CMD/CTRL + ALT + C**). Similar to DevTools, every call along with its headers and payloads will be logged to the Postman Console.



## For the Chrome app

You can also use the DevTools window to inspect the request and response payloads. If the Interceptor is disabled, switch to the Network tab, and you should see each call as it's made. Clicking on this will let you view the headers and payloads for the requests and responses:

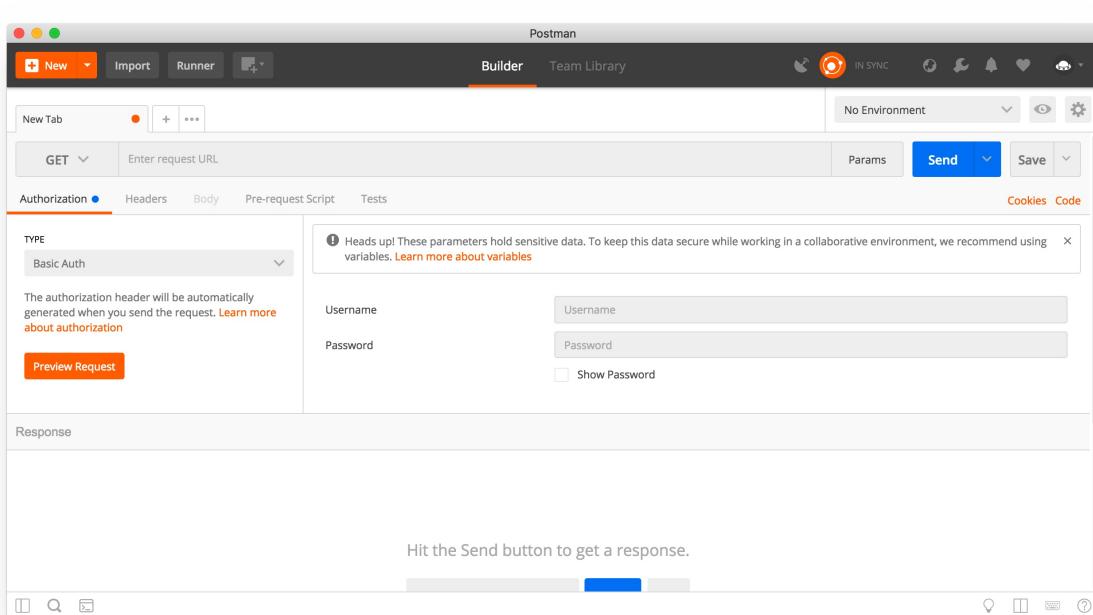




# Authorization

While the request editor is powerful enough to construct any kind of requests, sometimes you might need some help. Postman has “helpers”, which can simplify some repetitive and complex tasks. The current set of helpers let you deal with authentication protocols easily. You can use environment variables with all helpers.

You can choose to save helper data to collection requests. This will cause the signature to be regenerated each time. These helpers will even work in Newman!



**Update:** Starting with Postman 5.3, you will notice a few changes around the request authorization flows. Postman has updated the authorization framework to improve existing authorization types, like OAuth 2.0, and also introduced new authorization types, like NTLM. Additionally, there is no need to manually update the request. If you want to inspect the authorization headers and parameters that Postman generates, you can use the **Preview Request** button. Alternatively, inspect the **Postman console** to get a raw dump of the entire request after it is sent.

## Basic Auth

The screenshot shows the Postman interface with a request configuration. The method is set to GET, the URL is https://postman-echo.com/basic-auth, and the authorization type is set to Basic Auth. The username field contains 'postman' and the password field contains '\*\*\*\*\*'. A note indicates that the authorization header will be generated and added as a custom header. There is also a checkbox for 'Save helper data to request' which is unchecked. Buttons for 'Send', 'Save', 'Clear', and 'Update Request' are visible.

Enter the username and password fields and hit “Update Request” to generate the authorization header.

## Digest Auth

The screenshot shows the Postman interface with a request configuration. The method is set to GET, the URL is https://postman-echo.com/digest-auth, and the authorization type is set to Digest Auth. The username field contains 'postman', the realm field contains '{{echo\_digest\_realm}}', and the password field contains '\*\*\*\*\*'. A note indicates that the authorization header will be generated and added as a custom header. A checkbox for 'Save helper data to request' is checked. Other fields shown include Nonce, Algorithm (set to MD5), qop, Nonce Count, Client Nonce, and Opaque, all of which are currently empty. Buttons for 'Send', 'Save', 'Clear', and 'Update Request' are visible.

Digest auth is more complicated than basic auth and uses the values currently set in the request to generate the authorization header. Make sure they are set properly before you generate the header. Postman will remove the existing header if it's already present.

## OAuth 1.0a

The screenshot shows the Postman interface with a GET request to <https://postman-echo.com/oauth1>. The 'Authorization' tab is selected. The 'Type' dropdown is set to 'OAuth 1.0'. The configuration fields include:

- Consumer Key:** RKCGzna7bv9YD57c
- Consumer Secret:** CaeyGPr2mns1WCq4Cpm5aLvz6
- Token:** WCq4Cpm5ALvz6GsGPr2mns1V
- Token Secret:** feBD3nofINFL3mof9Hf3
- Signature Method:** HMAC-SHA1
- Timestamp:** 1448881347
- Nonce:** IV4Xwg
- Version:** 1.0
- Realm:** Optional

On the right, there are several checkboxes with descriptions:

- Add params to header
- Add empty params to signature
- Encode OAuth signature
- Save helper data to request

A note states: "Postman will auto-generate the timestamp and nonce values if left blank".

Postman's OAuth helper lets you sign requests which support OAuth 1.0a based authentication. Currently, it does not let you acquire the access token. That's something you would need from the API provider. The OAuth 1.0 helper can set values in either the header or as query parameters.

As subsequent OAuth requests might expect a different nonce value, Postman can refresh the OAuth signature just before the request is sent if auto add parameters is enabled.

The OAuth 1.0 spec is quite complicated and there are many variations. Postman tries to support as many of those variations as possible but if something does not work for you, [file an issue on Github](#). These are few of the options that we've included:

### Add params to header

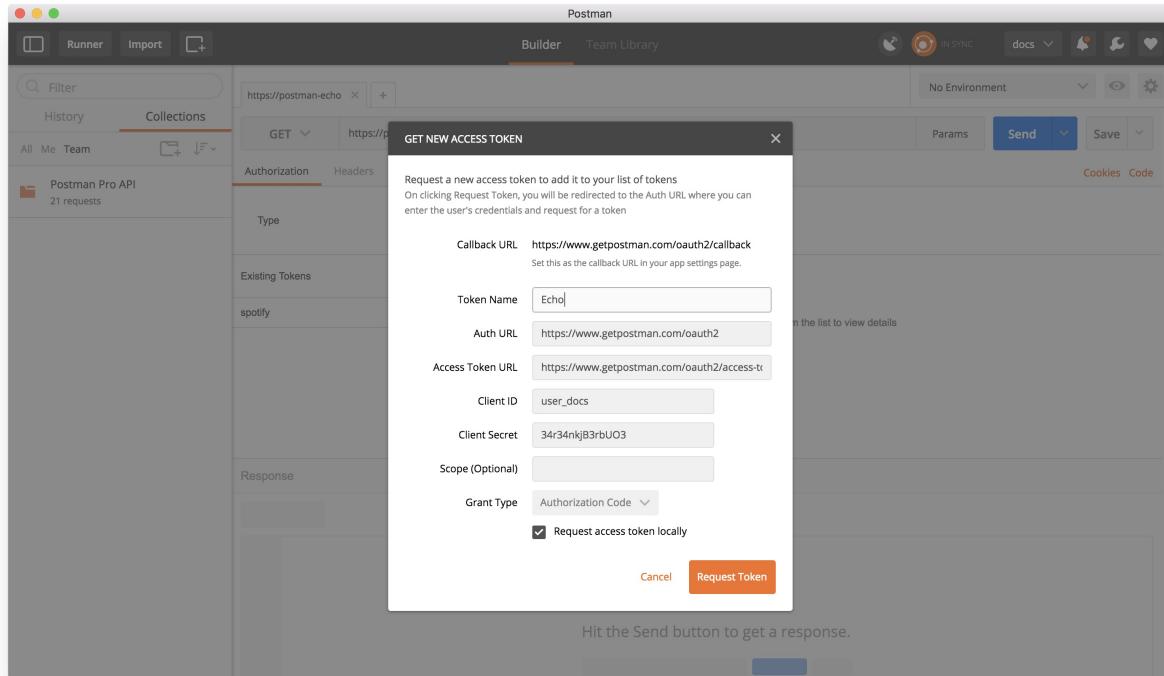
If this checkbox is enabled, params are added to the header. If not, the URL params for a GET request, and the request body for POST/PUT requests.

### Add empty params to signature

Some implementations of OAuth1.0 require empty parameters to be added to the signature.

## OAuth 2.0

Postman supports getting the OAuth 2.0 token as well as adding it to requests really easily. To get an access token from an OAuth 2.0 provider, follow these steps:



- Click the “Get New Access Token” button to open a modal. You will see <https://www.getpostman.com/oauth2/callback> as the Callback URL.
- From your API provider, get the values for Authorization URL, Access Token URL, Client ID and Client Secret. These values will be provided by your API provider. Optionally, you can set the Scope parameter which is needed by some APIs to set the level of access you have within the API.
- Press the “Request Token” button to initiate the OAuth 2.0 flow. If everything is set up properly, you would be redirected to the Postman server which will pick up your access token and send it to the Postman app. To finish adding the token to Postman, give it a name so that you can access it quickly later.
- If your OAuth2 provider is not publicly accessible (hosted locally or on your intranet), make sure the ‘Request Access Token Locally’ option is enabled.
- Access tokens are stored locally and will show up in the helper list. To add an access token to the request, click the token name.

## Hawk authentication

Hawk is an HTTP authentication scheme using a message authentication code (MAC) algorithm to provide partial HTTP request cryptographic verification.

Read more on the [Hawk Github page](#).

## AWS authentication

AWS users have to use a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. Postman supports this out of the box.

Read more about the AWS Signature on AWS documentation:

- <http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html>
- <http://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-use-postman-to-call-api.html>

# Cookies

Postman's native apps provide a **MANAGE COOKIES** modal that lets you edit cookies that are associated with each domain.

## Getting to the cookie manager

To open the **MANAGE COOKIES** modal, click the **Cookies** link under the **Send** button.

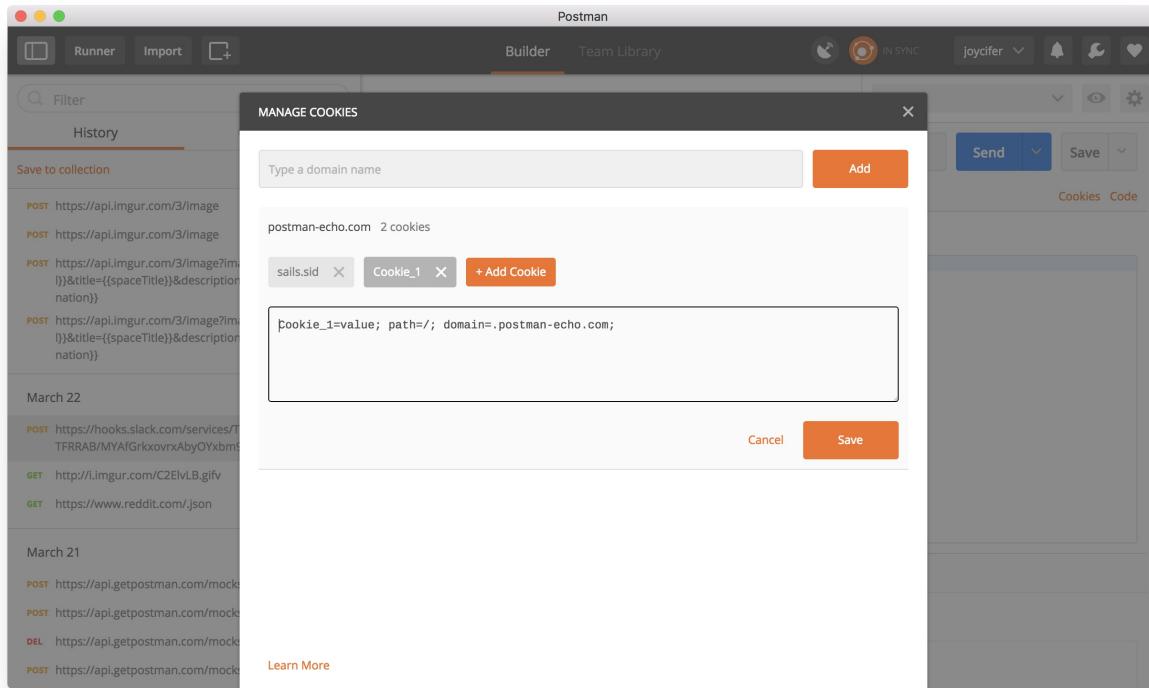
The screenshot shows the Postman interface with a request to 'postman-echo.com/get'. The 'Cookies' tab is highlighted in the bottom right corner of the main window. The 'Authorization' tab is currently selected.

This will open up the **MANAGE COOKIES** modal, and display a list of domains and the cookies associated with them.

The screenshot shows the Postman interface with the 'MANAGE COOKIES' modal open. The modal has a search bar at the top and a list of domains below. The 'postman-echo.com' domain is listed with the note '1 cookie'. A single cookie named 'sails.sid' is shown with a delete button. At the bottom of the modal, there is a red box around the '+ Add Cookie' button.

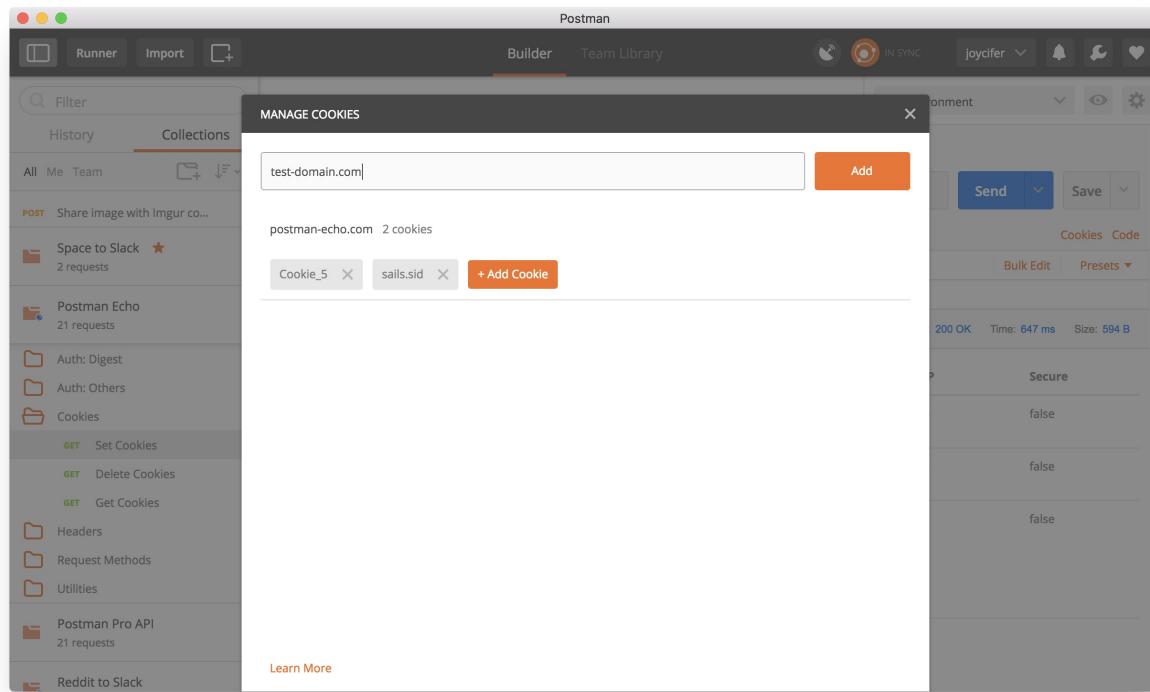
## Create a cookie

To add a new cookie for the domain, click on the **Add Cookie** button. A pre-generated cookie string according to the [HTTP State Management standards](#) will be created, but you can edit it using the text input that appears below it. Clicking the **Save** button will save it to the app's cookie store under the relevant domain.



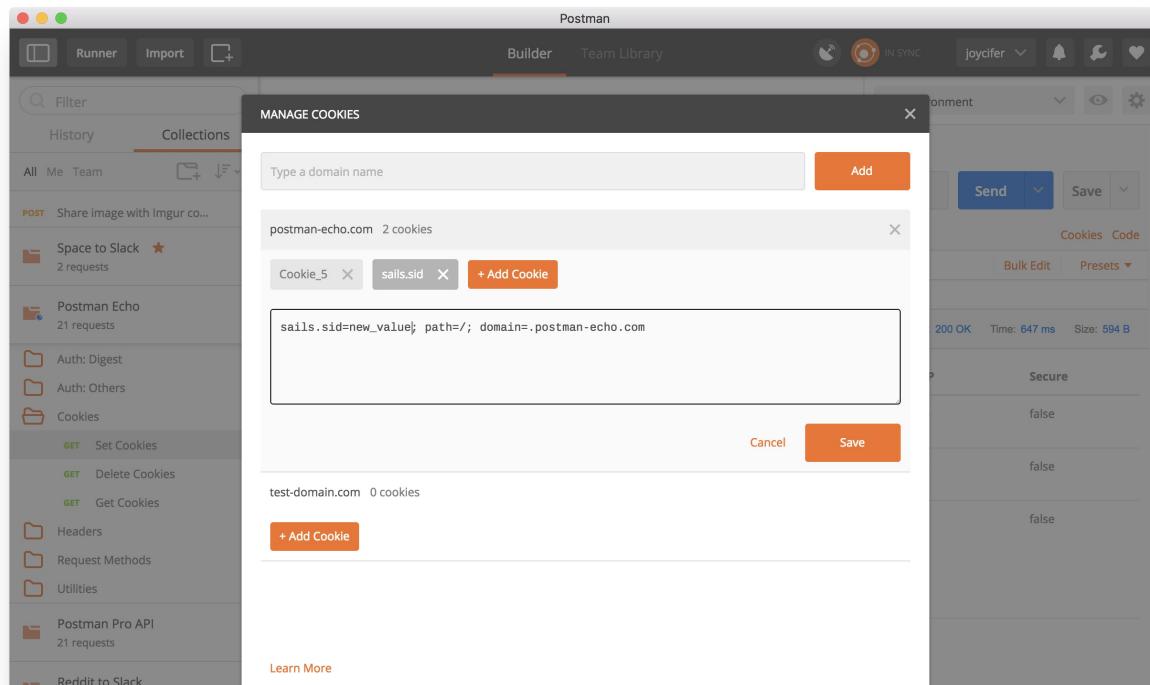
## Adding a domain

If you want to add a cookie for a domain that isn't present in the domain list, you can add one by entering the hostname (without the port or the http://) in the input box at the top. Clicking the **Add** button will add it to the domain list. You can then add cookies for this domain by selecting it, and entering a new cookie value as described above.



## Updating a cookie

To update an existing cookie, go to the domain from the domain list, and click the cookie you want to edit. You can edit any property, and hit **Save** to update.



## Adding Cookies through Set-Cookie header

You can also add/edit the cookies through the [Set-Cookie header](#) through the response.

## Coming soon

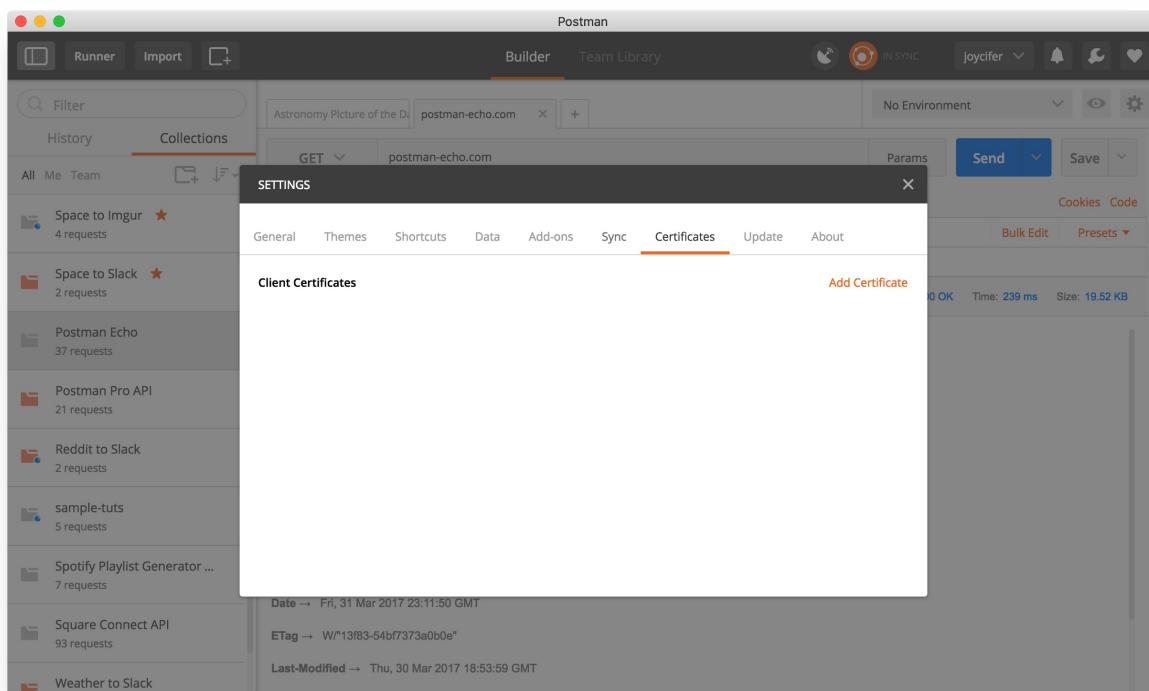
These are two properties which are not yet supported Postman.

- SameSite
- Cookie Prefixes
  - \_\_Secure-
  - \_\_Host-

# Certificates

Postman's native apps provide a way to view and set SSL certificates on a per domain basis.

To manage your client certificates, click the wrench icon on the right side of the header toolbar, choose “Settings”, and select the **Certificates** tab.



## Adding a Client Certificate

To add a new client certificate, click the **Add Certificate** link.

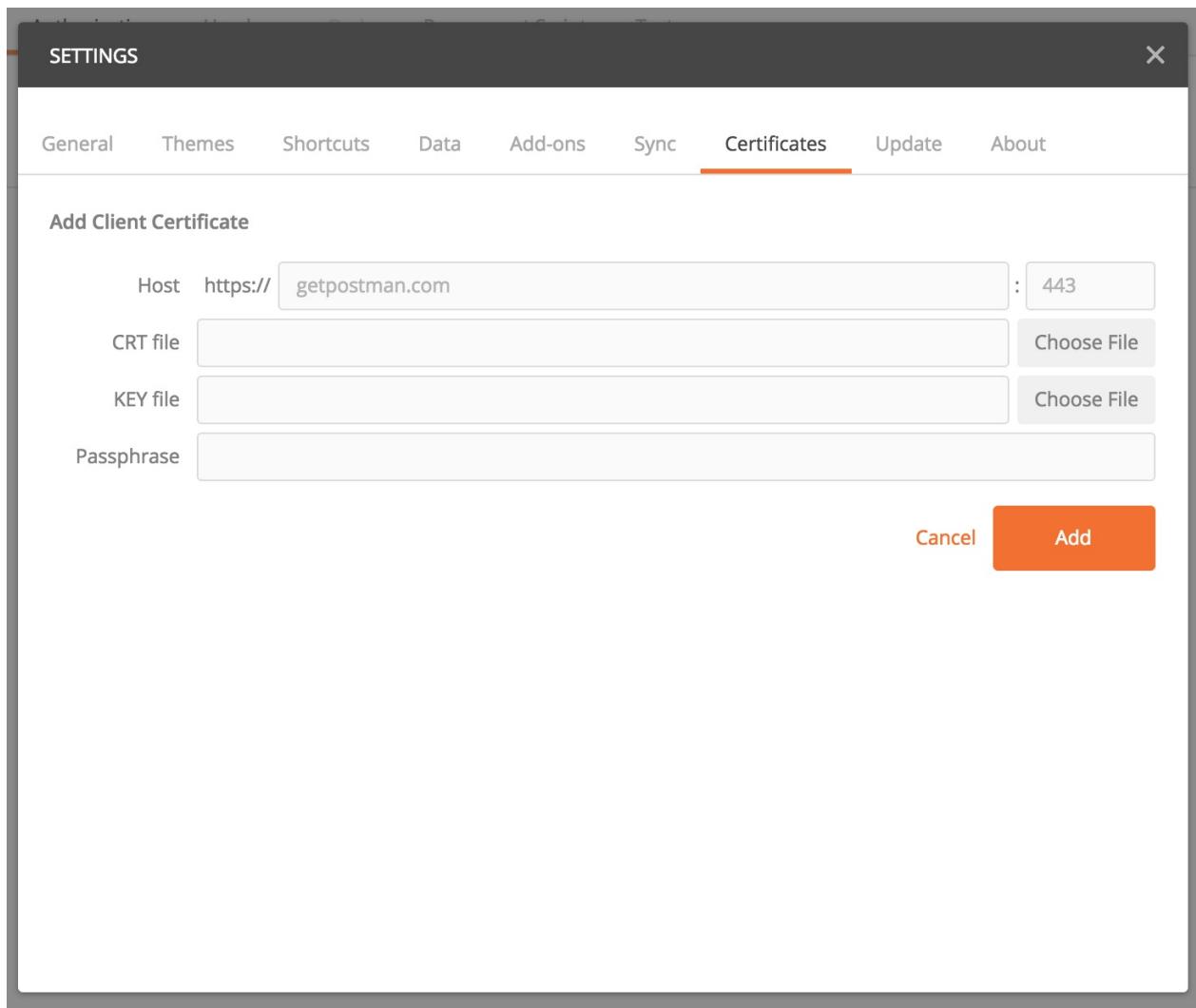
In the **Host** field, enter the domain (without protocol) of the request URL for which you want to use the certificate, for example, [echo.getpostman.com](https://echo.getpostman.com).

You can also specify a custom port to associate with this domain in the **Port** field. This is optional. If left empty, the default HTTPS port (443) will be used.

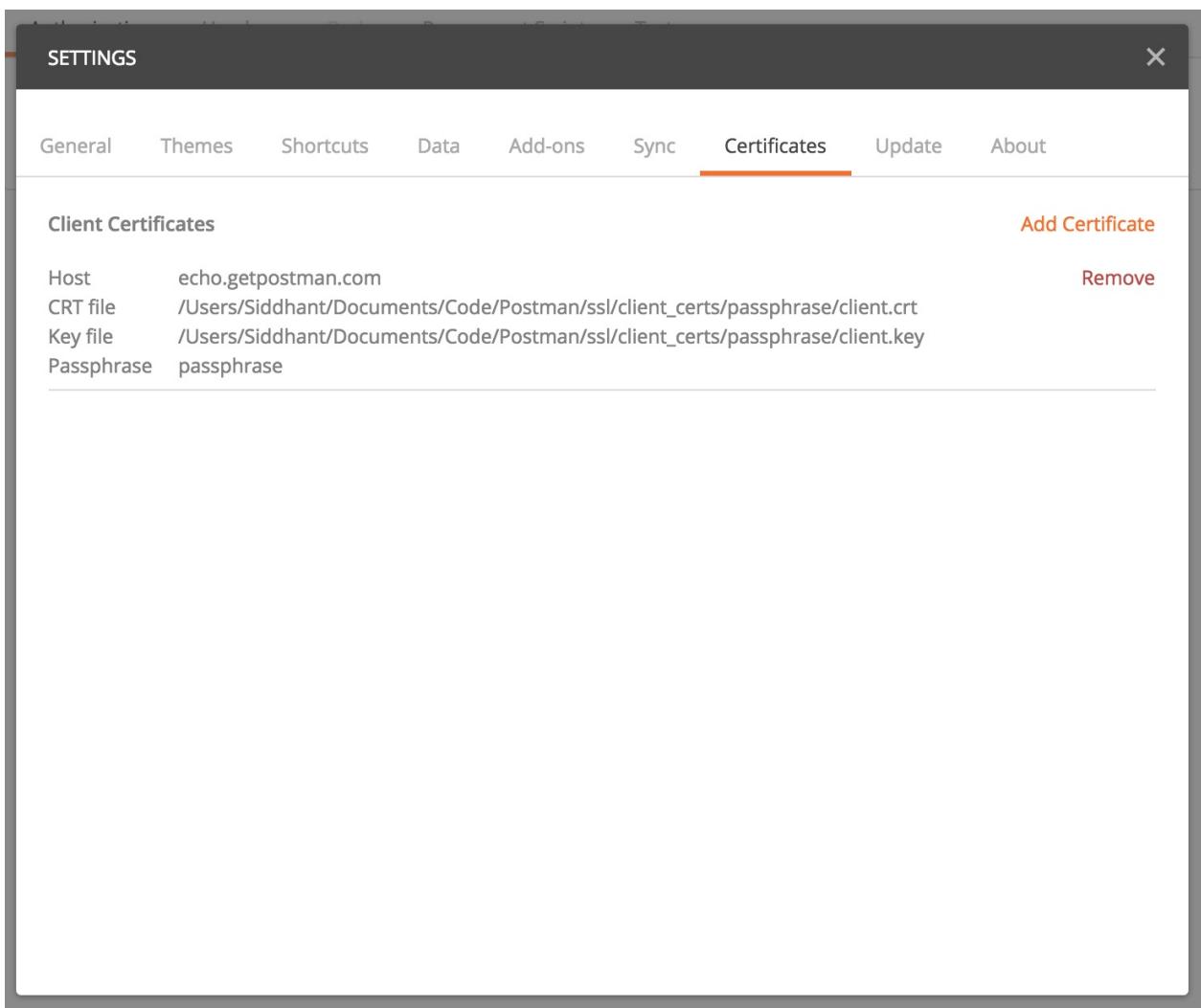
Choose your client certificate file in the **CRT file** field. Currently, we only support the CRT format. Support for other formats (like PFX) will come soon.

Choose your client certificate key file in the **KEY file** field.

If you used a passphrase while generating the client certificate, you'll need to supply the passphrase in the **Passphrase** field. Otherwise, leave it blank.



Once your certificate is added, it should appear in the client certificates list.



**NOTE:** You should not have multiple certificates set for the same domain. If you have multiple ones set, only the last one added will be used.

## Using a Certificate

You do not have to perform any extra steps to use a client certificate if it has been added. If you make a request to a configured domain, the certificate will automatically be sent with the request, provided you make the request over HTTPS.

You can verify this. To do so, open up your Postman console (**CMD/CTRL + ALT + C**). You can read more about the [Postman Console](#). A new window will open up.

Now, send a request to <https://echo.getpostman.com/get>, keeping the Postman Console open. Notice we're using https to make sure the certificate is sent. Once the response arrives, switch over to the Postman console to see your request. If you expand your request, you will be able to see which certificate was sent along with the request.

```

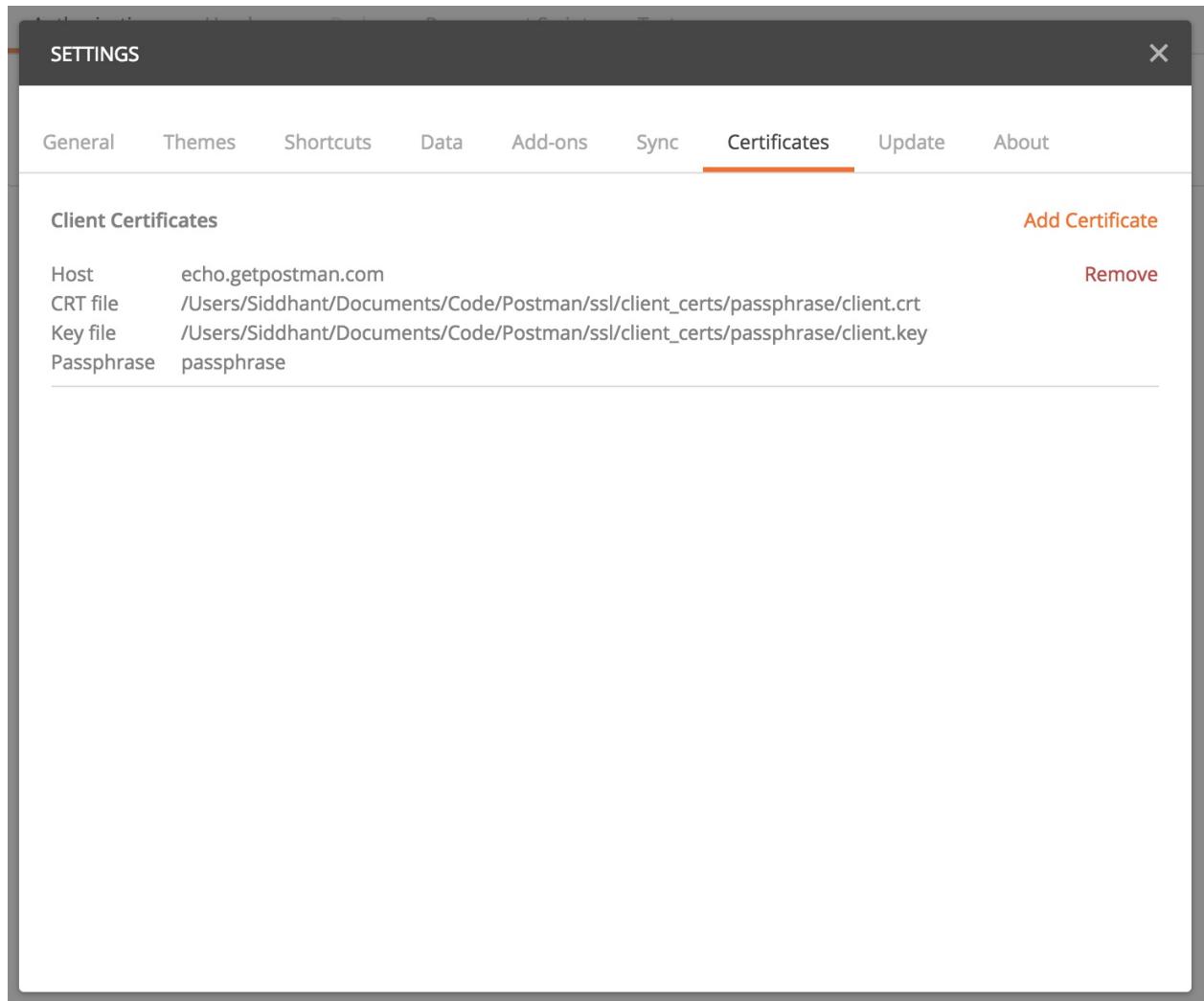
Console
Filter Messages
Clear
03:29:31.007
200
289 ms

GET https://echo.getpostman.com/get
Client Certificate:
  pemPath: "/Users/Siddhant/Documents/Code/Postman/ssl/client_certs/passphrase/client.crt"
  keyPath: "/Users/Siddhant/Documents/Code/Postman/ssl/client_certs/passphrase/client.key"
Request Headers:
  cache-control: "no-cache"
  Postman-Token: "4290c93f-e53a-4827-b631-caec2e9a9083"
  User-Agent: "PostmanRuntime/3.0.1"
  Accept: "*/*"
  Host: "echo.getpostman.com"
  cookie: "sails.sid=s%3AEL_SDxA5-ctsJEyUo-UJQdfQDZ8-LQA-.DbvKctvA0yVJrn30UB%2B1c%2B%2FikXLAXiCeB5WLdoAuLas"
  accept-encoding: "gzip, deflate"
Response Headers:
  access-control-allow-credentials: ""
  access-control-allow-headers: ""
  access-control-allow-methods: ""
  access-control-allow-origin: ""
  access-control-expose-headers: ""
  content-encoding: "gzip"
  content-type: "application/json; charset=utf-8"
  date: "Wed, 26 Oct 2016 09:59:31 GMT"
  etag: "W/\"1a4-xvgsZA0buPIzvhLdg26E9g\""
  server: "nginx/1.8.1"
  vary: "Accept-Encoding"
  content-length: "320"

```

## Removing a Certificate

To remove a certificate, use the **Remove** link next to the certificate under the **Certificates** tab in the Settings.



## Editing a Certificate

You cannot edit a certificate after it has been created. To make changes to it, you will need to remove the certificate and create a new one.

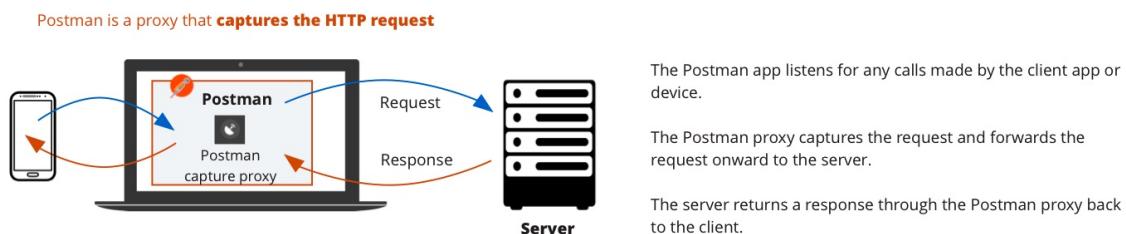
# Capturing HTTP requests

If you are using APIs to build client-side applications - mobile apps, websites or desktop applications - you might want to see the actual HTTP request traffic that is being sent and received in the application. In some cases, you might discover APIs that are not even documented. Postman gives you tools to see and capture this network traffic easily. You can use the built-in proxy in the Postman native apps or use the [Interceptor extension](#) for the Postman Chrome app.

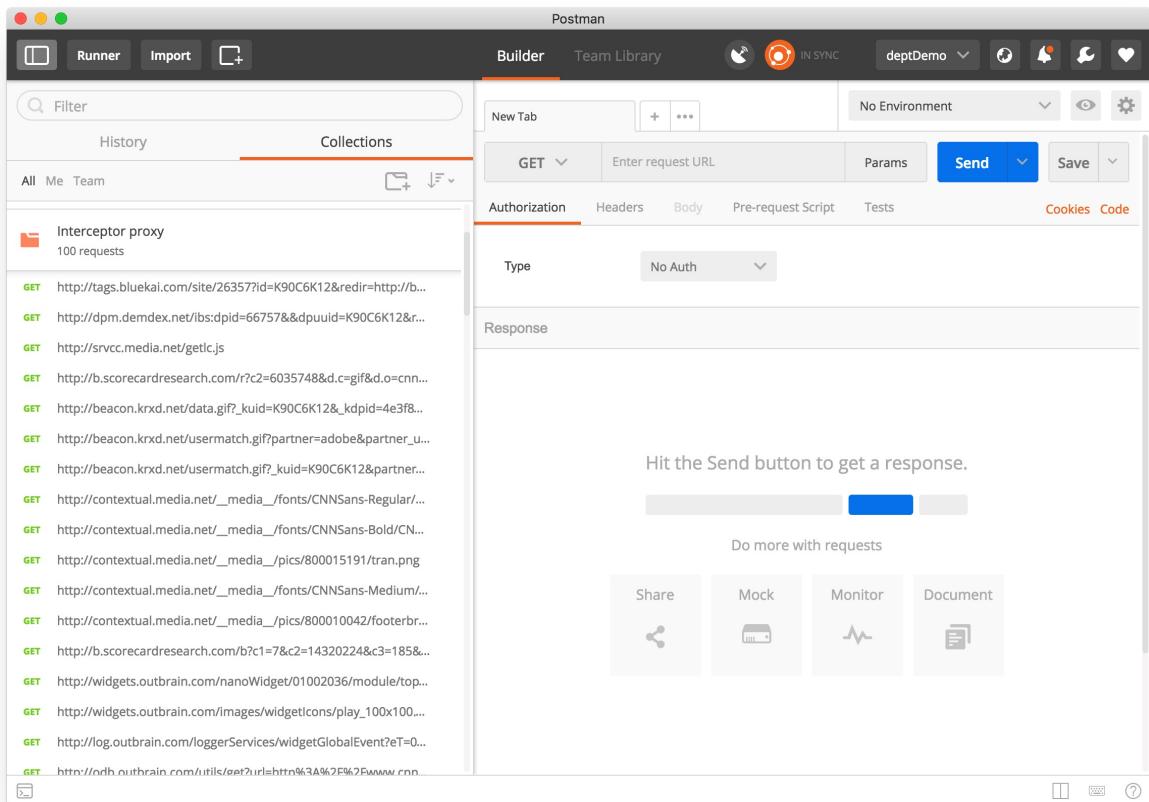
## The Postman built-in proxy

Postman has a proxy in the Postman app that captures the HTTP request.

1. The Postman app listens for any calls made by the client app or device.
2. The Postman proxy captures the request and forwards the request onward to the server.
3. The server returns a response through the Postman proxy back to the client.



Similar to the [Interceptor Chrome extension](#), the Postman app proxy also INTERCEPTS and captures your requests. In this scenario, the Postman app is the proxy, and you can inspect HTTP communication going out from your phone like in the following example, and log all network requests under the History tab of the sidebar.



## Using Postman's proxy example

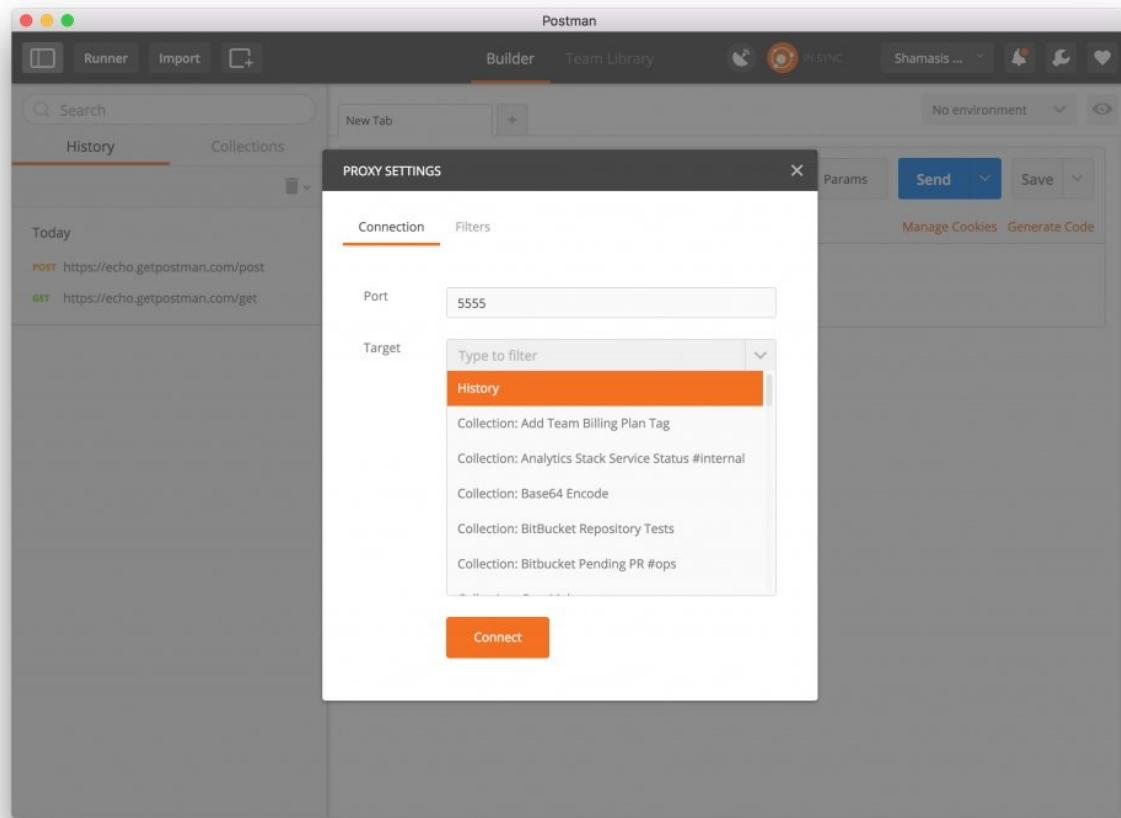
In this tutorial, we will use Postman's proxy feature to inspect HTTP communication going out from your phone. To get started, make sure your computer and mobile are connected to the same local wireless network.

### Step 1: Set up the proxy in Postman

Open the **PROXY SETTINGS** modal in the Postman app (MacOS) by clicking the icon in the header toolbar.

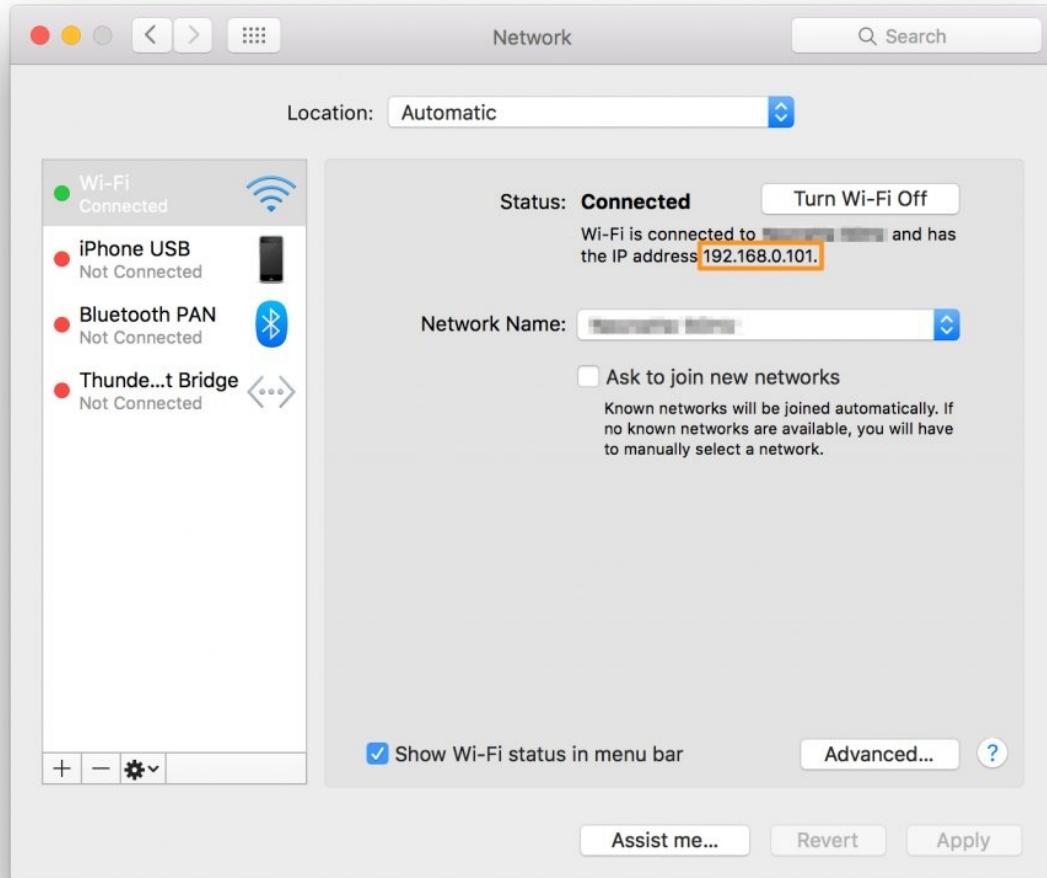


Keep a note of the port mentioned in the proxy settings. In this case, let's keep it at the default port 5555. Set the target to "History". This will cause all your requests to be captured and stored in the History sidebar panel.



### Step 2: Note your computer's IP address

On OS X, the computer's IP address can be found in *System Preferences > Network*. The IP address of your system will be something like the example here 192.168.0.101.



### Step 3: Configure HTTP proxy on your mobile device

Open the wireless settings of your mobile device and update the configuration of the wireless connection to use HTTP Proxy. Set the IP address with the IP you retrieved from your computer in the second step. Set the port with the port you established in Postman in **Step 1**.



IP Address 192.168.0.105

Subnet Mask 255.255.255.0

Router 192.168.0.1

DNS 192.168.0.1

Search Domains

Client ID

[Renew Lease](#)

HTTP PROXY

Off

Manual

Auto

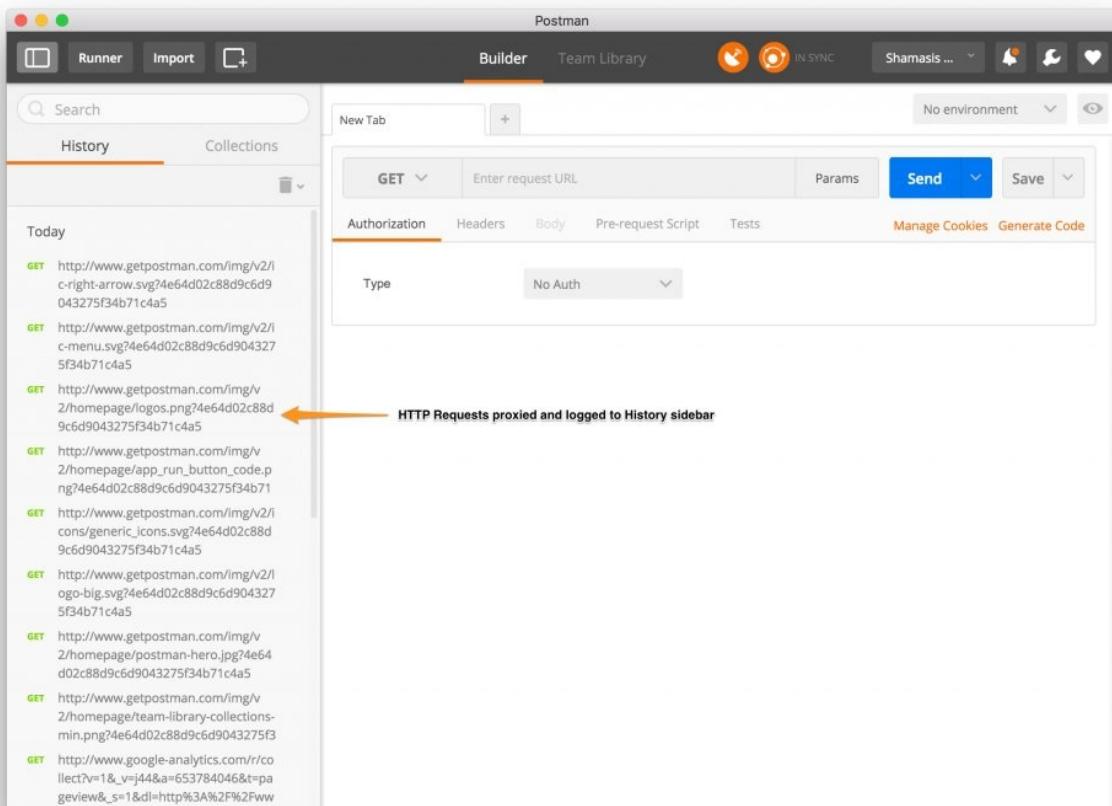
Server 192.168.0.101

Port 5555

Authentication

Set the proxy IP address of your device (an iPhone in this example) to the IP address you obtained from your system and port 5555.

You are all set! Head over to the Postman app, and you will start seeing the network calls listed under the **History** tab of the sidebar. Open your device's web browser or your application and you will start seeing HTTP traffic passing through the app or the browser.



## Connect to proxy for target devices

The broader development community has published some useful tutorials for setting up a proxy server on various operating systems.

- [Windows](#)
- [Linux](#)
- [macOS](#)
- [Android](#)

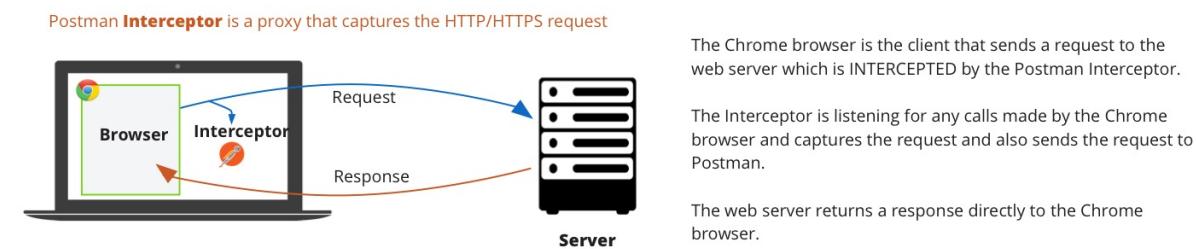
# Interceptor extension

## What is Interceptor

Postman Interceptor is a Chrome extension that functions as a proxy to capture HTTP or HTTPS requests. It can capture network requests directly from Chrome and save them to Postman's history. This means you can debug your web apps APIs in real time!

In this example:

1. The Chrome browser is the client that sends a request to the web server which is INTERCEPTED by the Postman Interceptor.
2. The Interceptor is listening for any calls made by the Chrome browser and captures the request, forwards the request onward, and also sends the request to Postman.
3. The web server returns a response directly to the Chrome browser.



There is no need to install or configure a proxy. There are no code changes required either. You can filter requests according to the URL based on a regular expression. If you have a web app for which you don't have a collection built already, or you just want to debug the APIs that your app is using, this can save a lot of time. The Postman Chrome app can be used in tandem with the Postman Interceptor extension to make and capture requests. It can also capture and manipulate cookies or set certain HTTP headers that are blocked on the Chrome platform by default.

The screenshot shows a web browser window with the Postman Interceptor extension installed. The extension's sidebar is open, providing information about its features and settings.

**Postman Interceptor**  
Helps you send requests which use browser cookies through the [Postman Chrome app](#)

**Request Capture**  
Captured requests will show up inside Postman's history

**ON**

**Filter requests**  
\*

**Apply filter**

**Last 10 requests**

- GET** [http://staticcxx.facebook.com/connect/xd\\_arbiter/r/0sTQzbapM8j.js?version=42&cb=f35d1e22cda239&domain=www.cnn.com&origin=http%3A%2F%2Fwww.cnn.com%2Ff2a](http://staticcxx.facebook.com/connect/xd_arbiter/r/0sTQzbapM8j.js?version=42&cb=f35d1e22cda239&domain=www.cnn.com&origin=http%3A%2F%2Fwww.cnn.com%2Ff2a)
- GET** [https://staticcxx.facebook.com/connect/xd\\_arbiter/r/0sTQzbapM8j.js?version=42&channel=f2ae97e1f34006&origin=http%3A%2F%2Fwww.cnn.com](https://staticcxx.facebook.com/connect/xd_arbiter/r/0sTQzbapM8j.js?version=42&channel=f2ae97e1f34006&origin=http%3A%2F%2Fwww.cnn.com)
- GET** [https://www.facebook.com/connect/ping?client\\_id=80401312489&domain=www.facebook.com](https://www.facebook.com/connect/ping?client_id=80401312489&domain=www.facebook.com)

**Get Forecast**

**Recent Locations**

New York City, NY  
59°  
Make Default

## Installing Interceptor

Here how to get started:

- Install Postman from the Chrome Web Store, if you don't have it already.
- Install the [Interceptor extension](#).
- Open Postman, and click on the Interceptor icon in the toolbar to switch the toggle to “on”.
- Browse your app or your website and monitor the requests as they stream in.

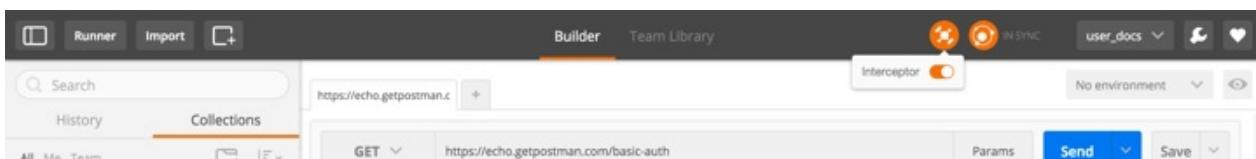
**Note on security:** The only entity that the Interceptor communicates with is Postman which then saves it to your history. We have open-sourced Interceptor and you can find the code on [Github](#). Postman saves all your data locally inside IndexedDB.

## Capturing cookies

Unlike the Postman native apps, the Postman Chrome app is not equipped to handle cookies by itself. You can use the Interceptor extension to overcome this. With the Interceptor on, you can retrieve cookies set on a particular domain and include cookies while sending requests.

## Retrieving cookies

Make sure the Interceptor is enabled in the Postman header toolbar.



Under the **Tests** tab, you can use the “responseCookies” object. This will return an array of cookie objects. To retrieve a particular name, use “`postman.getResponseCookie(cookieName)`”. This will return a single cookie object. Each cookie object will contain the following properties: domain, hostOnly, httpOnly, name, path, secure, session, storeId, value.

## Setting Cookies

- Make sure the Interceptor is enabled.
- Include the “Cookie” header in the headers section (eg. `Cookie: name=value; name2=value2`).
- Send the request. The cookies you set will be sent by Chrome along with your request.

## Restricted Headers

Unfortunately some headers are restricted by Chrome and the XMLHttpRequest specification. The following headers are blocked:

- Accept-Charset
- Accept-Encoding
- Access-Control-Request-Headers
- Access-Control-Request-Method
- Connection
- Content-Length
- Cookie
- Cookie 2
- Content-Transfer-Encoding
- Date
- Expect
- Host
- Keep-Alive
- Origin
- Referer
- TE
- Trailer
- Transfer-Encoding
- Upgrade
- User-Agent
- Via

However sending these restricted headers is easy. Follow the steps below:



- Install the Interceptor extension either by clicking on the Interceptor icon in the Postman toolbar or through the [Chrome Web Store](#).
- Once it's installed, click on the icon again in the Postman app and toggle it on.

That's it! You can now send requests which use these headers.



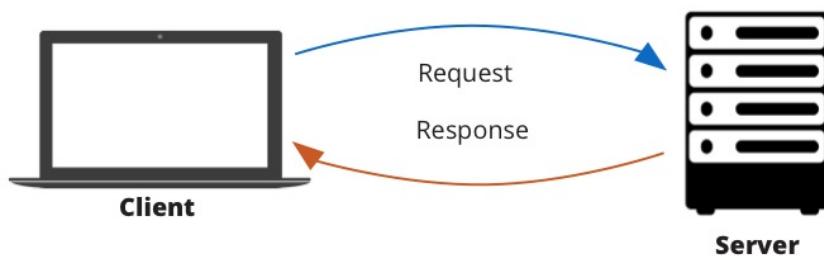
# Proxy

A proxy server acts as a security barrier between your internal network and the Internet, keeping others on the Internet from accessing information on your internal network.

## What is a proxy?

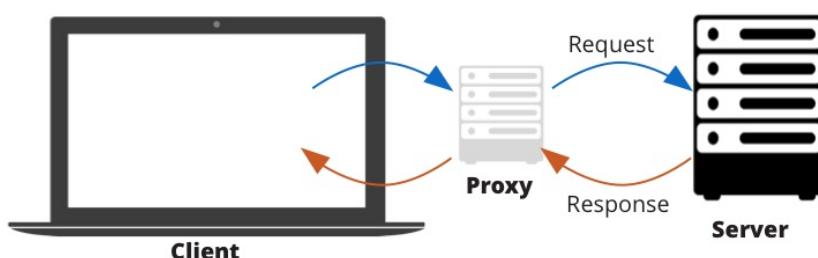
In basic web speak, a client makes a request to the server, and the server sends back a response.

Standard request and response



A proxy server is an application or system that acts as an intermediary between your computer and the internet, or more specifically, the client and server. The proxy makes requests on your behalf to websites, servers, and other internet services.

Standard web proxy



The proxy can reside on your local machine, somewhere in your network, or at any point between your client and the destination server on the internet.

Similar to the way parents might speak to each other through a child, the child is a proxy relaying all communications between the 2 parents.

**Parent 1:** Ask your father if he can pick you up after school.

**Timmy:** Can you pick me up after school.

**Parent 2:** Yeah.

**Timmy:** Dad says yeah. In this analogy, the child forwards the information on behalf of each parent. Besides just relaying information, [proxies can do much more](#).

- Record all traffic between your machine and the internet
- Reveal the contents of all requests, responses, cookies, and headers
- Route traffic to specified internet locations
- Debugging
- Security from direct attacks
- DevOps load balancing

A proxy acts like a go-between to perform various functions. Postman has a [built-in web proxy to capture API requests](#), the [Postman Interceptor to intercept network traffic](#), and proxy settings to direct API requests.

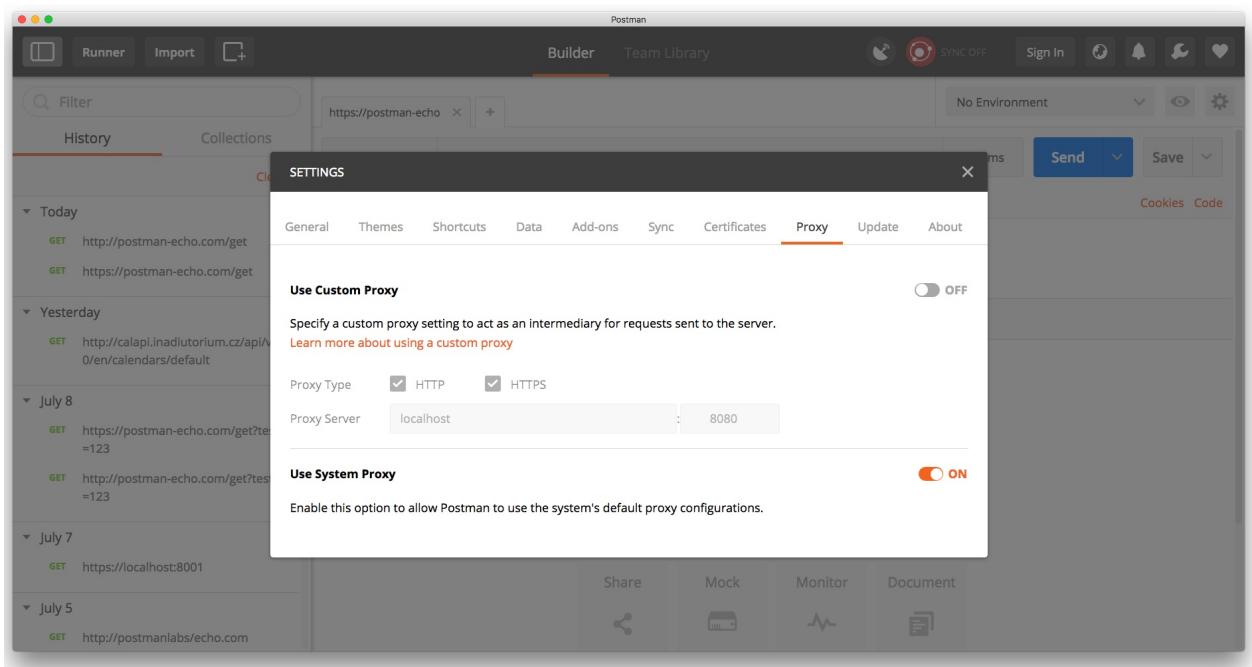
## Configuring proxy settings

This article describes how to configure the proxy settings in Postman to direct all requests made in the Postman app to route through a proxy server. This is different from [capturing network traffic](#) with the built-in proxy which allows Postman to intercept network traffic.

Postman's native apps for Mac, Windows, and Linux support configuring proxies. You can either specify to use a **custom proxy** or to use the **system proxy** defined in the operating system.

Use the **system proxy** if all of your applications need to use the same proxy. Use the **custom proxy** if you want to direct the requests from Postman go through a custom proxy server.

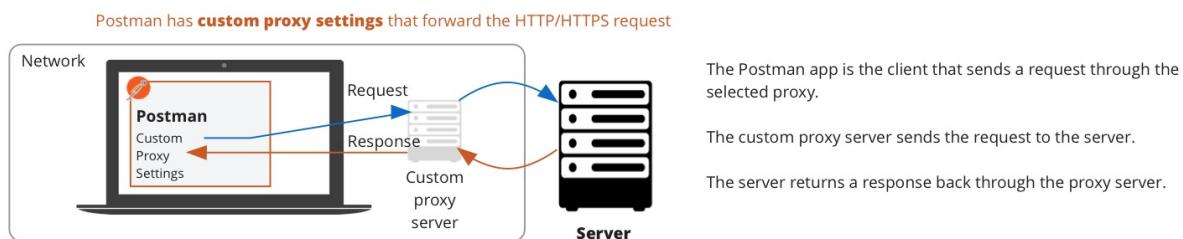
To configure the proxy settings, click the wrench icon on the right side of the header toolbar, choose “Settings”, and select the **Proxy** tab.



## Using custom proxy

Postman allows you to configure **custom proxy settings** that direct Postman to forward your HTTP or HTTPS requests through a proxy server. In other words, this will route all requests sent via the Postman app through a proxy server of your choosing.

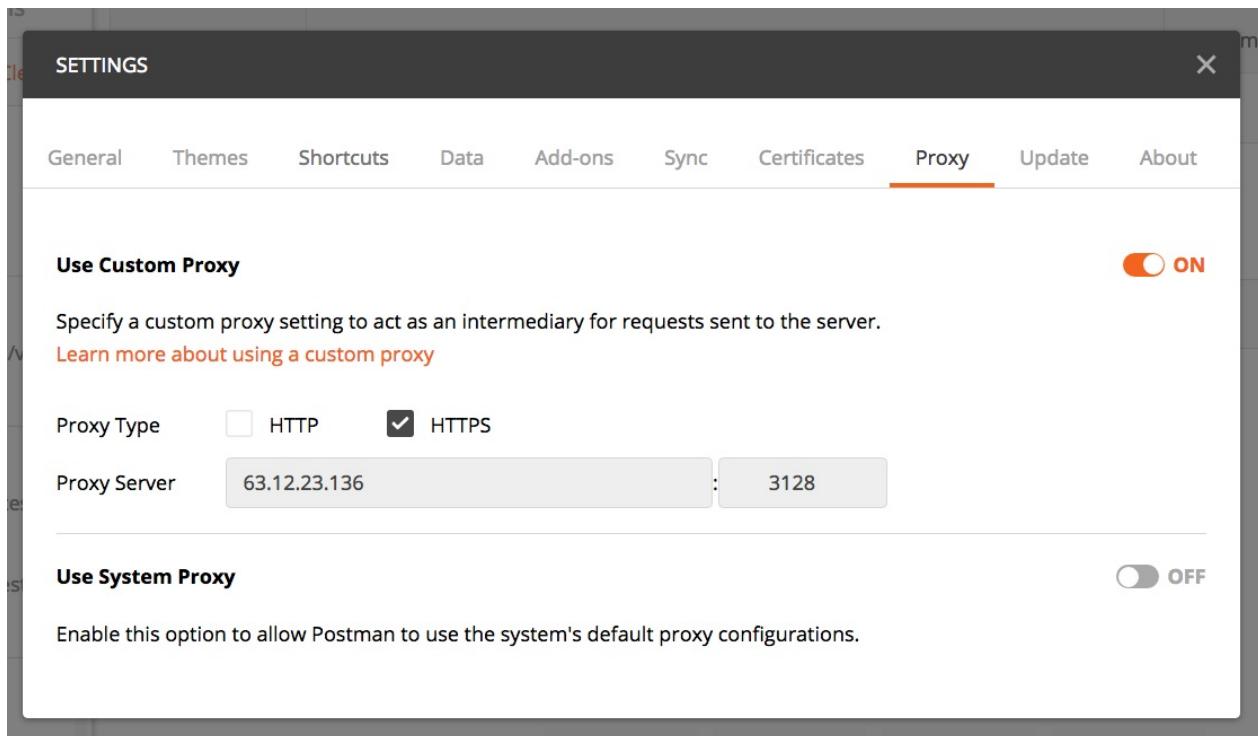
1. The Postman app is the client that sends a request through the selected proxy.
2. The proxy server sends the request to the server.
3. The server returns a response back through the proxy server.



Custom proxy settings are disabled by default and can be turned on using the toggle switch.

Choose the type of proxy server by checking the appropriate checkboxes. By default, both HTTP and HTTPS are checked. This means that both HTTP and HTTPS requests will go through the proxy server.

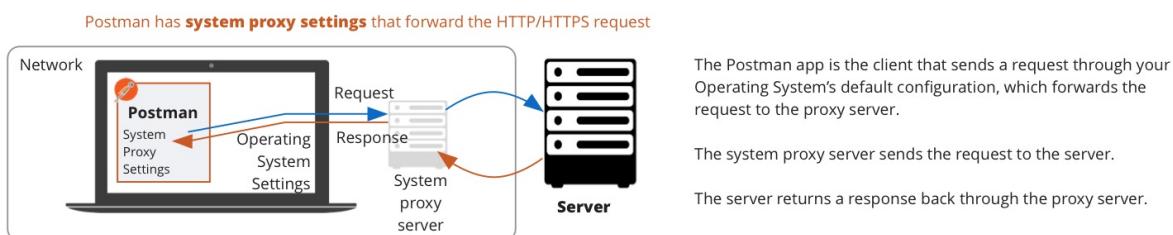
In the first field under **Proxy Server**, enter the **host** or **IP address** (without protocol) of the proxy server. In the second field under **Proxy Server**, enter the **port** of the proxy server.



## Using system proxy

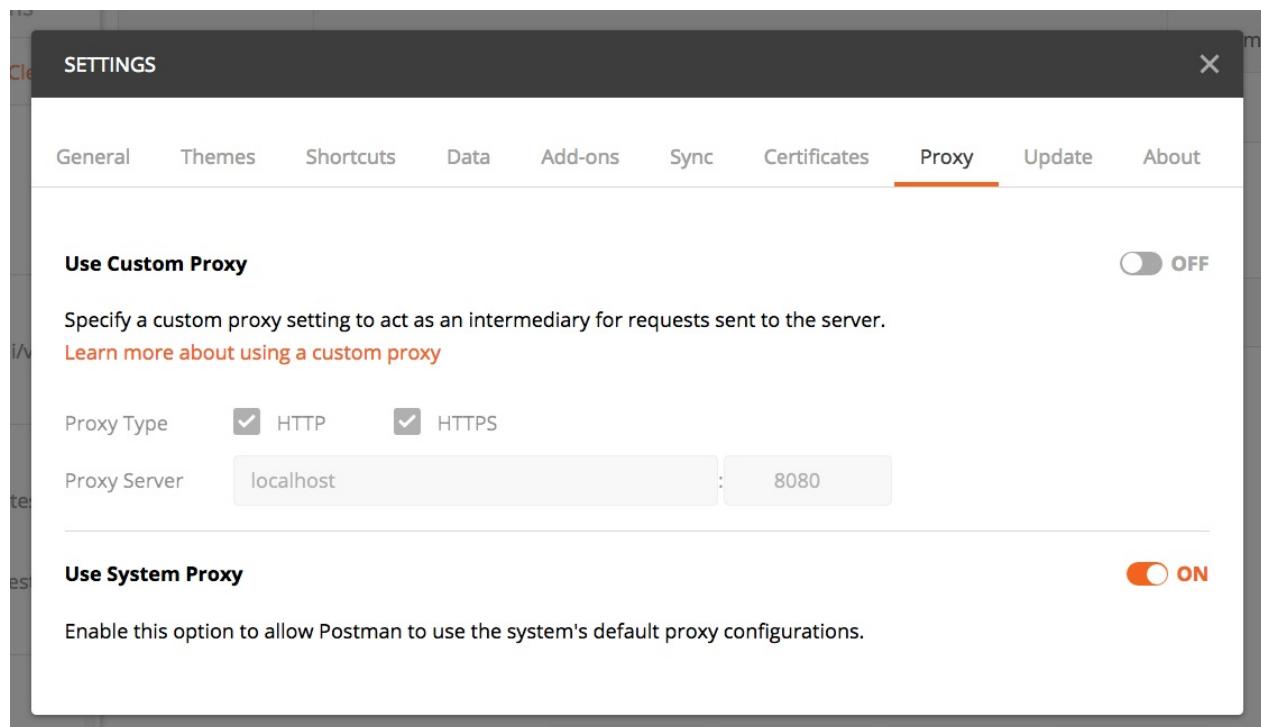
If all of your applications need to use the same proxy, you might have a default proxy configured at the Operating System level. Use the **system proxy settings** to forward your HTTP or HTTPS requests in Postman through your OS's default configuration. In other words, you are telling the Postman app and all requests sent using Postman to follow your OS's default configuration.

1. The Postman app is the client that sends a request through your Operating System's default configuration, which forwards the request to the proxy server.
2. The system proxy server sends the request to the server.
3. The server returns a response back through the proxy server.



System proxy settings are enabled by default. Any request made through Postman will go through the system proxy.

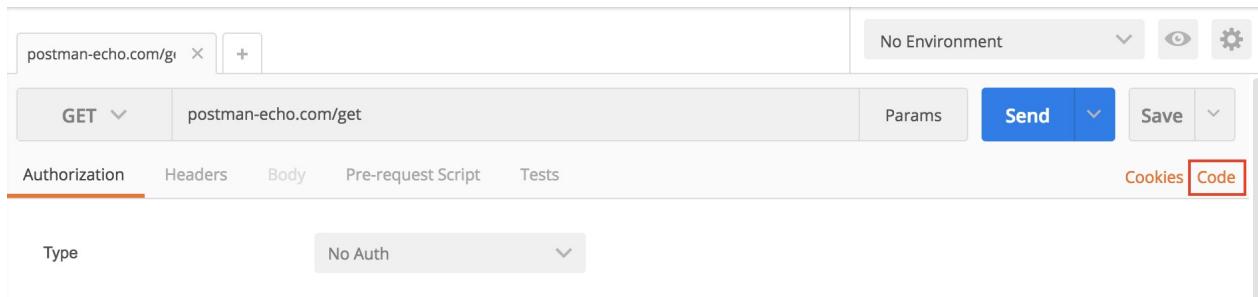
You can turn this setting on and off using the toggle switch. When turned off, all the requests are made directly.



**NOTE:** If the **System Proxy** and the **Custom Proxy** are both turned on, then the **Custom Proxy** will take precedence.

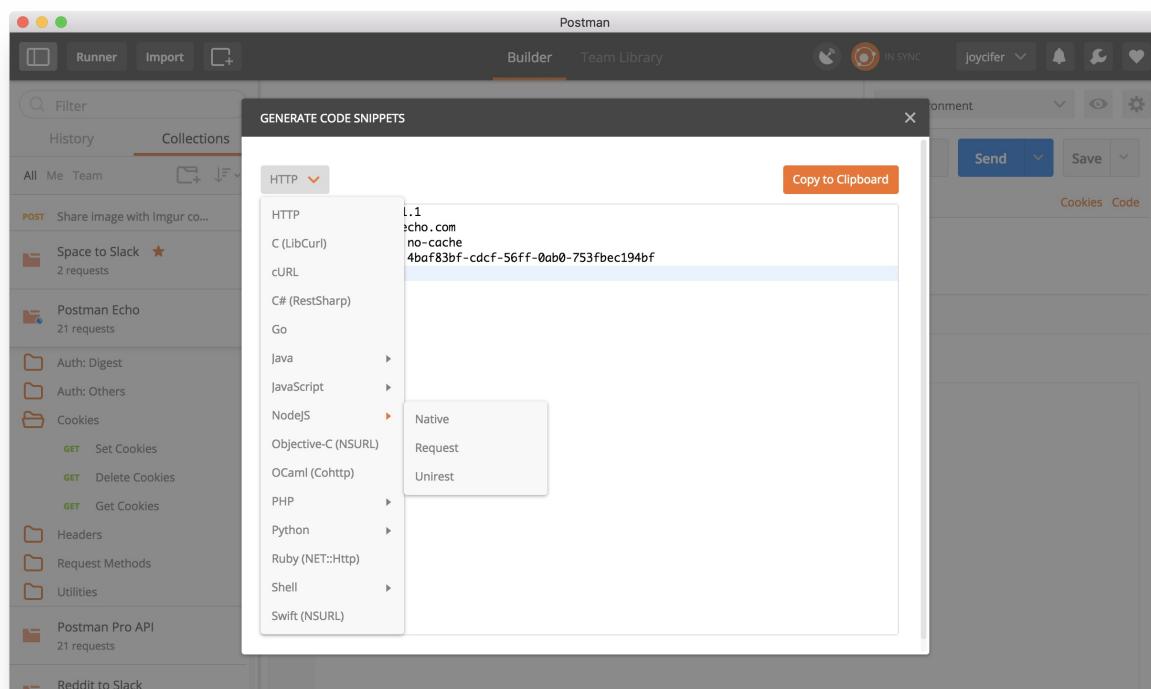
# Generate code snippets

Once you've finalized and saved your request in Postman, you might want to make the same request from your own application. Postman lets you generate snippets of code in various languages and frameworks that will help you do this. You'll need to click the **Code** link under the blue **Send** button to open the **GENERATE CODE SNIPPETS** modal.



## Selecting a language

Use the dropdown menu to select a language - some languages have multiple options. This lets you select different frameworks from which to make your request.



## Supported languages/frameworks

Postman currently supports the following options:

**Language**\*\*  
**Framework**\*\*  
HTTPNone (Raw HTTP request)  
CLibCurlcURLNone (Raw cURL command)  
C#RestSharpGoBuilt-in http package  
JavaOkHttpJavaUnirestJavaScriptjQuery  
AJAXJavaScriptBuilt-in XHRNodeJSBuilt-in http  
moduleNodeJSRequestNodeJSUnirestObjective-CBuilt-in  
NSURLSessionOCamlCohttpPHPHttpRequestPHPpecl\_httpPHPBuilt-in curlPythonBuilt-in  
http.client (Python 3)PythonRequestsRubyBuilt-in  
NET::HttpShellwgetShellHTTPPieShellcURLSwiftBuilt-in NSURLSession

# Making SOAP requests

To make SOAP requests using Postman:

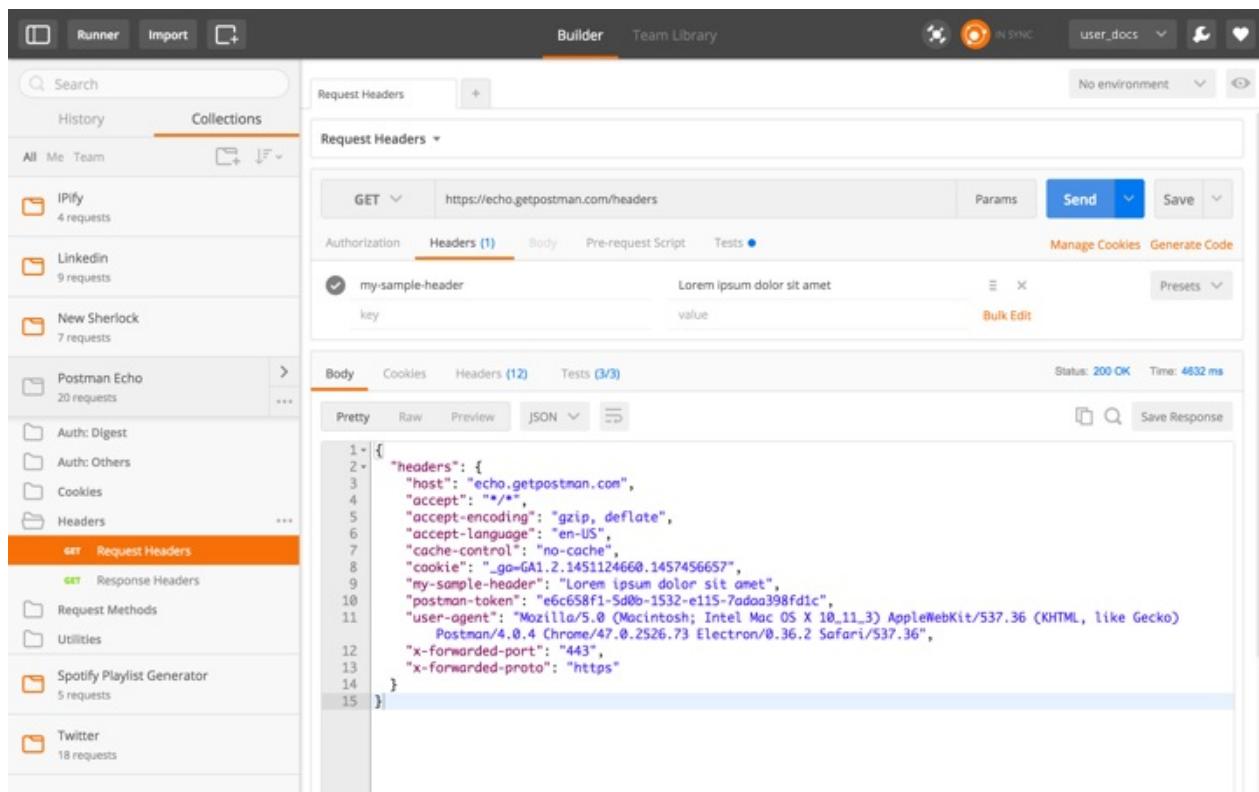
1. Give the SOAP endpoint as the URL. If you are using a WSDL, then give the path to the WSDL as the URL.
2. Set the request method to POST.
3. Open the raw editor, and set the body type as “text/xml”.
4. In the request body, define the SOAP Envelope, Header and Body tags as required. Start by giving the SOAP Envelope tag, which is necessary, and define all the namespaces. Give the SOAP header and the body. The name of the SOAP method (operation) should be specified in the SOAP body.

Check out [an example of making SOAP requests using Postman](#).

# Creating collections

## What is a Postman Collection?

A Postman Collection lets you group individual requests together. These requests can be further organized into folders.

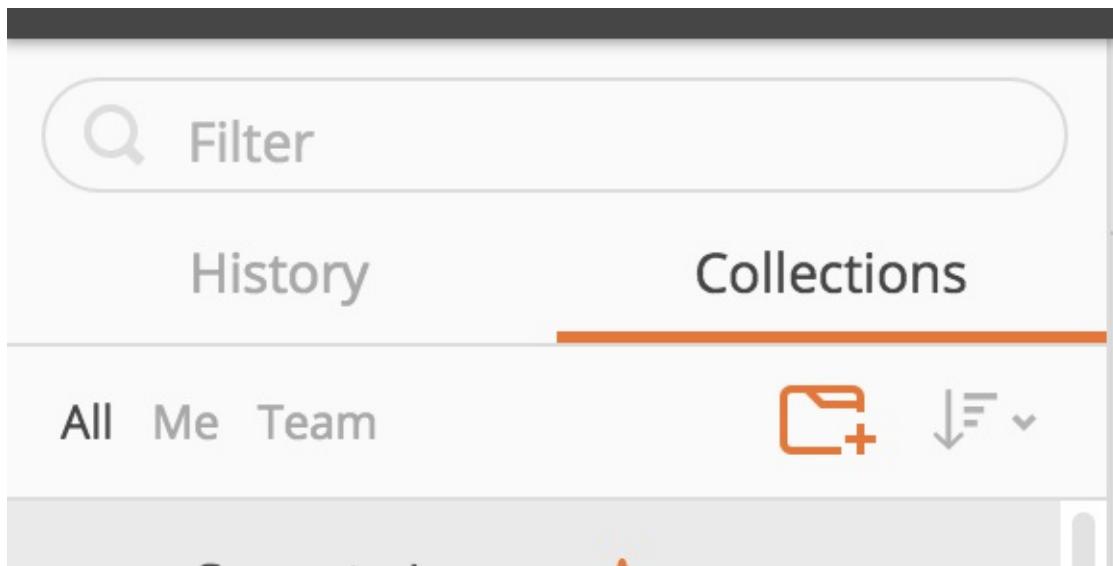


## Why create collections?

- Organization** - Group together requests into folders and collections, so that you don't have to search through your history over and over again.
- Documentation** - Add a name and descriptions to requests, folders, collections. Within Postman, you can use the collection browser to view this documentation. With Postman Pro, you can create and publish beautiful API documentation pages.
- Test suites** - Attach test scripts to requests and build integration test suites.
- Conditional workflows** - Using scripts, you can pass data between API requests and build workflows that mirror your actual use case of APIs.

## Creating a new collection

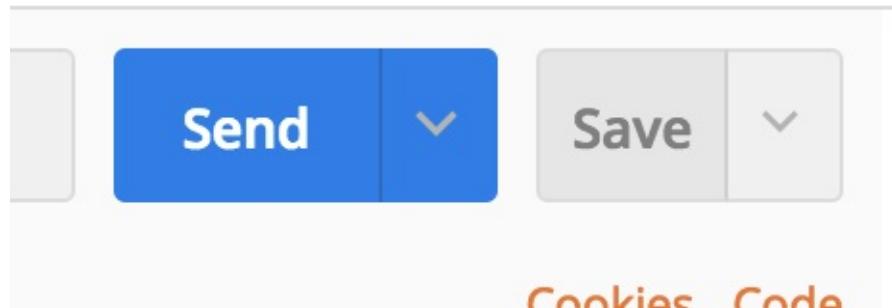
1. Go to the **Collections** tab in the sidebar.
2. Click on the new collection icon in orange below.



3. Enter a name for your collection (and optional description).

## Saving a request to a collection

1. Create a new request in the builder.



2. Hit the **Save** button.
3. Choose an existing collection or a new collection, and hit **Save**.

## Saving to a collection from history

1. To save a single request to a collection, hover over a request under the **History** tab and click the plus icon (+).

The screenshot shows the Postman interface with the 'History' tab selected. At the top right, there is a 'Clear all' button. Below it, a section titled 'Today' is expanded, showing a single request: a GET request to `https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY&hd=True`. To the right of this request are three icons: a plus sign (+), a trash can, and three dots (...).

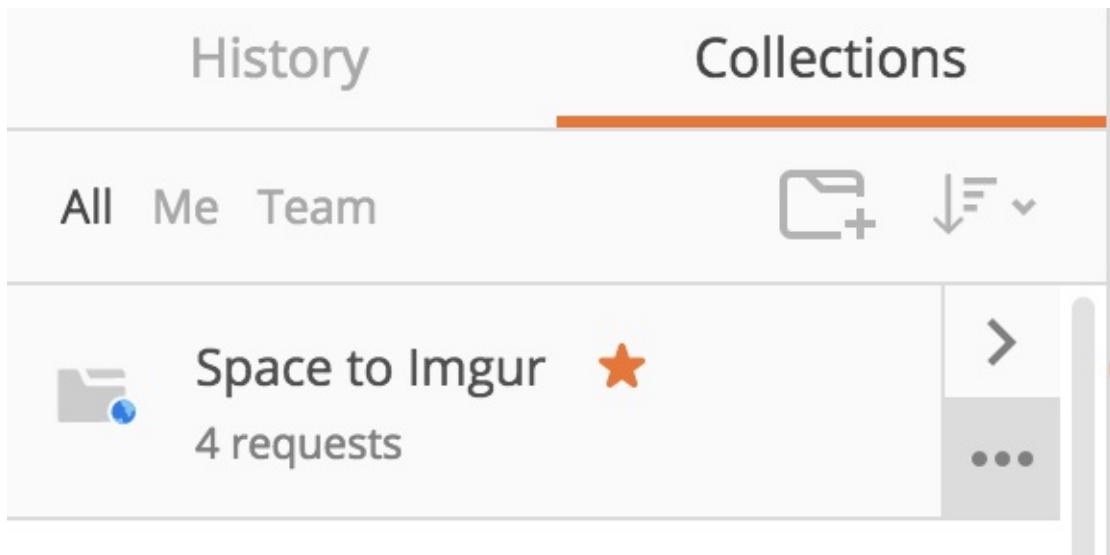
- When selecting multiple requests, click the plus icon (+) that displays at the top of the sidebar.

The screenshot shows the Postman interface with the 'History' tab selected. At the top right, there is a 'Clear all' button. Below it, a message '2 requests selected' is displayed, followed by a plus sign (+), a trash can, and three dots (...). The 'Today' section is expanded, showing two requests: a GET request to `https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY&hd=True` and a POST request to `https://postman-echo.com/post`.

- Chose an existing collection or a new collection, and hit **Save**.

## Duplicating an existing collection

- Click the ellipses (...) to expand the control dropdown.

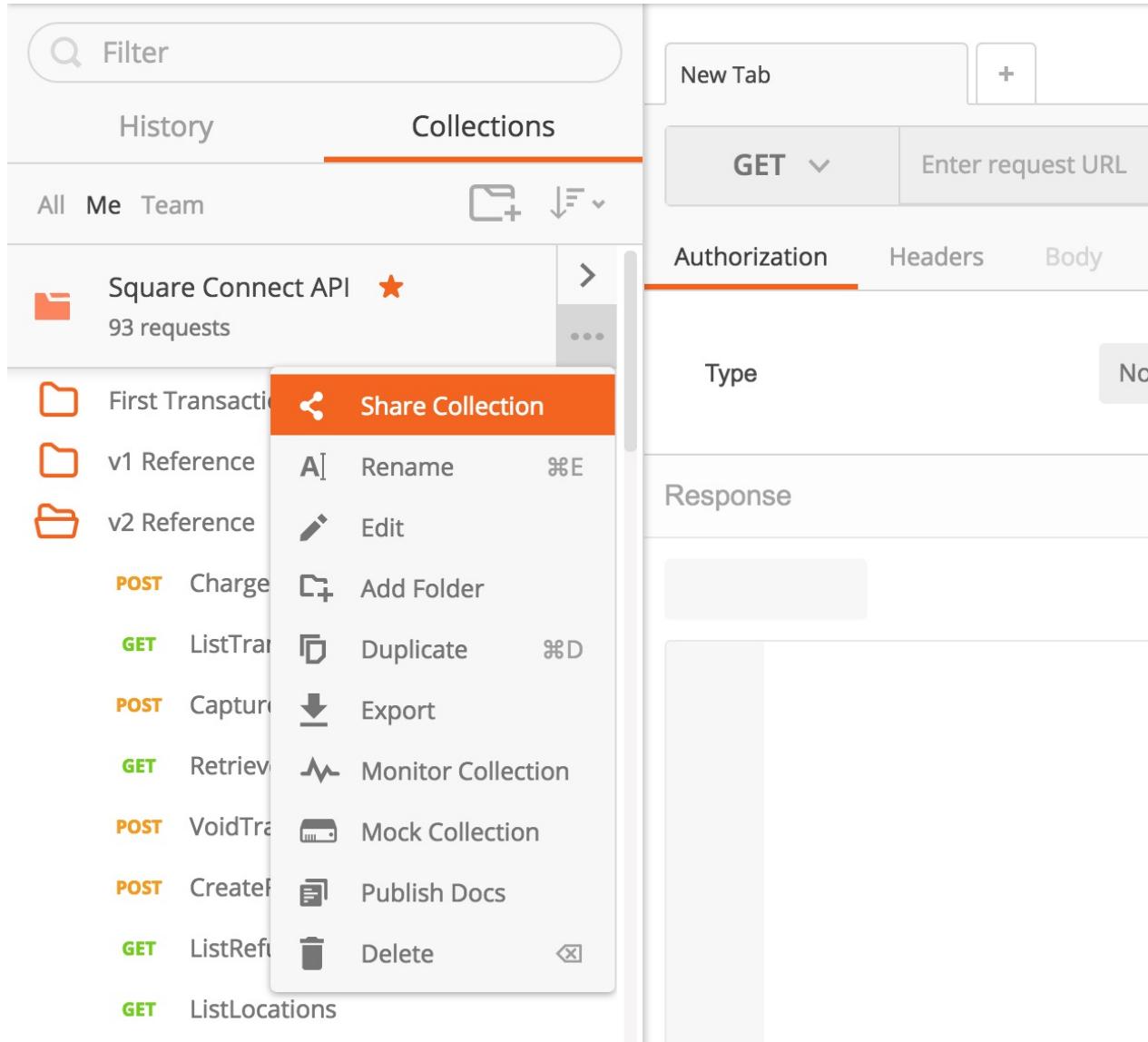


2. Select “Duplicate” from the menu.

Learn how to [share collections](#), [importing and exporting collections](#), and [other collection features](#).

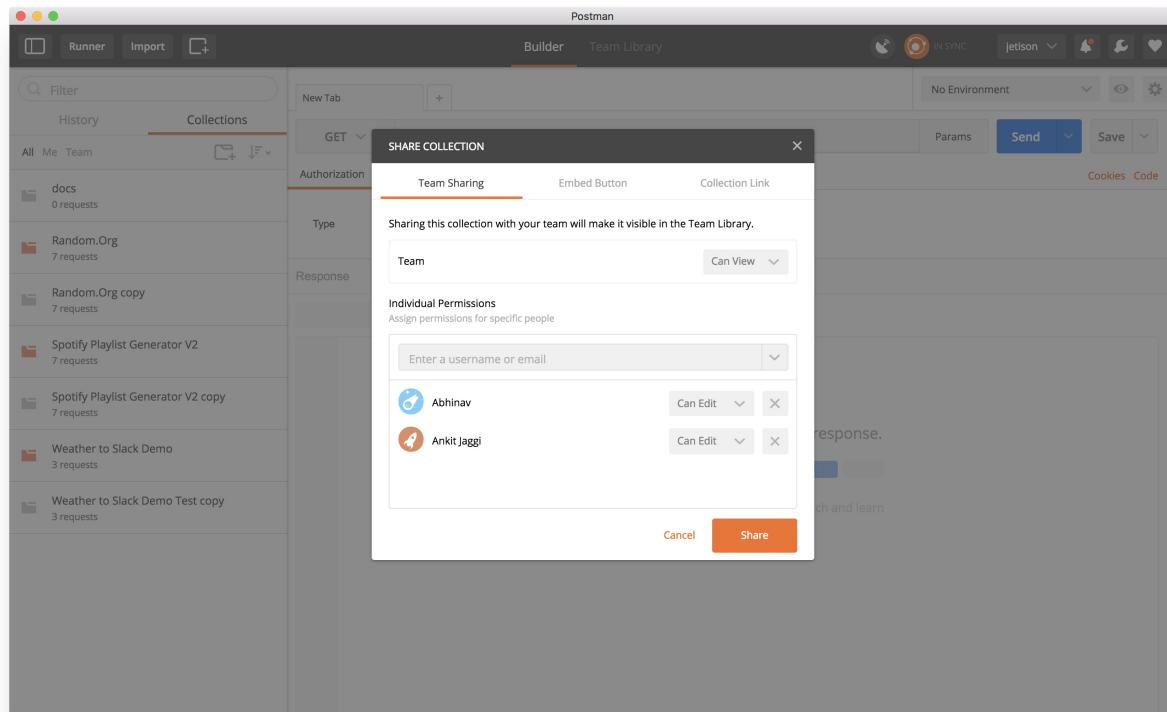
# Sharing collections

You must be signed in to your [Postman account](#) to upload or share a collection. Click on the ellipses (...) next to the collection you wish to share. Select “Share” to open the **SHARE COLLECTION** modal which will contain most of ways that you can share a collection.



## Sharing a collection with your team (Pro feature)

If you are a member of a team using Postman Pro or Enterprise, you can [share a collection with the rest of your team](#). Under the **Team Sharing** tab of the **SHARE COLLECTION** modal, you can designate view or edit permissions for your team. You can also choose to share the collection with your whole team or assign individual permissions for team members.



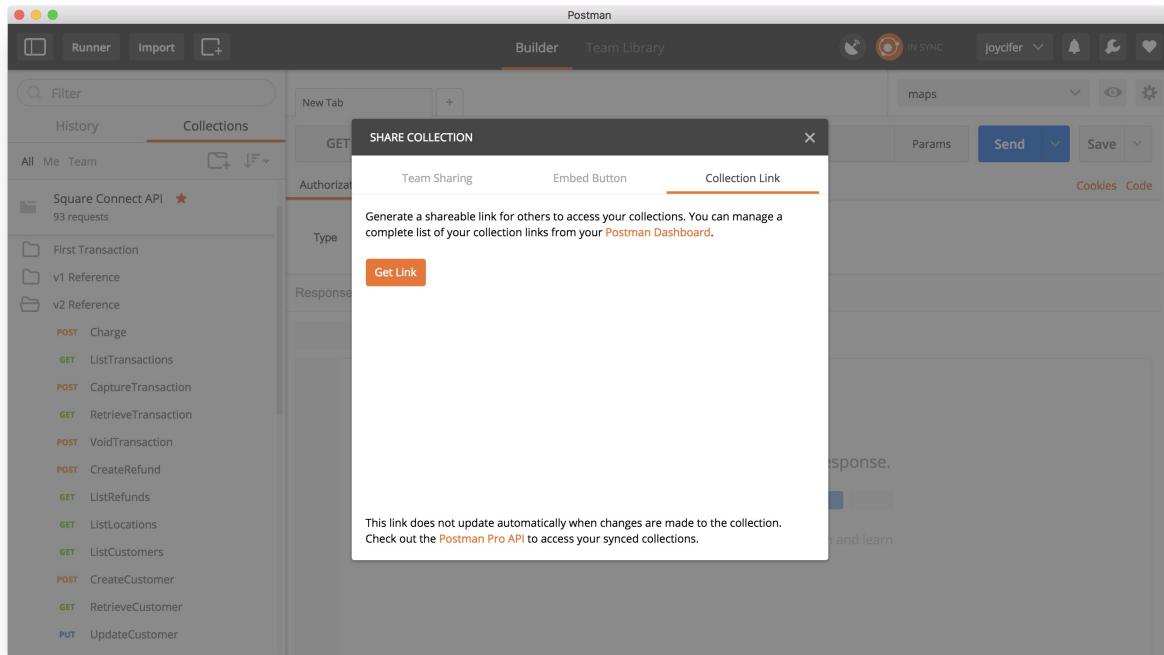
A team's shared collections can be viewed in the [team library](#). This is where you can subscribe to collections that others have shared.

Collection	Description	Owner	Actions
Postman Pro API	The Postman Pro API allows you to programmatically access data stored	joycifer	<a href="#">View Docs</a> <a href="#">Subscribe</a>
Spotify Playlist Generator V2	7 requests 0 subscribers	joycifer	<a href="#">View Docs</a> <a href="#">Subscribe</a>
Postman Echo	Postman Echo is service you can use to test your REST clients and make	joycifer	<a href="#">View Docs</a> <a href="#">Subscribe</a>
Reddit to Slack	1. GET request from Reddit to retrieve the first post	joycifer	<a href="#">View Docs</a> <a href="#">Subscribe</a>
Space v1	Super Cool Space Collection This is a sample collection and	joycifer	<a href="#">View Docs</a> <a href="#">Subscribe</a>

## Sharing with a link

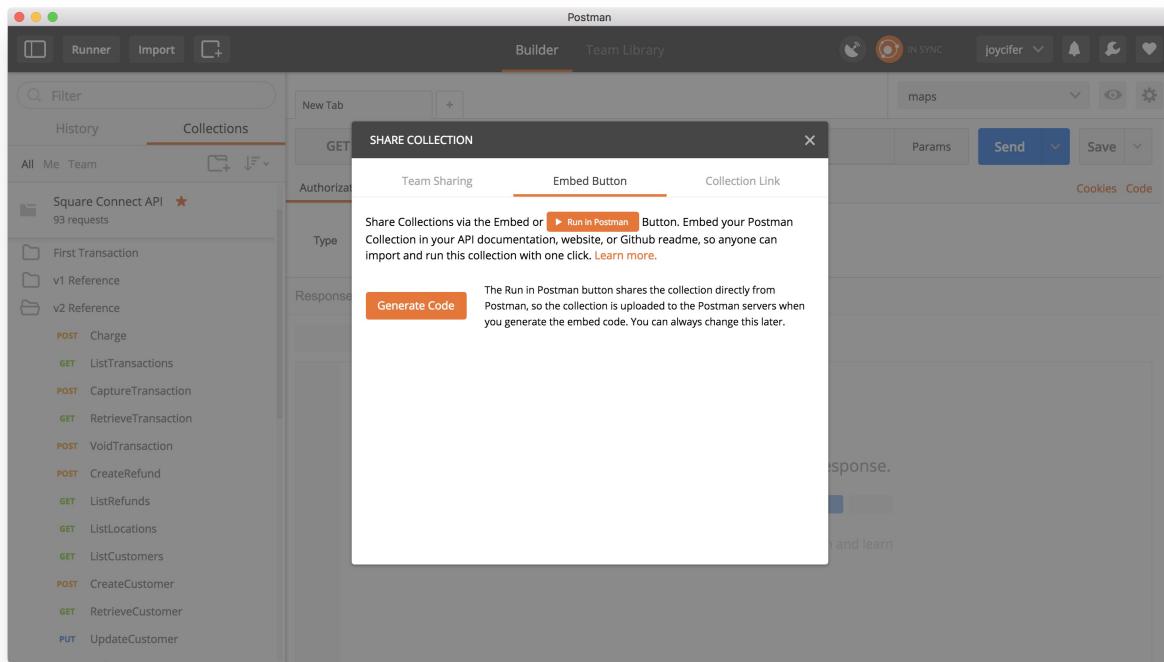
Generate a shareable link for others to access your collections. This is not the recommended method for sharing collections. Collection links reflect the collection as a snapshot in time, and must be updated to refresh the changes to the collection.

You can manage a complete list of collection links from the [dashboard](#).



## Sharing as a Run in Postman button

Under the **Embed Button** tab of the **SHARE COLLECTION** modal, you can create a **Run in Postman** button to share your collection. Just like with collection links, the collection should be manually updated to reflect new changes in the collection. Learn more about [generating and embedding the button](#).



## Sharing as a file

Collections can be downloaded as a JSON file which you can share with others, with or without signing in through your Postman account. You can share collections anonymously, but it is strongly recommended to sign in to your Postman account when uploading collections. This will let you update your existing collection, make it public, or delete it later.

Learn more about [exporting and importing collections](#), and the differences between collection formats [v1](#) and [v2](#).

# Managing collections

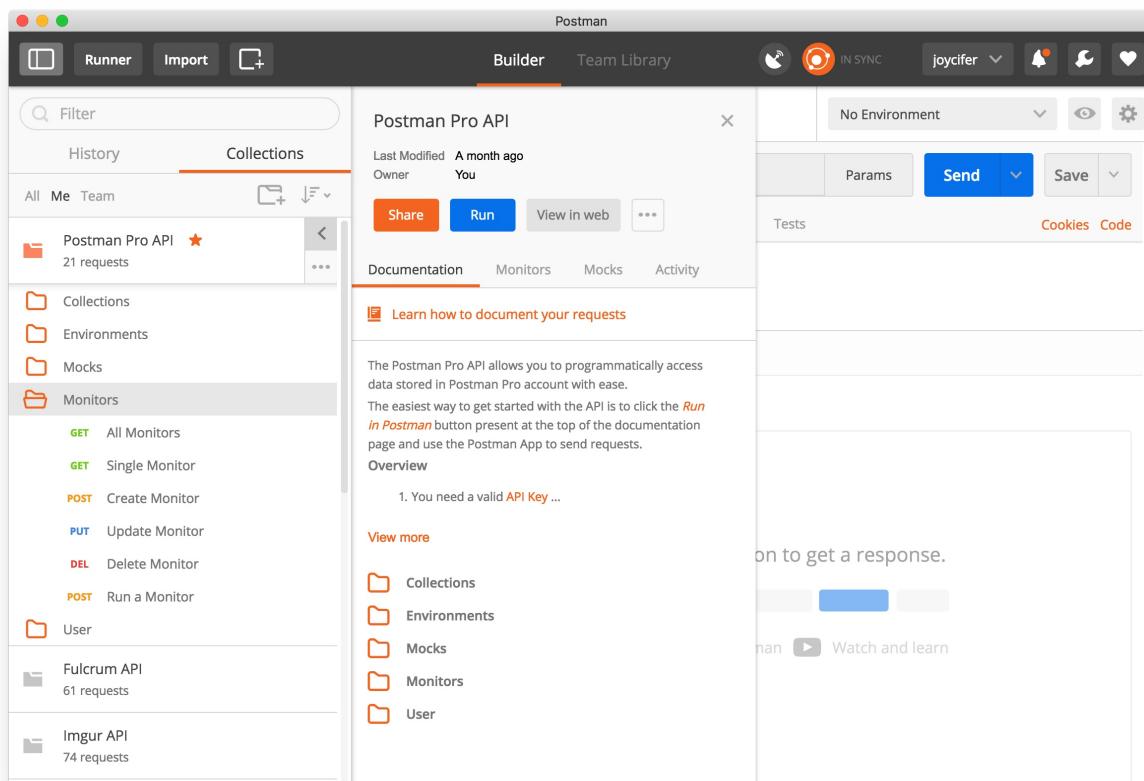
From the **Collections** tab in the sidebar:

## Navigate through collections

Click on a collection to show or hide the requests that comprise the collection. Use the up and down arrow keys on your keyboard to navigate through the collections.

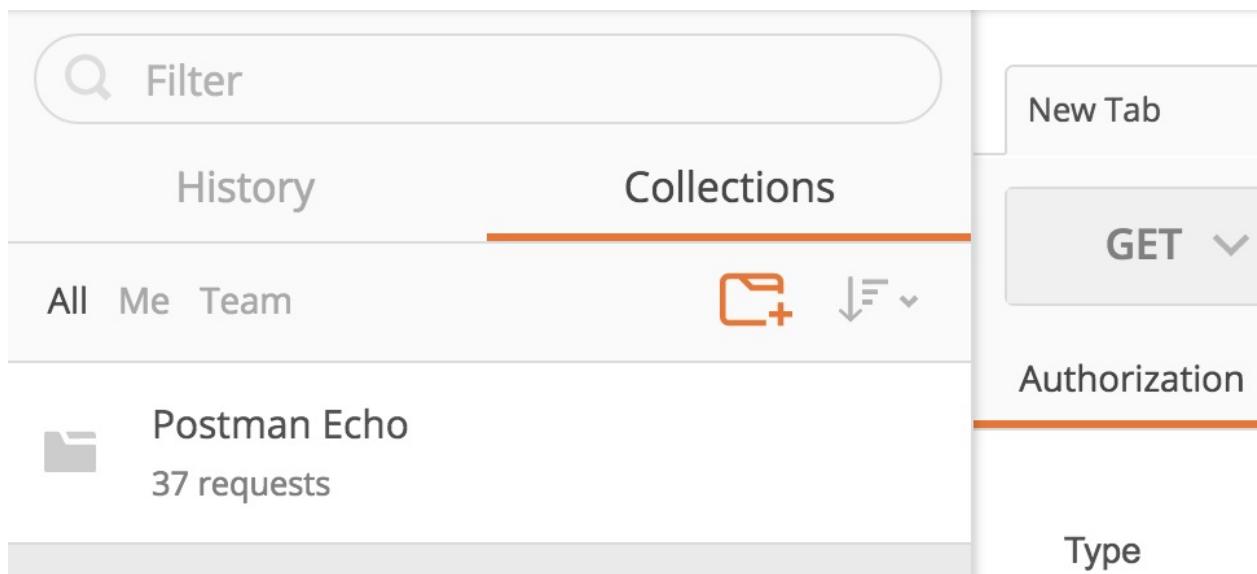
## Edit and view collection details

Expand the right angle bracket (>) to show the details view for the collection. Collapse the left angle bracket (<) to hide the details view. You can add metadata like name and description so that all the information a developer needs to use your API is available easily.



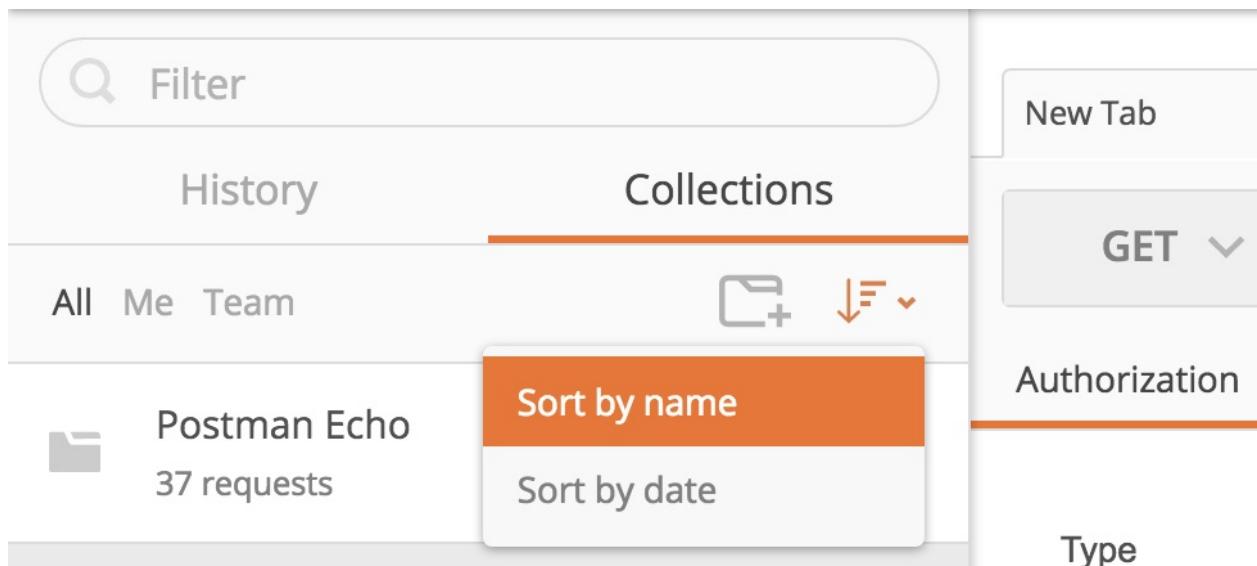
## Create a new collection

Click the “new collection” icon on the top right, or [save a current request to a new collection](#).



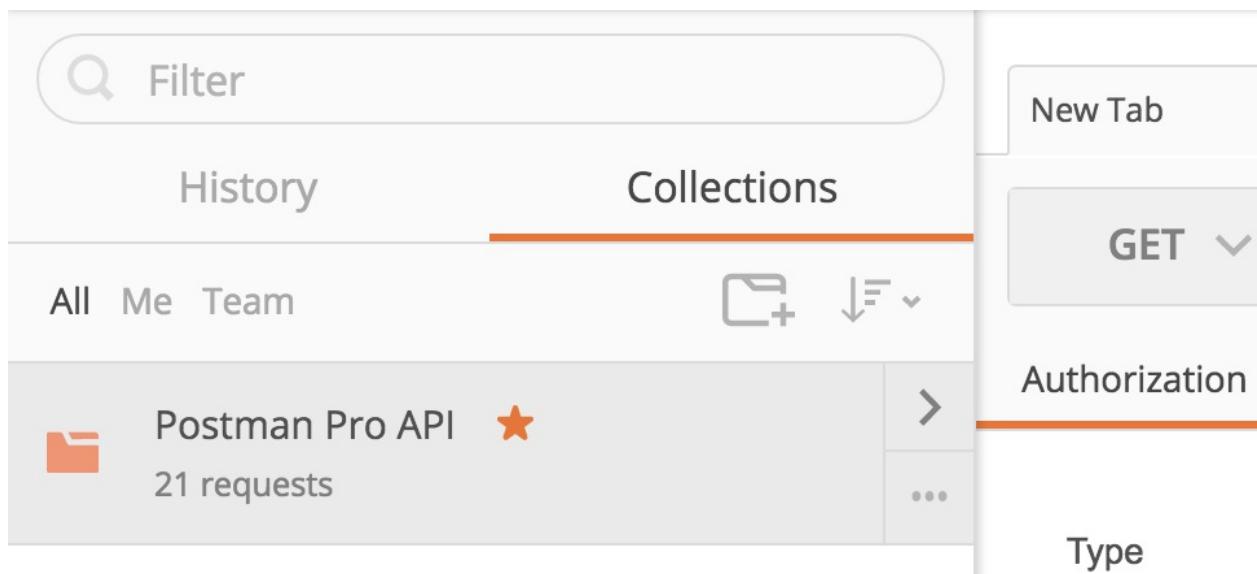
### Reorder collections

Collections can be sorted either alphabetically by name or by when they were last updated. To do this, select the "sort" icon on the top right and select Sort by name or Sort by date.



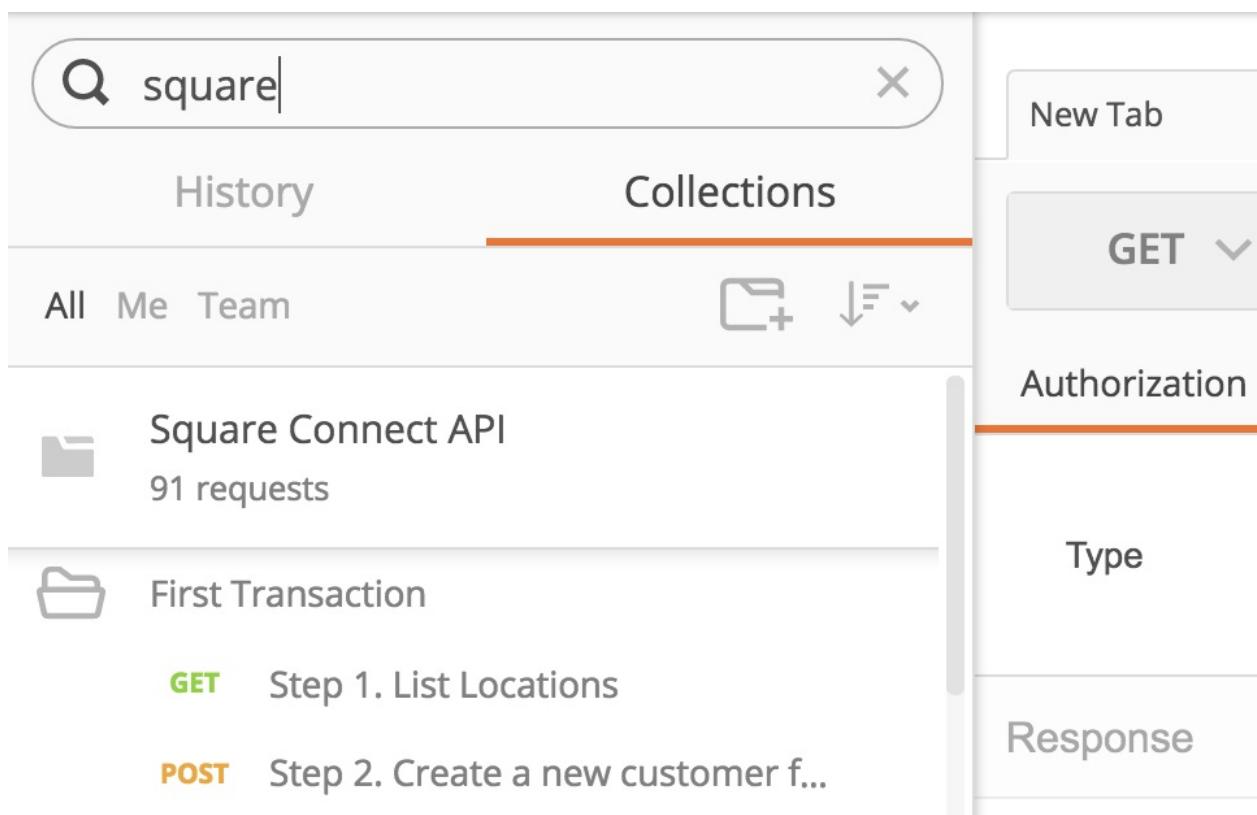
### Favoriting a collection

If you're working on a few collections in particular, you can click on the star icon to bring the collection(s) to the top of the list.



### Filter collections

If you have a lot of collections, filter collections in the sidebar using the search input field.



### Delete a collection

Click the ellipses (...) next to a collection, and select “Delete”. If you didn’t intend to delete the collection, you can click the **Undo** link in the notification that appears at the top of the Postman app.



## Share a collection

Learn more about [sharing collections](#).

## Other collection features

### Reorder requests

Within a collection or folder, you can reorder requests using drag and drop. You can also reorder folders within a collection using drag and drop.

### Save responses

Requests can also store [sample responses](#) when saved in a collection.

### Use examples

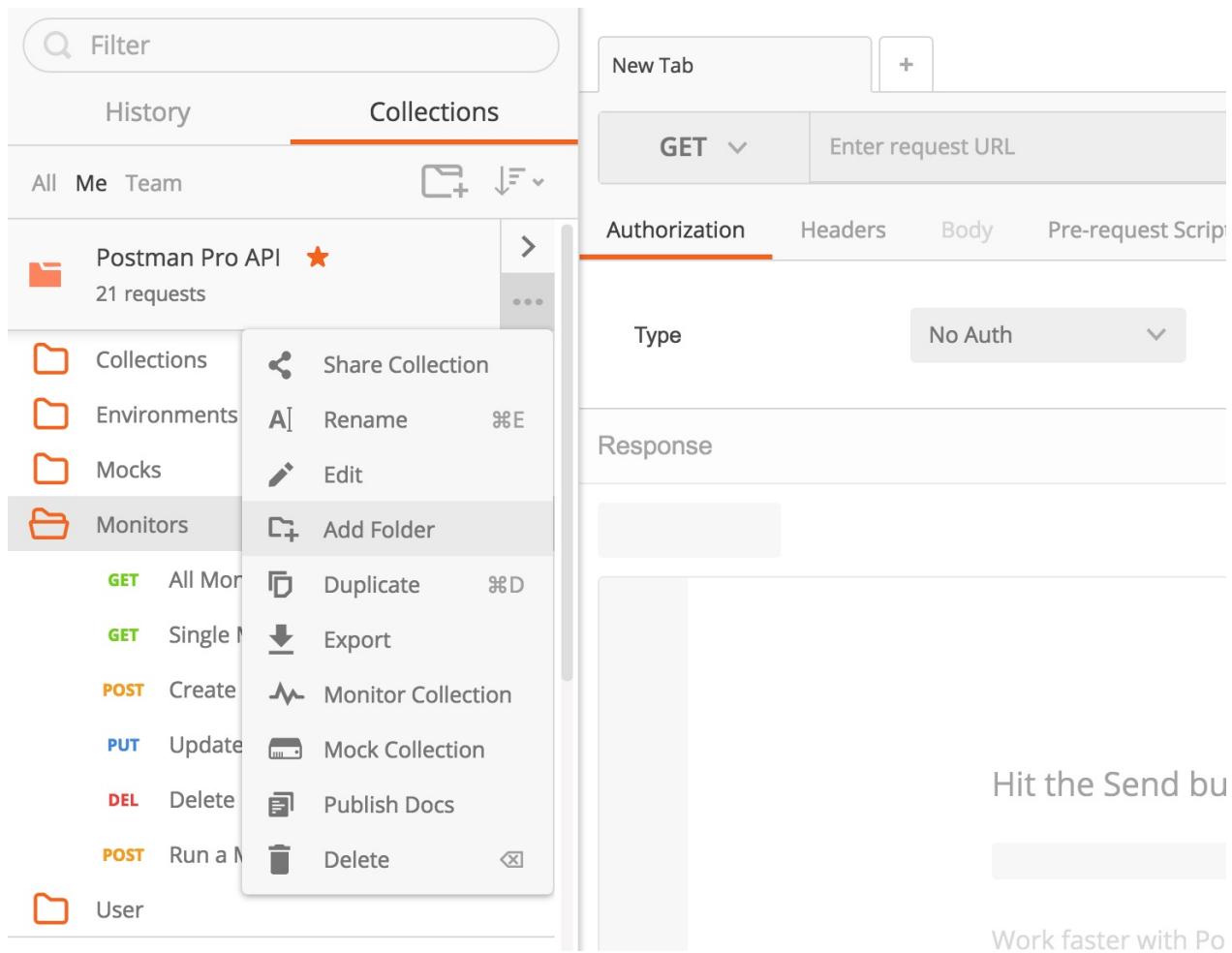
With [examples](#), you can mock raw responses and save them to a collection. Then, you'll be able to generate a mock endpoint for each of them using Postman's [mock service](#).

### Add scripts

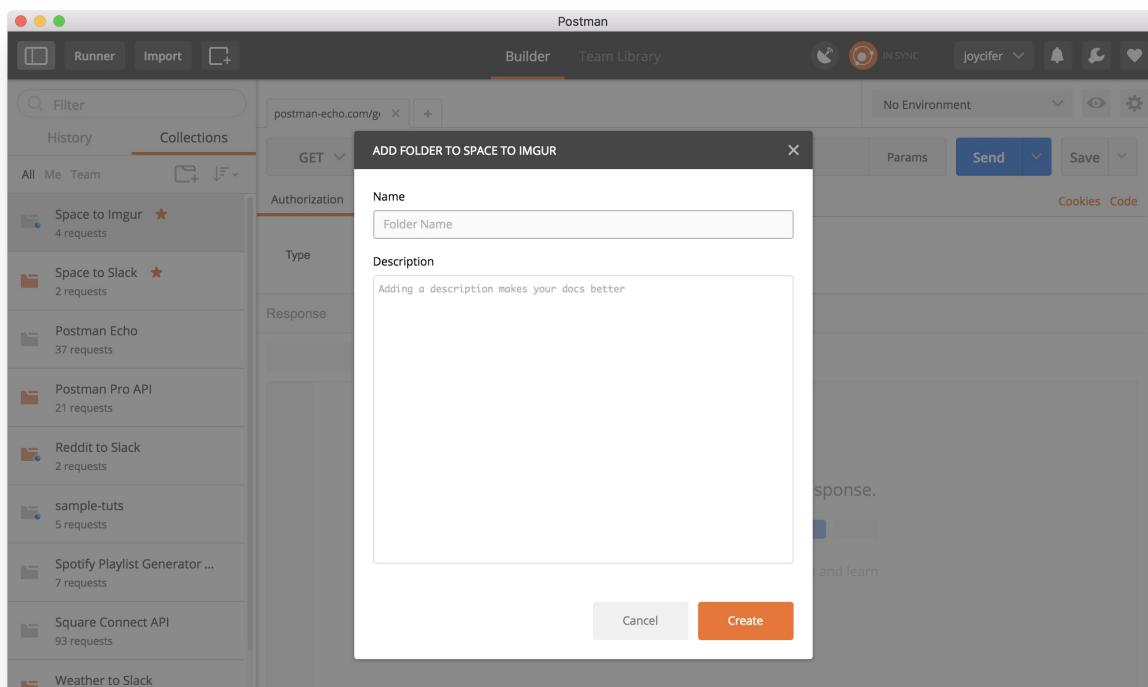
Requests stored inside a collection can contain [scripts](#) to add dynamic behavior to the collection.

## Adding folders

Folders are a way to organize your API endpoints within a collection into intuitive and logical groups to mirror your workflow. Next to the collection to which you want to add a folder, click on the ellipses (...) and select “Add Folder”.



Add a name and description to the folder. Folders are initially ordered alphabetically by name, and the folder name and description will be reflected in your API documentation.



You can add deeper levels of nesting for folders. Drag and drop the folders to reorder them to create the ultimate customized folder structure.

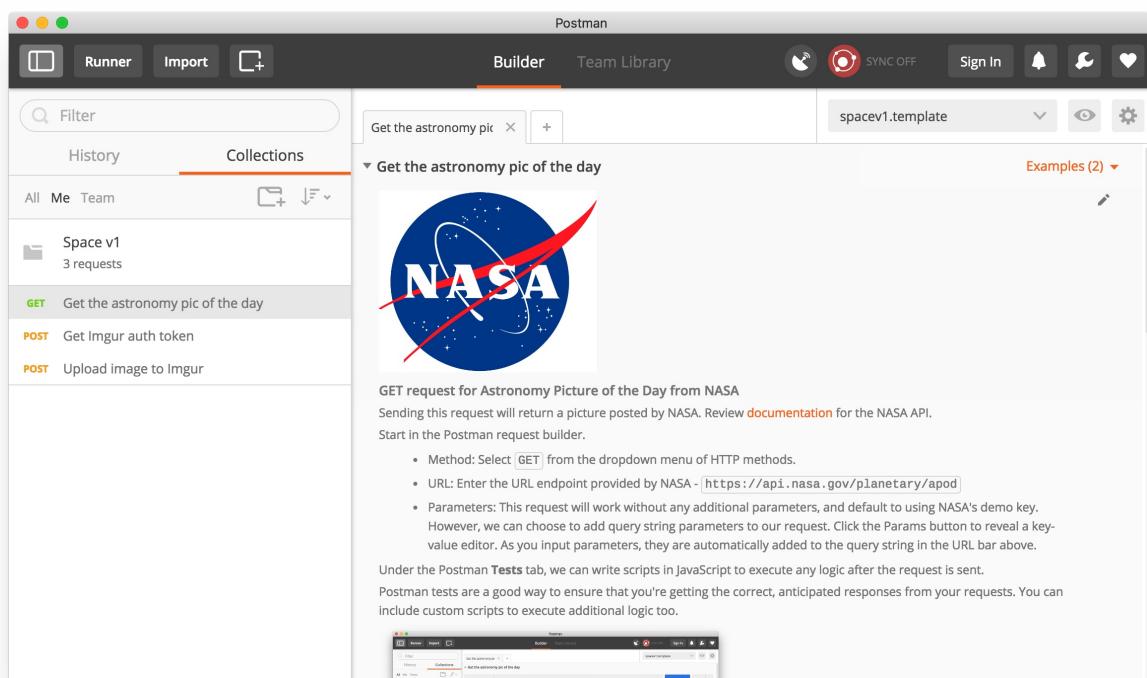
# Using Markdown for descriptions

Postman supports [Markdown](#) as a way to style text descriptions for requests, [collections](#), and [folders](#) within collections. You can even embed screenshots and other images for more descriptive flair.

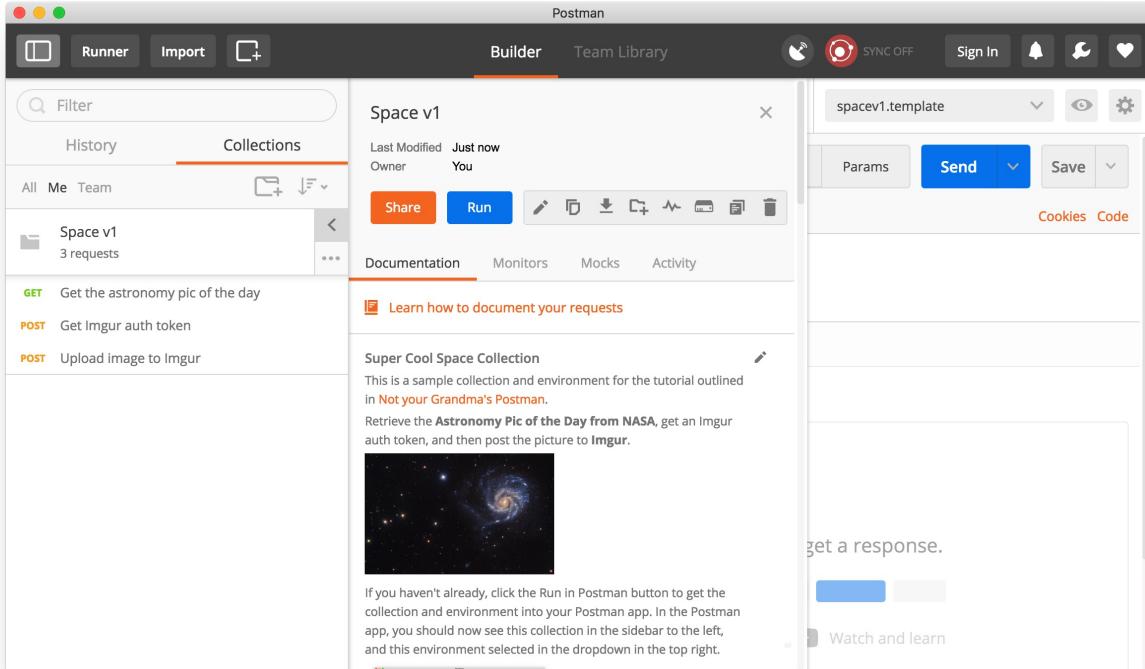
Review [reference for using Markdown](#) in API documentation.

Postman renders this markdown in the following places:

In the request builder, the request description can be styled with markdown.



The collections details view can be styled with markdown in the descriptions for collections and folders.



For either public or internal [API documentation](#), automatically generated API descriptions will be styled beautifully and precisely with markdown.

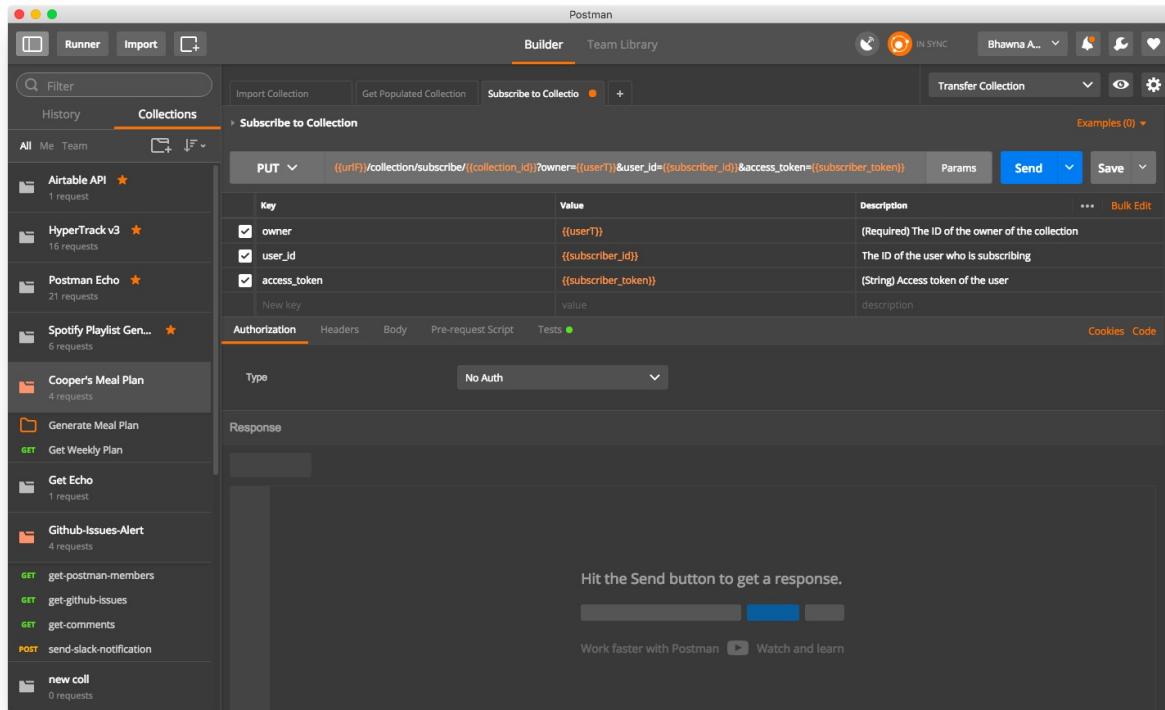
The screenshot shows a browser window displaying API documentation for the 'Space to Imgur' collection. On the left, there's a sidebar with 'LAST UPDATED A MINUTE AGO' and sections for 'Introduction', 'Super Cool Space Collection', and 'Space to Slack'. Under 'Space to Slack', there's a 'GET Astronomy Picture of the Day' endpoint. Below that, there are 'POST Upload image to Imgur ANON' and 'POST Upload image to Imgur via OAuth 2.0' endpoints. There's also a 'POST Share image with Imgur community' endpoint. The main content area is titled 'POST Upload image to Imgur ANON' and shows the URL 'https://api.imgur.com/3/upload'. Below this, there's a section titled 'Upload an image anonymously to Imgur' with usage instructions. On the right, there's a 'Sample Request' section containing a curl command:

```
curl --request POST \
--url https://api.imgur.com/3/upload \
--header 'authorization: Client-ID 9cddb08dc9da19f' \
--header 'content-type: multipart/form-data; boundary=---01100001011100
--form 'image={{spaceUrl}}' \
--form 'title={{spaceTitle}}' \
--form 'description={{spaceExplanation}}'
```

## Descriptions for request attributes

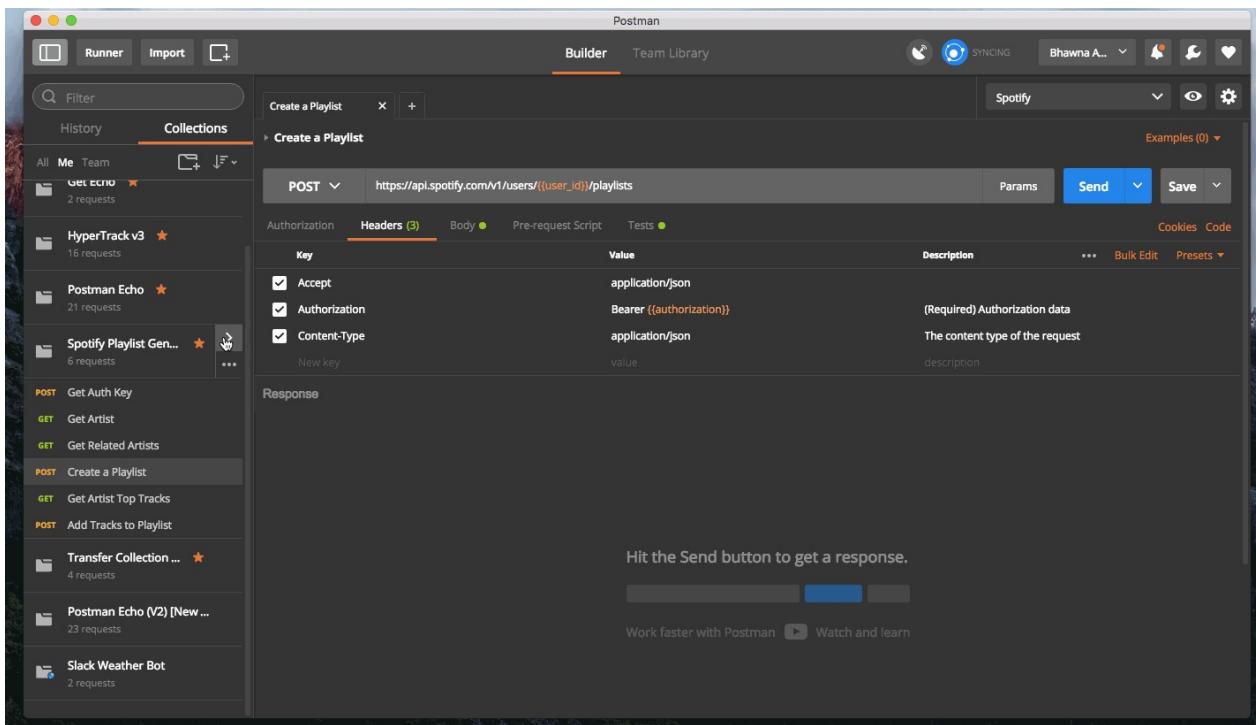
The description column in the [data editor](#) makes your requests easier to understand. You can add comments and details for each of your query parameters, path variables, headers, and body (form-data and urlencoded) - all from right within the Postman app.

For example, specify if an element is required or optional, indicate the accepted data type, or use alternative terminology to provide additional clarification for developers who are working with your requests.



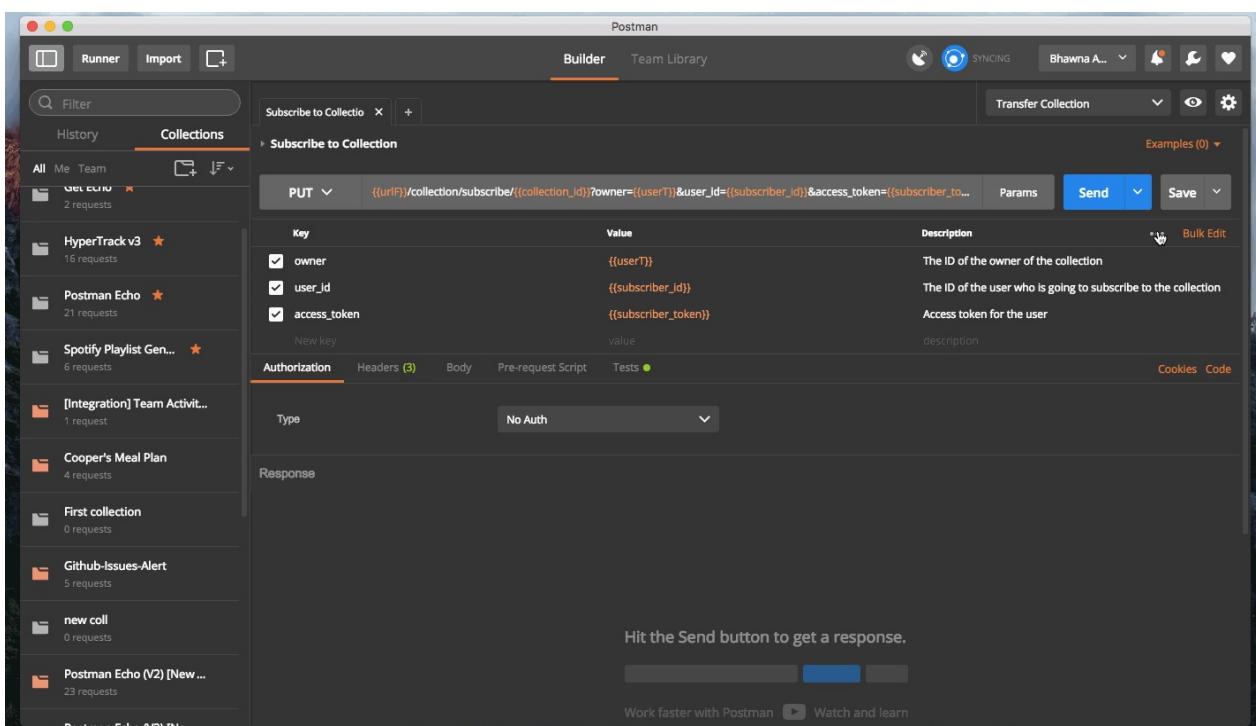
For Postman users publishing internal or public API documentation, these descriptions are displayed in the [automatically generated documentation](#) for that collection.

## 描述使用 Markdown



*Note: Descriptions for path variables and URL params are currently NOT shown in the documentation.*

You can hide and show the value and description column in the data editor by clicking on the ellipsis (...) in the top right corner of the editor, and unchecking the columns that you want to hide.



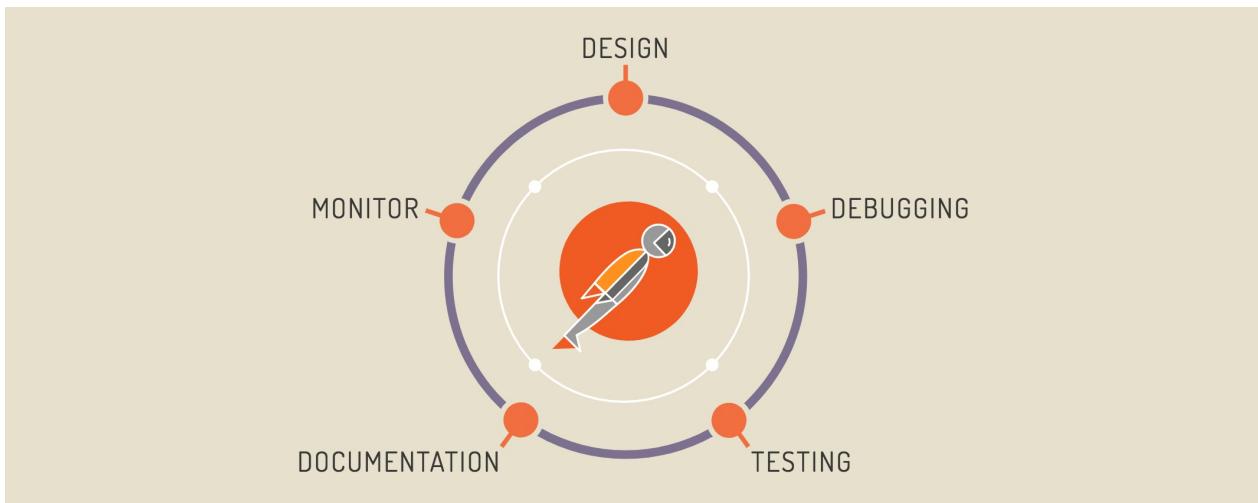
*Note: Descriptions are metadata for a request and are NOT sent with your HTTP request. This reminder is displayed when you mouse over the title of the description column.*

## 描述使用 Markdown

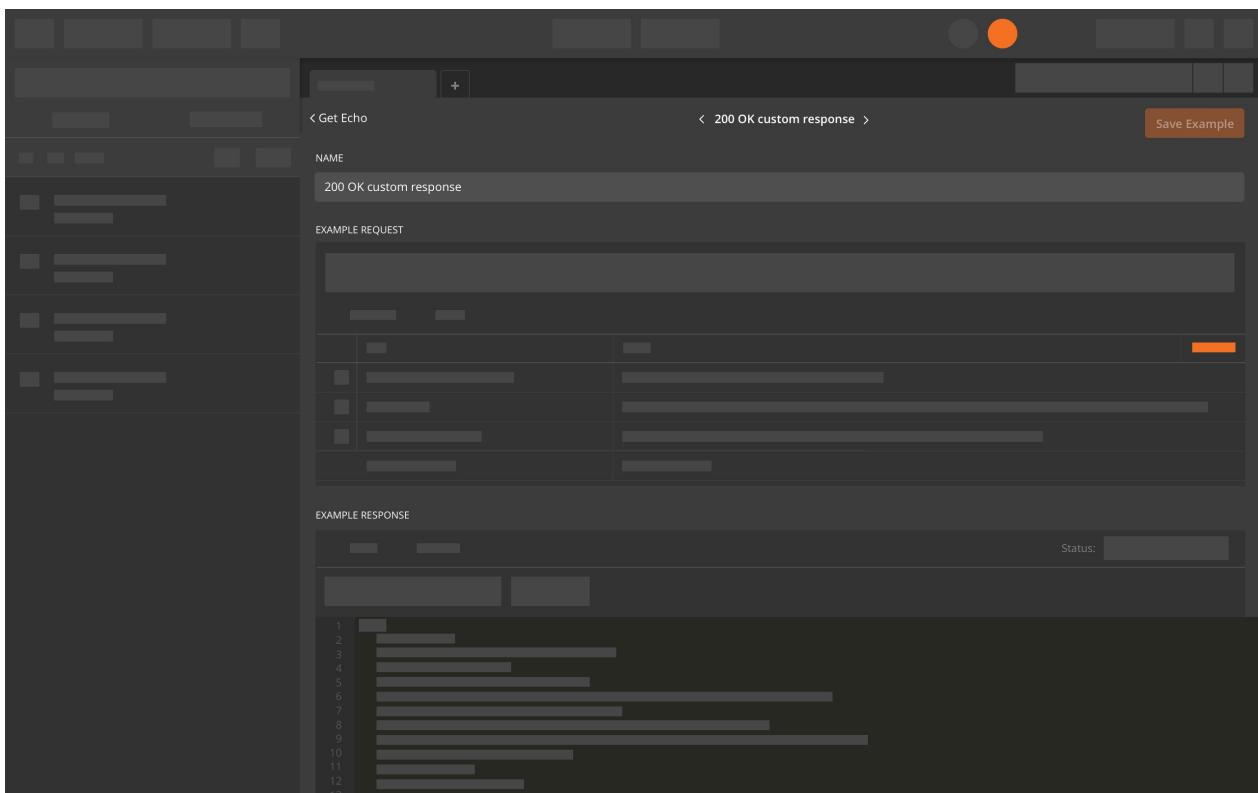
The screenshot shows the Postman application interface. On the left, there's a sidebar titled 'Collections' containing a list of various API collections. The main area is titled 'Builder' and shows a 'Subscribe to Collection' request. The method is set to 'PUT' and the URL is `({{url}})/collection/subscribe/{{{collection_id}}}?owner={{user_id}}&user_id={{subscriber_id}}`. The 'Authorization' tab is selected, showing 'Type: No Auth'. Below the request area, a message says 'Hit the Send button to get a response.' and there's a 'Work faster with Postman' button.

# Examples

Developers can mock a request and response in Postman before sending the actual request or setting up a single endpoint to return the response. Establishing an **example** during the earliest phase of API development requires clear communication between team members, aligns their expectations, and means developers and testers can get started more quickly.



## What is an example?



An example is a tightly coupled request and response pair. For instance, in this illustration, ‘200 OK custom response’ is the name of an example, which contains an ‘example request’ and an ‘example response’.

## Why use examples?

More often than not, it is useful to create and save a couple of example responses alongside a request – status codes for a 200, a 404, a 500, etc. – to make your API more understandable to others. Thus, a teammate looking at your API can quickly view these examples and get a good idea of the responses a particular request is going to return – all this, without having to hit ‘Send’ on the request.

Furthermore, let’s say that you are going to build an API with an endpoint which does not yet exist, or your server just isn’t ready. With examples, you can mock raw responses and save them. Then, you’ll be able to generate a mock endpoint for each of them using [Postman’s mock service](#). With this setup, developers can make requests to the mock endpoint, and get started on front-end development or [writing tests](#) based on the mock response returned from the mock endpoint.

## Adding an example

The screenshot shows the Postman application interface. On the left, there's a sidebar with collections like 'Airtable API' and 'Postman Echo'. The main area shows a 'Create Design Job' request under the 'POST' method. In the 'Headers' tab, 'Authorization' and 'Content-Type' are set. The 'Body' tab contains a JSON object representing a design job. The 'Response' pane shows a successful 200 OK response with a detailed JSON structure.

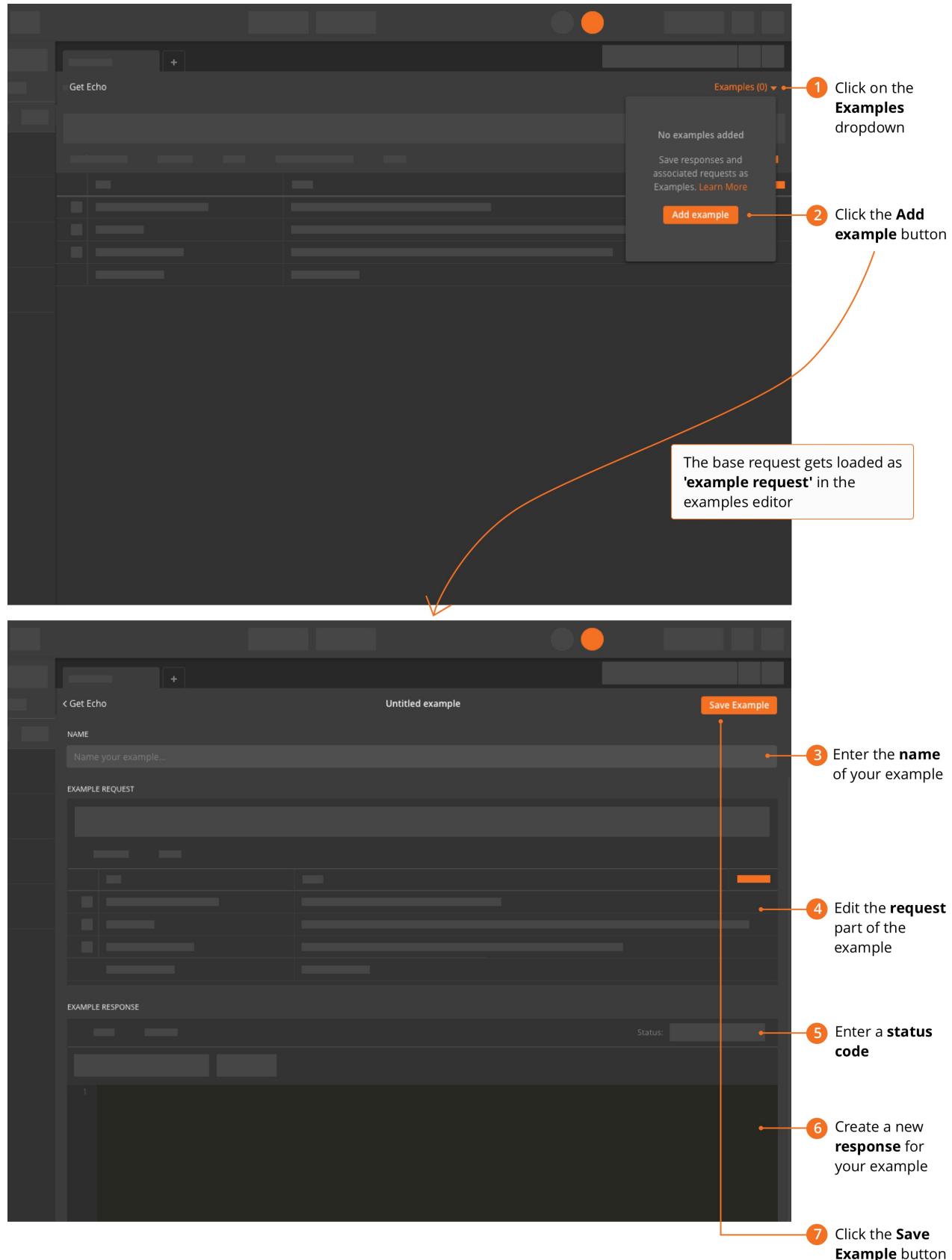
```

1 - {
2 -   "Id": "rec8LKEZl0mVxZdk",
3 -   "fields": {
4 -     "Job Request": "Assets for new software update",
5 -     "Job Description": "We need creative assets for the launch of our new software update\nwill allow the PorchCom to be remotely p...",
6 -     "Assigned to": [
7 -       "recmpZlujMkIkg"
8 -     ],
9 -     "Assets approved?": true,
10 -    "Submitter": "Katrina Dickson",
11 -    "Status": "Completed",
12 -    "Active Projects": 0,
13 -    "Submitter Dept.": "Communications & PR",
14 -    "Submitted": "2016-10-26",
15 -    "Priority": "2. Important (\u201cThis is necessary, but not needed ASAP\u201d)",
16 -    "Due Date": "2016-11-02",
17 -    "Expected Completion": "2016-10-28",
18 -    "Type of Asset": "Web",
19 -    "Target Audience": "Targeted at both existing PorchCom users and people who are following/interested in PorchCom",
20 -    "Attachments": []
}
  
```

Adding examples to each of your API endpoints involves just a few clicks. Let’s say you are working on a request that is saved inside a [collection](#). You can add examples to this request with **a new custom response** or **the response received from the server**.

### A new custom response

Examples let you define what the response should look like by letting you create your own custom responses from scratch. The illustration below outlines the steps for creating an example with a new response.



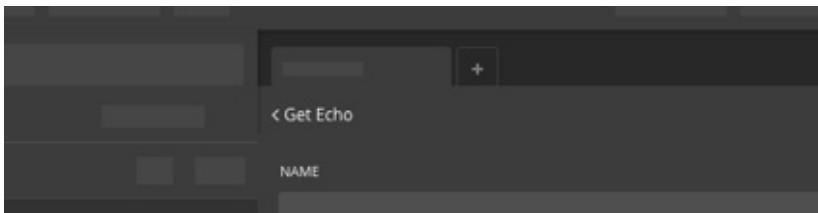
1. Click on the **Examples** dropdown.
2. Click the **Add Example** button. The base request gets loaded as ‘example request’ in the examples editor.
3. Enter the **name** of your example.
4. Edit the **request** part of the example.
5. Enter a **status code**.
6. Create a new **response** for your example.
7. Click the **Save Example** button in the upper right corner of the builder (**CMD/CTRL + S**) to save your example.

### The response received from the server

After you’ve received a response from a server, you might want to save the current request and response pair as an example. Steps for doing so are similar to creating a new response from scratch.

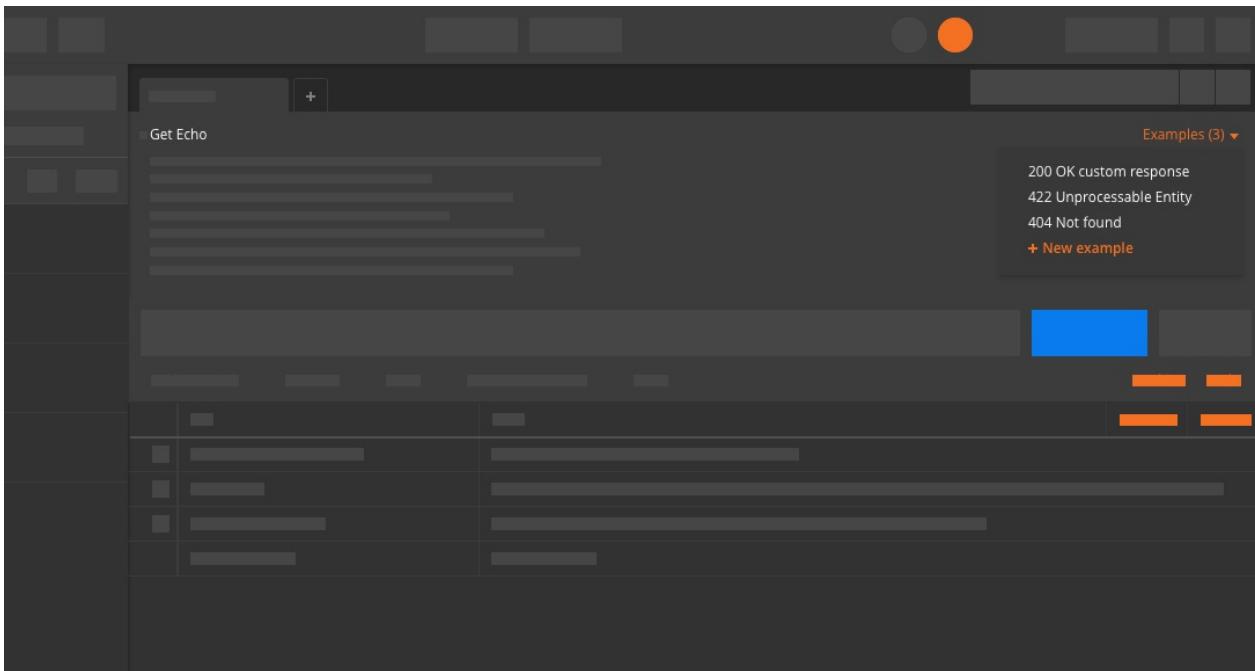


Later on, you can go back to your base request, and continue right where you left off by clicking on the request name in the upper left corner of the builder.



## Accessing your saved examples

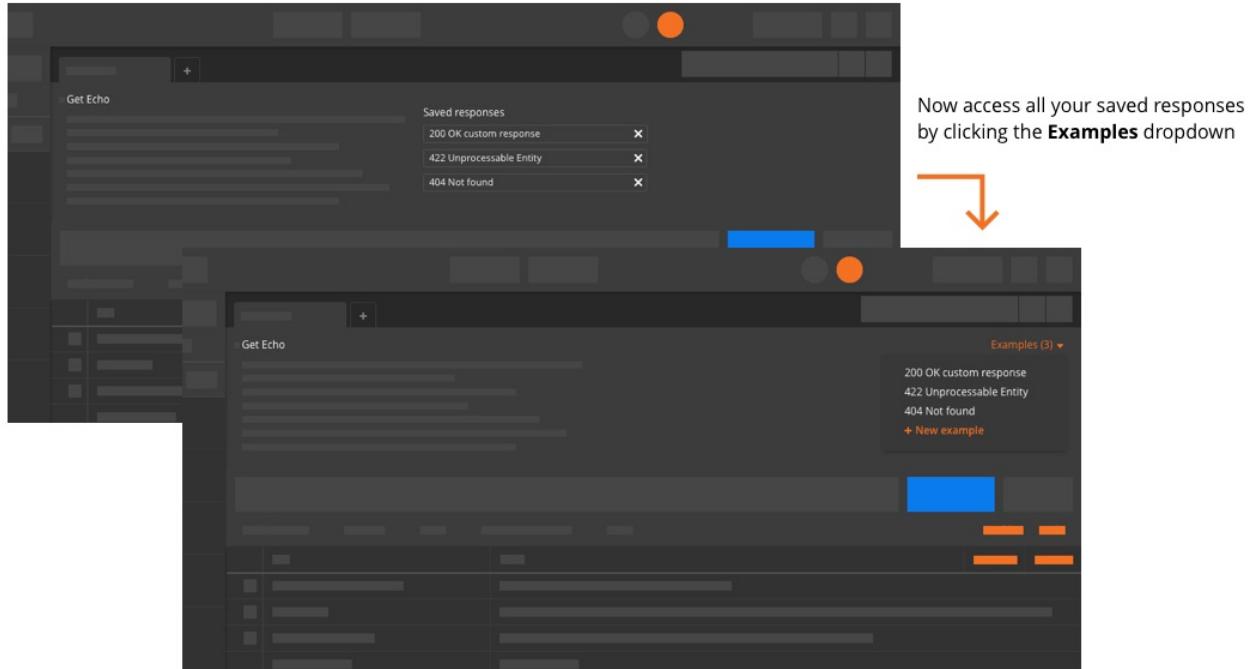
Click on the **Examples** dropdown in the upper right corner of the builder to access all your saved examples.



## What happened to the 'Save Response' feature?

The ability to [save responses](#) has been available in Postman for a long time. However, our users wanted to edit these responses before saving them, and also to add new responses. Examples make all of this possible!

Responses can be saved to examples. Save responses, like before, but now you can edit them whenever you want. Access your already saved responses by clicking on the **Examples** dropdown.



## How your examples appear in Postman documentation

You might already know that Postman has [API documentation](#), published to the web with a single click. Examples are displayed in your API documentation, providing additional details and clarification for your API.

And you can always go back and edit these examples, with real-time updates to the documentation!

This allows teams to mock an example request and response, along with simulating the endpoint using [mock servers](#). Front-end and back-end developers and testers can all begin working in parallel, based on the agreed-upon example.

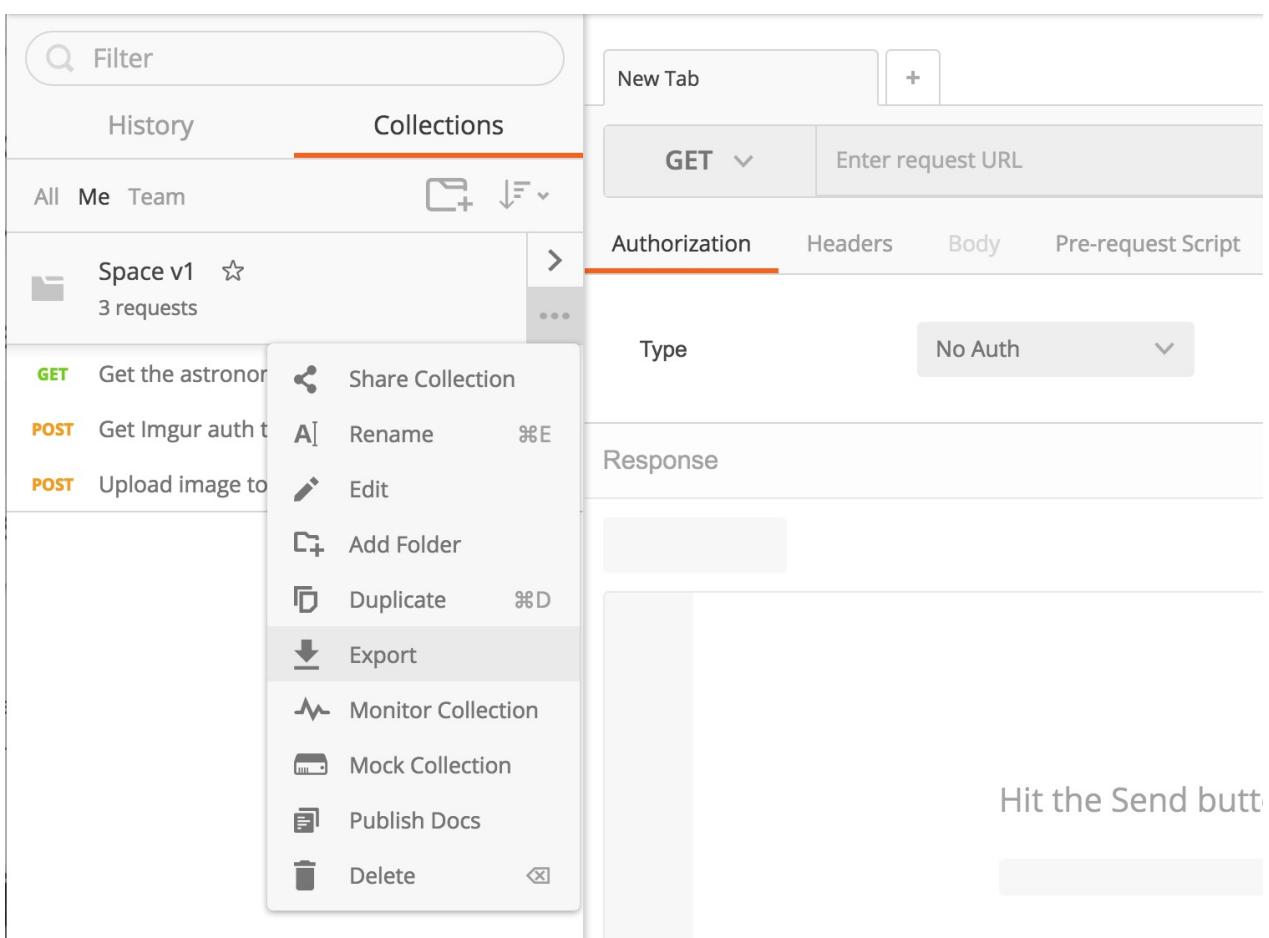
# Data formats

Postman can export and import collections, environments, globals and header presets as files and links.

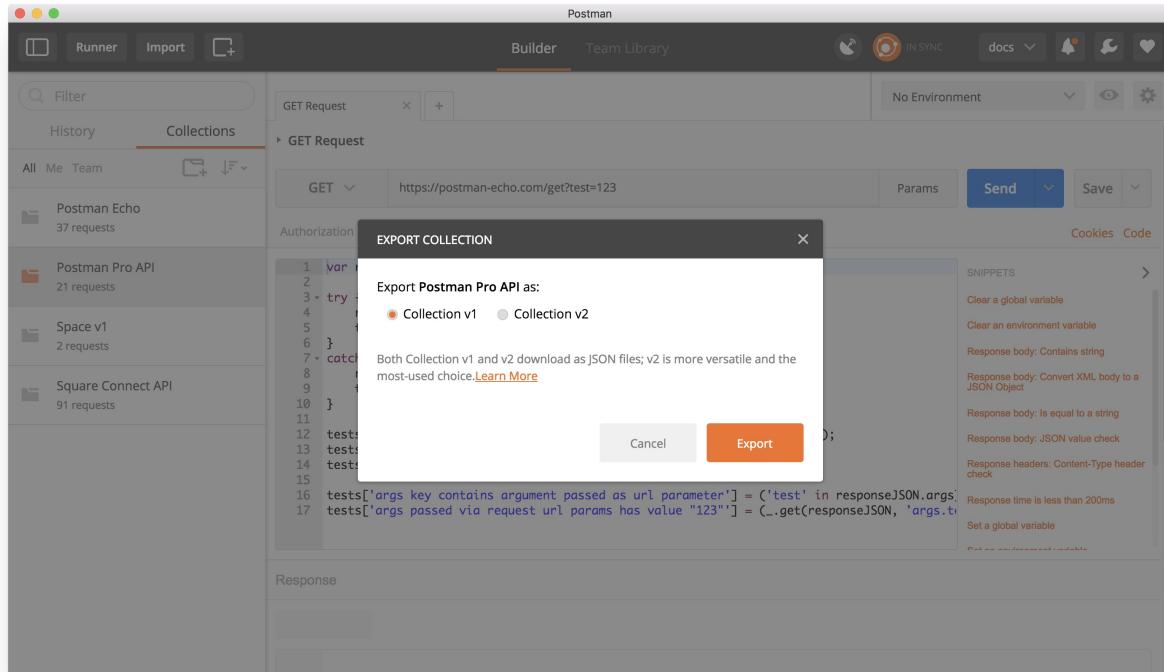
## Exporting and Importing Postman data

Postman can export and import the following formats as a file or generated URL. When you export a collection from the Postman app, the exported file is a JSON file. The file contains all data (and metadata) that is required by Postman to recreate the collection when imported back into Postman, or that is utilized by Newman to run the collection from the command line interface (CLI).

### Collections

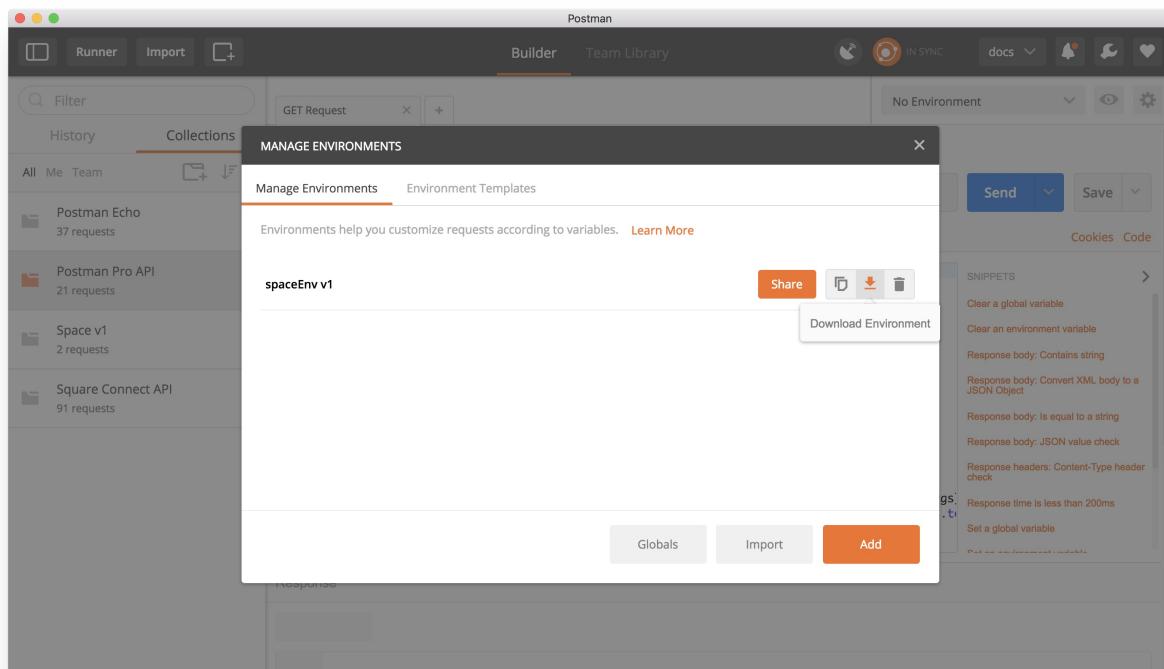


Postman can export collections in two formats - v1 and v2. Both Collection v1 and v2 download as JSON files; v2 is more versatile and the most-used choice. Learn more about the [v1 and v2 formats](#).

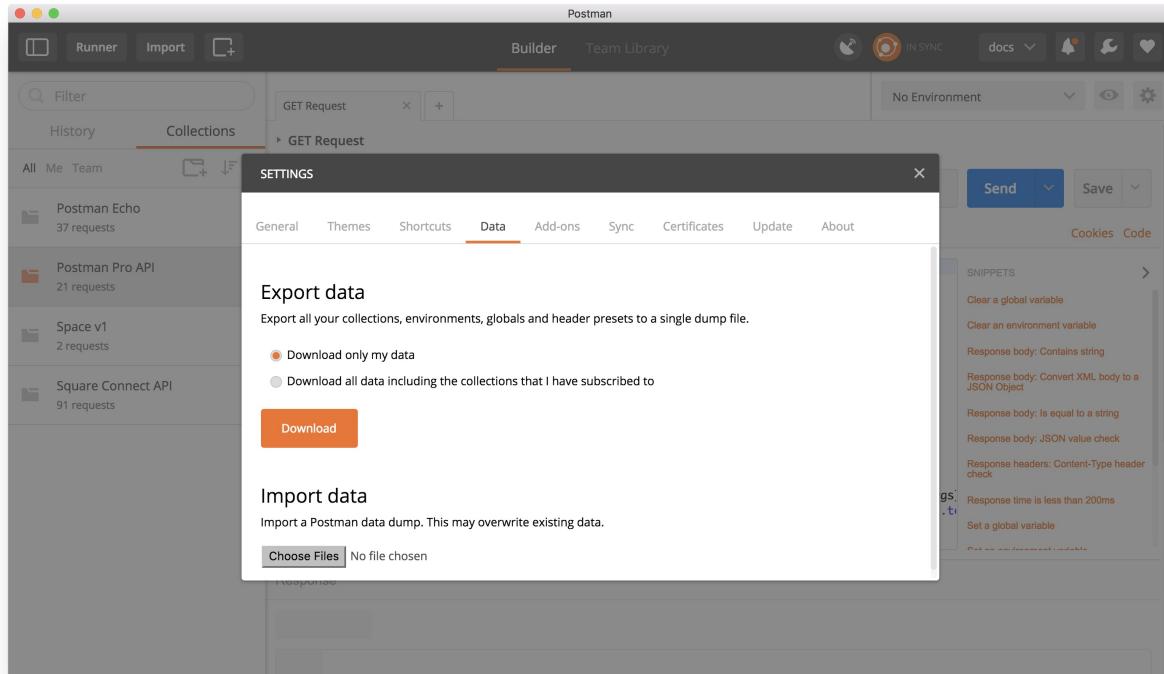


## Environments

Environments can be exported from the **MANAGE ENVIRONMENTS** modal, and imported here as well.



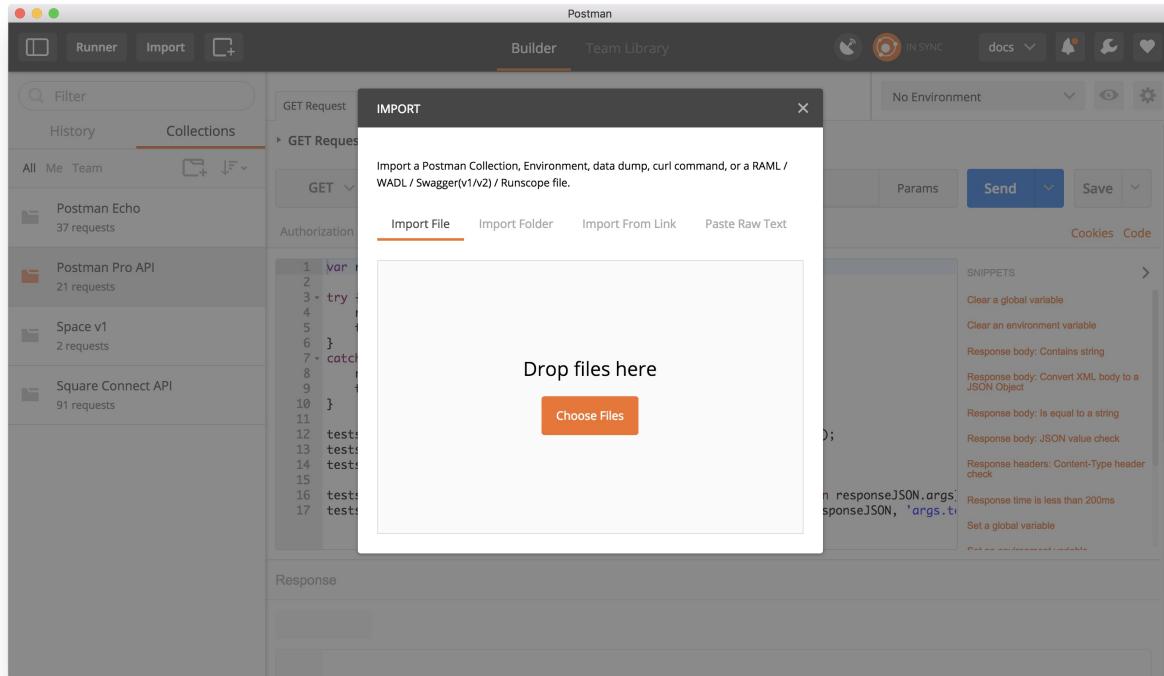
## Data dumps



From the **Data** tab of the **SETTINGS** modal, Postman allows you to export all collections, environments, globals and header presets into one JSON file. Postman does not export your history. You can import this data back into Postman.

## Importing Postman data

Postman data can be imported from the **Data** tab of the **SETTINGS** modal, or using the **Import** button in the header toolbar. Import a collection, environment, data dump, curl command, or a RAML / WADL / Swagger (v1/v2) / Runscope file using the **IMPORT** modal.



## Importing cURL

Most valid cURL (HTTP-only) commands can be imported into Postman. Postman's importer supports the following cURL options:

**Option\*\*Description\*\***

- A, --user-agentAn optional user-agent string
- d, --dataSends the specified data to the server with type application/x-www-form-urlencoded
- data-asciiSends the specified data to the server with type application/x-www-form-urlencoded
- data-urlencodeSends the specified data to the server with type application/x-www-form-urlencoded
- data-binaryData sent as-is
- F, --form A single form-data field (can be used multiple times)
- G, --getForces the request to be sent as GET, with the --data parameters appended to the query string
- H, --headerAdd a header (can be used multiple times)
- X, --requestSpecify a custom request method to be used
- urlAn alternate way to specify the URL

A few commands which can be imported include:

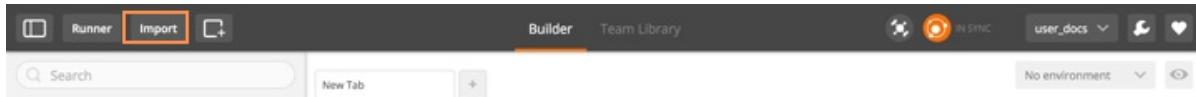
**cURL\*\*Effect\*\***

- curl <http://postman-echo.com/get>Creates a GET request in Postman with the URL prefilled
- curl --request POST --url <http://postman-echo.com/post> --form color=red --form color=greenCreates a POST request with a multivalue form data row
- curl -X PUT --data-binary hello <http://postman-echo.com/put>Creates a POST request with raw data
- curl -X PUT --data-ascii 'a=b&c=d' <http://postman-echo.com/put> -H 'AccessToken:1234'Creates a PUT request with urlencoded form data, and a custom header

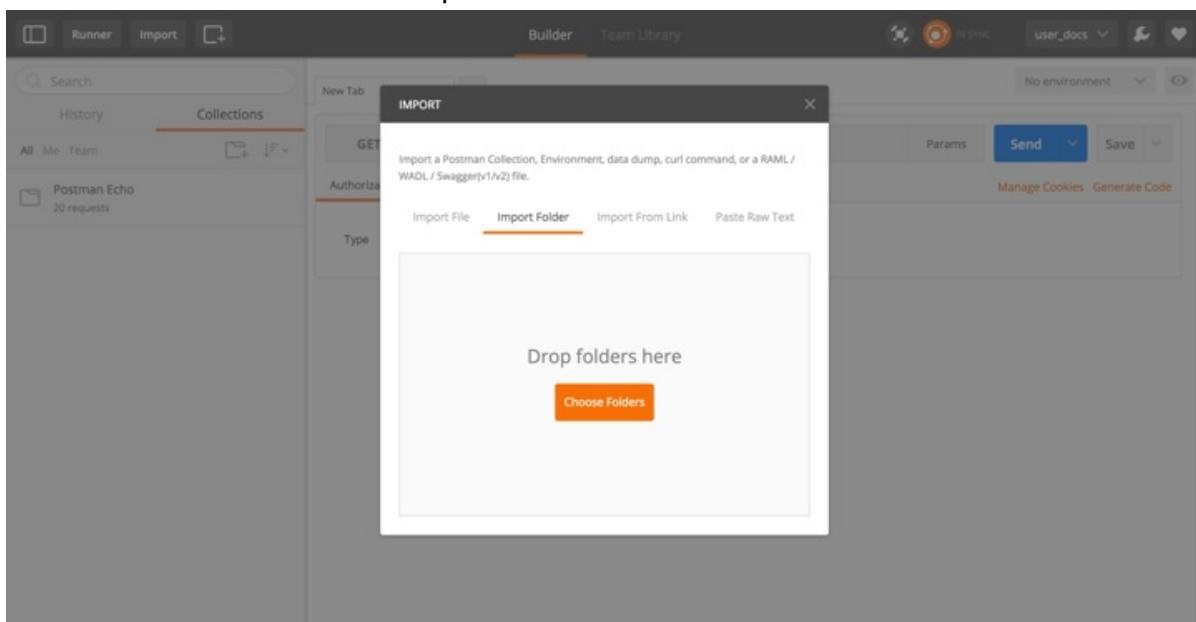
## Importing RAML

## Saving a RAML folder as a collection

1. Clone the repository containing the RAML definition to your local machine, or save it locally as a folder.
2. Click on the Import button, and choose the Import Folder tab.



3. Click on Choose Folders and upload the RAML folder.



You're done! Postman will detect all the RAML definitions and convert them internally to Postman and then show you an import success message.



## Examples

Download an example RAML file: [github-api-v3.raml](#)

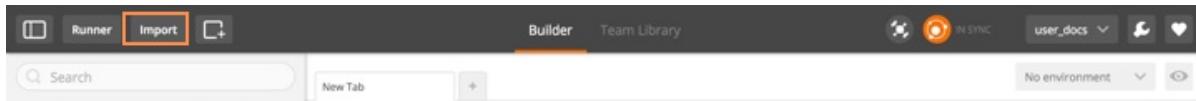
## Importing Swagger

A Swagger API definition usually lives as a single file, so we only support imports of single swagger files. If you have a lot of unrelated Swagger files in a folder, you can import those through the folder importer.

### Saving a Swagger file as a collection

1. Clone the repository containing the Swagger definition to your local machine. If you have it saved locally as file already, that's fine of course.
2. Click on the Import button, and choose the Import File tab. If you have a lot of unrelated

Swagger files in a folder, you can import those through the folder importer.



3. Click on file and upload the Swagger file.

You're done! Postman will detect all the Swagger definitions and convert them internally to Postman and then show you an import success message.



## Examples

Swagger 2.0: <https://github.com/OAI/OpenAPI-Specification/tree/master/examples/v2.0>

Swagger 1.2: <https://github.com/OAI/OpenAPI-Specification/wiki>Hello-World-Sample>

## Importing WADL

Postman lets you import WADL specs too. While all aspects are not supported yet, you can expect the various parameters that Postman uses (collections, folder, requests, headers, request payloads) to be correctly generated. We're currently working on extending this feature.

### Example WADL file

```

<application xmlns="http://wadl.dev.java.net/2009/02">
  <resources base="http://example.com/api">
    <resource path="books">
      <method name="GET"/>
      <resource path="{bookId}">
        <param required="true" style="template" name="bookId"/>
        <method name="GET"/>
        <method name="DELETE"/>
      <resource path="reviews">
        <method name="GET">
          <request>
            <param name="page" required="false" default="1" style="query"/>
            <param name="size" required="false" default="20" style="query"/>
          </request>
        </method>
      </resource>
    </resource>
    <resource path="readers">
      <method name="GET"/>
    </resource>
  </resources>
</application>

```

Taken from <http://www.nurkiewicz.com/2012/01/gentle-introduction-to-wadl-in-java.html>

## Validating Collection JSON files

To validate if a JSON file is in the correct collections format, you can use our [schema files for collections](#).

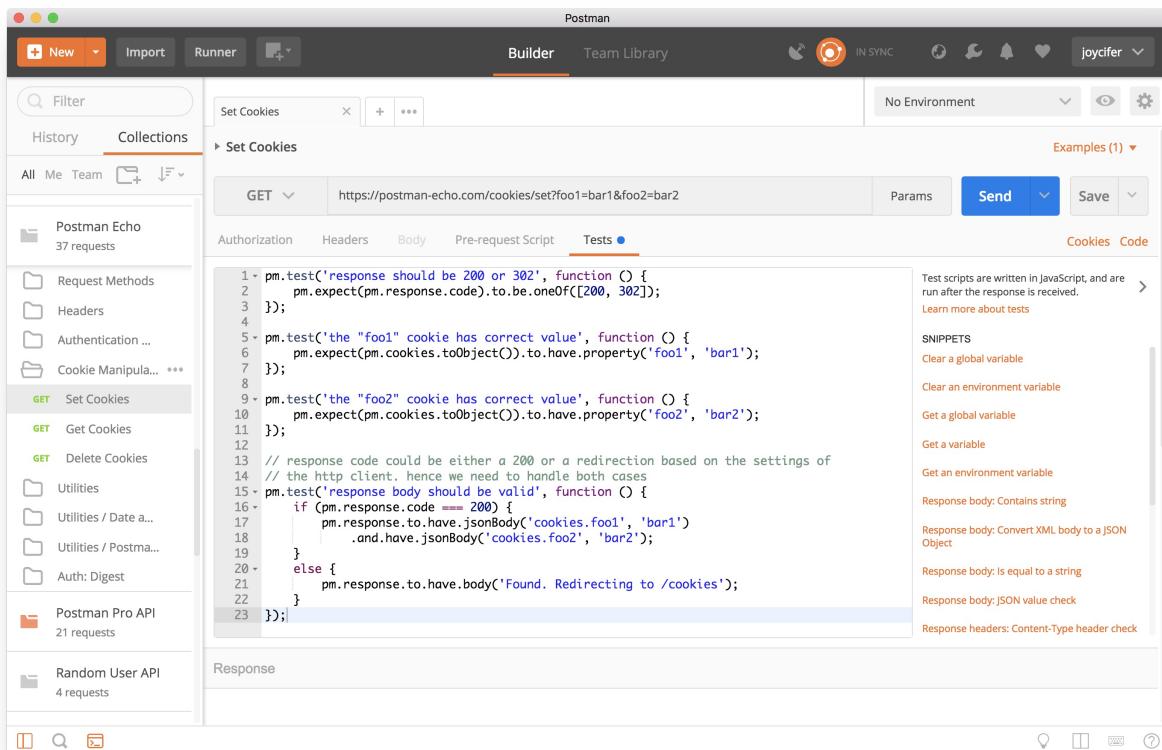
- The schema file is located at  
<http://schema.getpostman.com/json/collection/v1.0.0/collection.json>.
- The associated documentation can be found at  
<http://schema.getpostman.com/json/collection/v1.0.0/docs/index.html>.
- Everything is neatly stored on GitHub <https://github.com/postmanlabs/schemas>.
- To see an example of data validation using our schema and [is-my-json-valid](#) (a validator), check out [this blog post](#).

# 脚本介绍

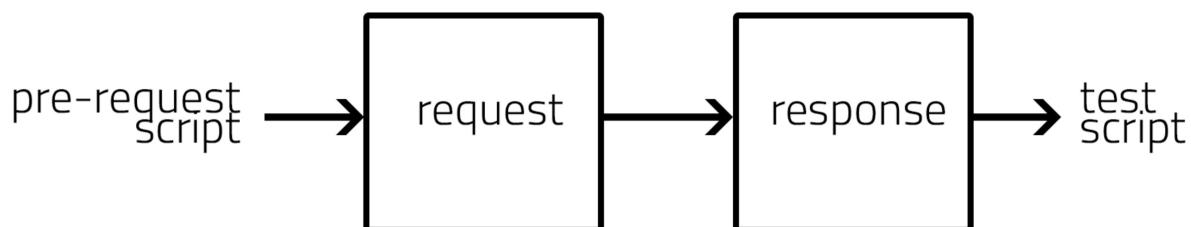
## Scripts in Postman

Postman包含一个基于Node.js的强大的运行时，它允许您向请求和集合添加动态行为。这样就可以编写测试套件，构建可以包含动态参数的请求，在请求之间传递数据等等。您可以在流程中的两个事件中添加要执行的JavaScript代码：

1. 在将请求发送到服务器之前，在 **Pre-request Script** 选项卡下的作为 **预请求脚本**。
2. 收到响应后，在 **Tests** 选项卡下的作为 **测试脚本**。



Postman中单个请求的执行流程如下所示：



怎么用

这是魔术？不，这是 [Postman Sandbox](#)。Postman Sandbox is a JavaScript 执行环境，你可以给每个请求添加预请求和测试脚本 (Postman和Newman都支持)。你写的脚本都会在 Sandbox 中执行。

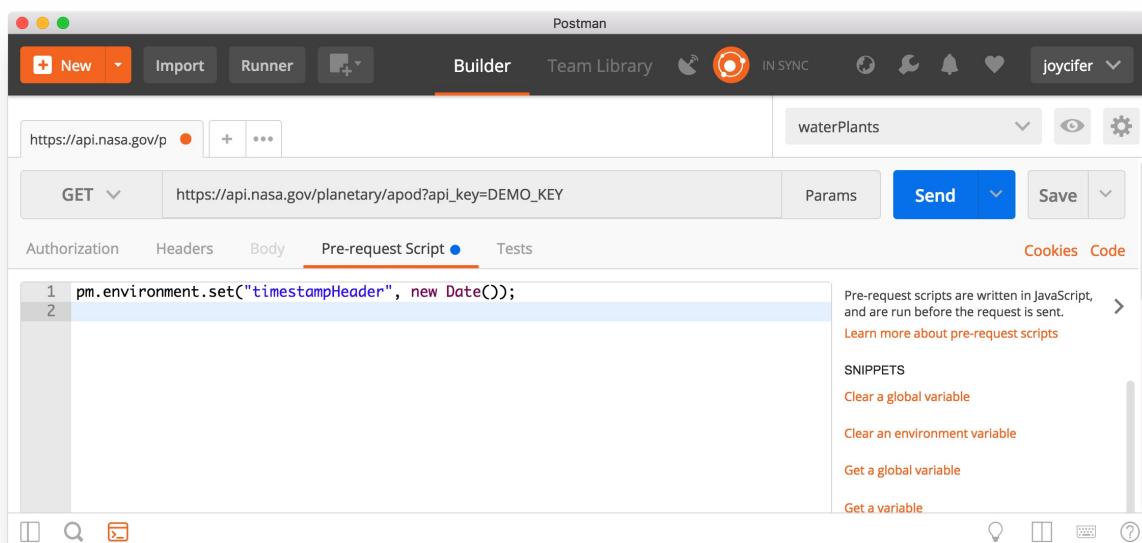
## 调试脚本

调试脚本可以在 **Pre-request Script** 和 **Tests** 选项卡下编写，并在 [Postman Console](#) 记录有用的信息。

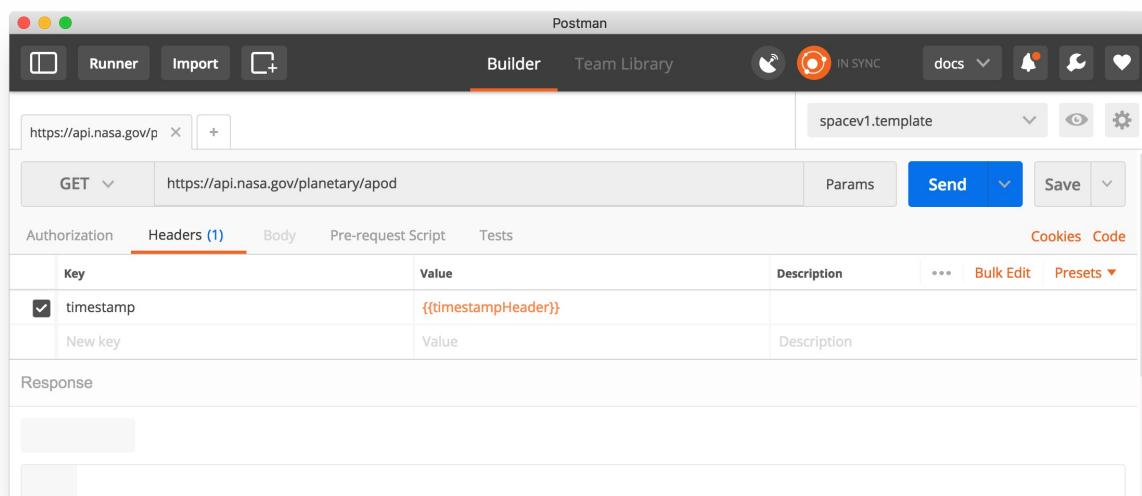
## 预请求脚本 (Pre-request)

Pre-request 是在请求发送前执行的代码片段。像在请求头中包含时间戳或在 URL 参数中发送随机的字母数字字符串等类似请求中非常适用。

例如，要在请求头中包含时间戳，可以使用从函数返回的值来设置环境变量。



你可以键入 `{{timestampHeader}}` 来访问头数据中的 **timestampHeader** 变量。当请求发送时，你的预请求脚本将被执行，并且 **timestampHeader** 的将取代 `{{timestampHeader}}`。



注意：要设置环境变量，该环境必须处于活动状态。

预请求脚本使用 **JavaScript** 编写，语法与 [测试脚本](#) 完全相同，但不存在响应对象。



# 测试脚本 (Test script)

在 Postman 中，您可以使用 JavaScript 语言为每个请求编写和运行测试。

The screenshot shows the Postman interface with a test script for a GET request to `https://randomuser.me/api?gender=male&nat=us`. The test script contains three assertions:

```

1 - pm.test("A single user was returned", function () {
2     pm.expect(pm.response.json().results).to.have.lengthOf(1);
3 });
4
5 // Gender tests
6 - pm.test("Gender is male", function () {
7     pm.expect(pm.response.json().results[0].gender).to.equal("male");
8     pm.expect(pm.response.json().results[0].name.title).to.equal("mr");
9 });
10
11 // Nationality test
12 - pm.test("The user is from the United States", function () {
13     pm.expect(pm.response.json().results[0].nat).to.equal("US");
14 });
15

```

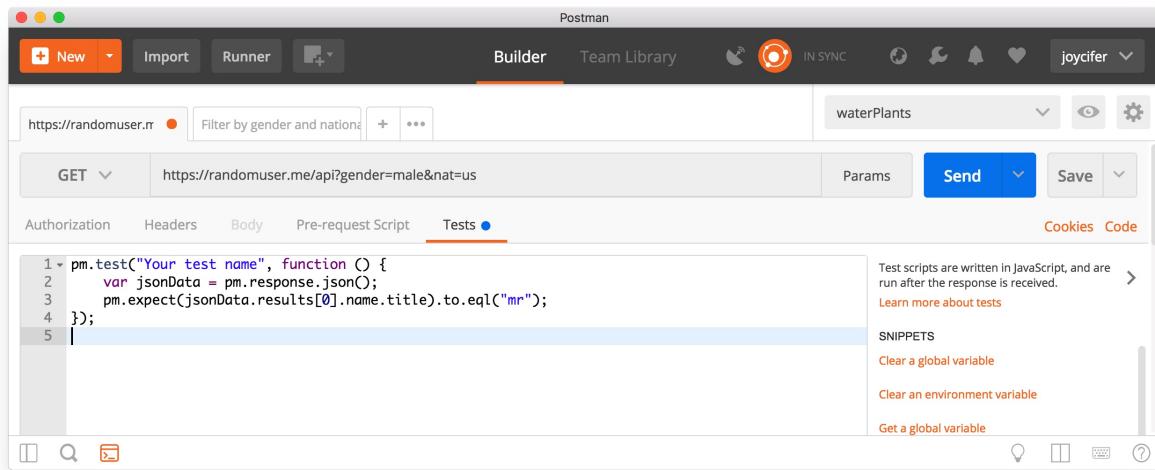
The test results section shows 3/3 tests passed:

- PASS A single user was returned
- PASS Gender is male
- PASS The user is from the United States

Postman status: Status: 200 OK Time: 201 ms Size: 1.23 KB

## 编写 Postman 测试

Postman 测试本质上是在发送请求后执行的 JavaScript 代码，允许访问 `pm.response` 对象。



这里有些例子：

```
// pm.response.to.have 示例
pm.test("response is ok", function () {
    pm.response.to.have.status(200);
});

// pm.expect() 示例
pm.test("environment to be production", function () {
    pm.expect(pm.environment.get("env")).to.equal("production");
});

// response assertions 示例
pm.test("response should be okay to process", function () {
    pm.response.to.not.be.error;
    pm.response.to.have.jsonBody("");
    pm.response.to.not.have.jsonBody("error");
});

// pm.response.to.be* 示例
pm.test("response must be valid and have a body", function () {
    // assert that the status code is 200
    pm.response.to.be.ok; // info, success, redirection, clientError, serverError, a
    re other variants
    // assert that the response has a valid JSON body
    pm.response.to.be.withBody;
    pm.response.to.be.json; // this assertion also checks if a body exists, so the a
    bove check is not needed
});
```

您可以根据需要添加任意数量的测试，具体取决于您要测试的内容。[更多 Postman 测试的例子](#)。

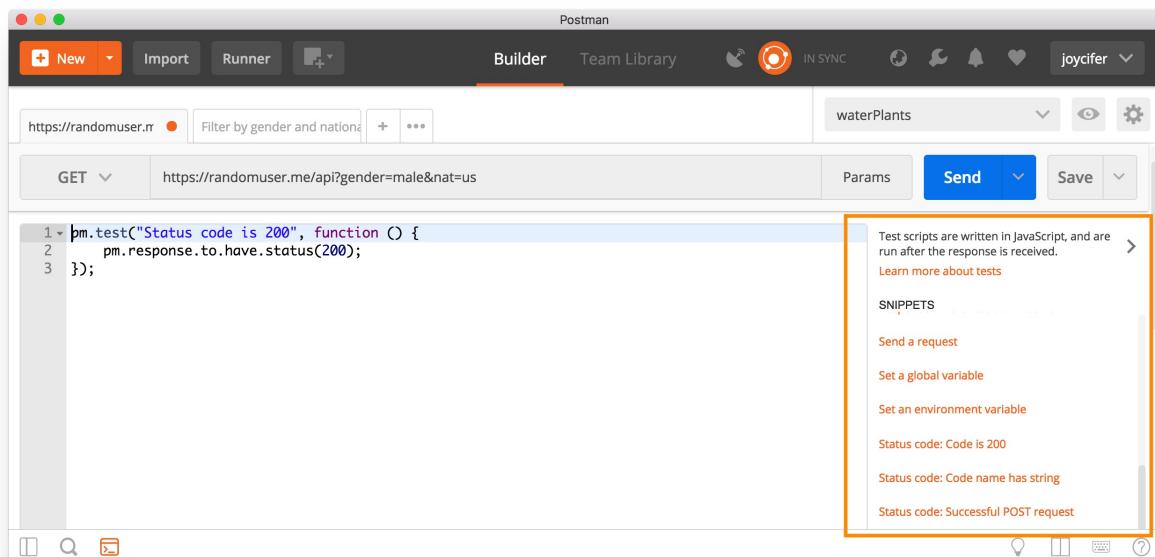
测试作为请求的一部分被保存，这对于前端或者后端开发人员确保一切正常运行都是很有用的。不用再眯着眼睛试图在代码中弄清楚出了什么问题。

## Sandbox

Postman 测试是在 **Sandbox** 环境中运行的，这与应用程序的执行环境是分开的。想要查看 **Sandbox** 中可用内容，请查看 [Sandbox 文档](#)。

## 脚本片段

虽然在编写测试时几乎没有什幺要记住的事情，Postman 试图通过在编辑器旁边列出常用的代码段来简化过程。您可以选择要添加的代码片段，相应的代码将添加到测试编辑器中。这是快速构建测试用例的好方法。



## 查看结果

Test Result	Description
PASS	Body contains cookies
PASS	Body contains cookie foo1
PASS	Body contains cookie foo2
PASS	Status code is 200

Postman 每次运行请求时都会运行测试。当然，您可以选择不查看测试结果！

结果显示在响应查看器下的 **Tests** 选项卡中。选项卡标题显示了通过的测试数量，测试结果也在下方列出。如果测试评估为 `true`，测试通过。



# 测试示例

测试脚本在请求发送并从服务器收到响应后运行。

我们来看一些 Postman 测试的例子。大多数示例在 Postman 代码片段中能找到。你可以根据需要编写多个测试。

## 设置环境变量

```
pm.environment.set("variable_key", "variable_value");
```

## 将嵌套对象设置为环境变量

```
var array = [1, 2, 3, 4];
pm.environment.set("array", JSON.stringify(array, null, 2));

var obj = { a: [1, 2, 3, 4], b: { c: 'val' } };
pm.environment.set("obj", JSON.stringify(obj));
```

## 获取环境变量

```
pm.environment.get("variable_key");
```

## 获取一个环境变量（其值是一个 **JSON** 字符串）

```
// 如果数据来自未知来源，这些语句应该包装在一个 try-catch 块。
var array = JSON.parse(pm.environment.get("array"));
var obj = JSON.parse(pm.environment.get("obj"));
```

## 清除一个环境变量

```
pm.environment.unset("variable_key");
```

## 设置一个全局变量

```
pm.globals.set("variable_key", "variable_value");
```

## 获取全局变量

```
pm.globals.get("variable_key");
```

## 清除一个全局变量

```
pm.globals.unset("variable_key");
```

## 获取变量

此函数在全局变量和活动环境之中搜索变量。

```
pm.variables.get("variable_key");
```

## 检查响应体是否包含一个字符串

```
pm.test("Body matches string", function () {
    pm.expect(pm.response.text()).to.include("string_you_want_to_search");
});
```

## 检查响应体是否等于一个字符串

```
pm.test("Body is correct", function () {
    pm.response.to.have.body("response_body_string");
});
```

## 检查 JSON 值

```
pm.test("Your test name", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.value).to.eql(100);
});
```

## Content-Type 是否存在

```
pm.test("Content-Type is present", function () {
    pm.response.to.have.header("Content-Type");
});
```

## 响应时间是否小于**200ms**

```
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});
```

## 状态码是否是**200**

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

代码名称是否包含一个字符串

```
pm.test("Status code name has string", function () {
    pm.response.to.have.status("Created");
});
```

成功的**POST**请求状态代码

```
pm.test("Successful POST request", function () {
    pm.expect(pm.response.code).to.be.oneOf([201, 202]);
});
```

将 **TinyValidator** 用于 **JSON** 数据

```
var schema = {
  "items": {
    "type": "boolean"
  }
};
var data1 = [true, false];
var data2 = [true, 123];

pm.test('Schema is valid', function() {
  pm.expect(tv4.validate(data1, schema)).to.be.true;
  pm.expect(tv4.validate(data2, schema)).to.be.true;
});
```

解码**base64**编码数据

```
var intermediate,
    base64Content, // assume this has a base64 encoded value
    rawContent = base64Content.slice('data:application/octet-stream;base64,'.length);

intermediate = CryptoJS.enc.Base64.parse(base64Content); // CryptoJS is an inbuilt object, documented here: https://www.npmjs.com/package/crypto-js
pm.test('Contents are valid', function() {
  pm.expect(CryptoJS.enc.Utf8.stringify(intermediate)).to.be.true;
```

发送异步请求

此功能既可以作为预请求和测试脚本。

```
pm.sendRequest("https://postman-echo.com/get", function (err, response) {  
    console.log(response.json());  
});
```

## 将 XML 体转换为 JSON 对象

```
var jsonObject = xml2Json(responseBody);
```

## 示例数据文件

JSON文件由键/值对组成。

[下载 JSON 文件](#)

对于CSV文件，顶行需要包含变量名。

[下载 CSV 文件](#)

## 旧版本 Postman 测试

The older style of writing Postman tests relies on setting values for the special tests object. You can set a descriptive key for an element in the object and then say if it's true or false. For example, tests["Body contains user\_id"] = responseBody.has("user\_id"); will check whether the response body contains the user\_id string.

You can add as many keys as needed, depending on how many things you want to test for. Under the **Tests** tab under the response viewer, you can view your test results. The tab header shows how many tests passed, and the keys that you set in the tests variable are listed here. If the value evaluates to true, the test passed.

### Setting an environment variable

```
postman.setEnvironmentVariable("key", "value");
```

### Setting a nested object as an environment variable

```
var array = [1, 2, 3, 4];  
postman.setEnvironmentVariable("array", JSON.stringify(array, null, 2));  
  
var obj = { a: [1, 2, 3, 4], b: { c: 'val' } };  
postman.setEnvironmentVariable("obj", JSON.stringify(obj));
```

### Getting an environment variable

```
postman.getEnvironmentVariable("key");
```

## Getting an environment variable (whose value is a stringified object)

```
// These statements should be wrapped in a try-catch block if the data is coming from  
an unknown source.  
  
var array = JSON.parse(postman.getEnvironmentVariable("array"));  
var obj = JSON.parse(postman.getEnvironmentVariable("obj"));
```

## Clear an environment variable

```
postman.clearEnvironmentVariable("key");
```

## Set a global variable

```
postman.setGlobalVariable("key", "value");
```

## Get a global variable

```
postman.getGlobalVariable("key");
```

## Clear a global variable

```
postman.clearGlobalVariable("key");
```

## Check if response body contains a string

```
tests["Body matches string"] = responseBody.has("string_you_want_to_search");
```

## Convert XML body to a JSON object

```
var jsonObject = xml2Json(responseBody);
```

## Check if response body is equal to a string

```
tests["Body is correct"] = responseBody === "response_body_string";
```

## Check for a JSON value

```
var data = JSON.parse(responseBody);
tests["Your test name"] = data.value === 100;
```

## Content-Type is present (Case-insensitive checking)

```
tests["Content-Type is present"] = postman.getResponseHeader("Content-Type"); //Note:
the getResponseHeader() method returns the header value, if it exists.
```

## Content-Type is present (Case-sensitive)

```
tests["Content-Type is present"] = responseHeaders.hasOwnProperty("Content-Type");
```

## Response time is less than 200ms

```
tests["Response time is less than 200ms"] = responseTime < 200;
```

## Response time is within a specific range (lower bound inclusive, upper bound exclusive)

```
tests["Response time is acceptable"] = _.inRange(responseTime, 100, 1001); // _ is the
inbuilt Lodash v3.10.1 object, documented at https://lodash.com/docs/3.10.1
```

## Status code is 200

```
tests["Status code is 200"] = responseCode.code === 200;
```

## Code name contains a string

```
tests["Status code name has string"] = responseCode.name.has("Created");
```

## Successful POST request status code

```
tests["Successful POST request"] = responseCode.code === 201 || responseCode.code ===
202;
```

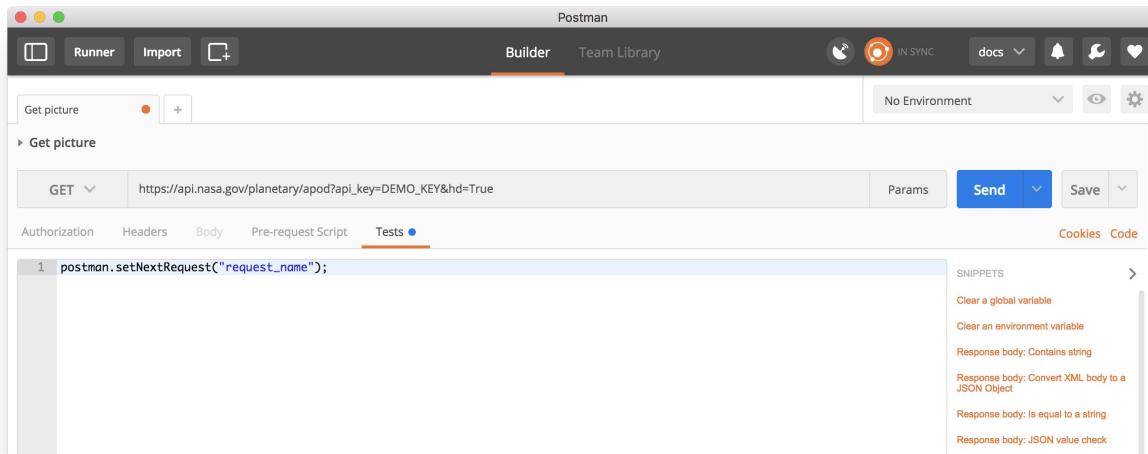
## Use TinyValidator for JSON data

```
var schema = {
  "items": {
    "type": "boolean"
```



# 分支和循环

运行集合时，您可以使用该 `postman.setNextRequest("request_name");` 功能在Postman中对 API请求进行分支和循环。



设置要接着执行的请求

```
postman.setNextRequest("request_name");
```

停止工作流

```
postman.setNextRequest(null);
```

一些关于 `postman.setNextRequest()` 的关键点：

1. 指定后续请求的名称或ID，集合运行器将处理其余请求。
2. 它可以在预请求或测试脚本中使用。在多次作业的情况下，后面的值将会覆盖前面的值。
3. 如果 `postman.setNextRequest()` 在请求中不存在，则处理器默认为线性执行并移动到下一个请求。

有关控制流程的更多信息，请参阅[构建工作流程](#)。

# Postman Sandbox

Postman Sandbox是一个 JavaScript 执行环境，在编写预请求脚本和测试脚本（在 Postman 和 Newman 中）时可用。在预请求/测试脚本部分中编写的代码都在 Sandbox 中执行。

## 常用的库和实用程序

- [Lodash](#): JS 实用程序库
- [cheerio](#): 快速，精简的核心 jQuery API 实现（版本4.6.0及更高版本）
- [BackboneJS](#) 已弃用: 提供简单的模型，视图和集合。这将在以后版本的沙箱中删除。
- [SugarJS](#) 已弃用: 使用有用的方法扩展本机JS对象。这将在以后版本的沙箱中删除。
- [tv4 JSON schema validator](#): 根据json-schema草案的v4验证JSON对象
- [CryptoJS](#): 标准和安全的加密算法。支持的算法：AES，DES，EvpKDF，HMAC-MD5，HMAC-SHA1 / 3/256/512，MD5，PBKDF2，Rabbit，SHA1 / 3/224/256/512，TripleDES
- `xml2Json(xmlString)` : 这个函数在 Newman 和 Postman 中是一样的
- `xmlToJson(xmlString)` 已弃用: 此功能在Newman和Postman中的行为不相同
- `postman.getResponseHeader(headerName)` Test-only : 返回名称为“headerName”的响应头（如果存在）。如果没有这样的头存在，则返回null。注意：根据W3C规范，头名不区分大小写。这个方法会照顾这个。`postman.getResponseHeader("Content-type")` 和 `postman.getResponseHeader("content-Type")` 将返回相同的值。

注意:自4.6.0版起，jQuery支持已经停用，支持[cheerio](#)。

## 环境和全局变量

- `postman.setEnvironmentVariable(variableName, variableValue)`: Sets an environment variable “variableName”，and assigns the string “variableValue” to it. You must have an environment selected for this method to work. **Note:** Only strings can be stored. Storing other types of data will result in unexpected behavior.
- `postman.getEnvironmentVariable(variableName)`: Returns the value of an environment variable “variableName”，for use in pre-request & test scripts. You must have an environment selected for this method to work.
- `postman.setGlobalVariable(variableName, variableValue)`: Sets a global variable “variableName”，and assigns the string “variableValue” to it. **Note:** Only strings can be stored. Storing other types of data will result in unexpected behavior.
- `postman.getGlobalVariable(variableName)`: Returns the value of a global variable “variableName”，for use in pre-request & test scripts.
- `postman.clearEnvironmentVariable(variableName)`: Clears the environment variable

named “variableName”. You must have an environment selected for this method to work.

- `postman.clearGlobalVariable(variableName)`: Clears the global variable named “variableName”.
- `postman.clearEnvironmentVariables()`: Clears all environment variables. You must have an environment selected for this method to work.
- `postman.clearGlobalVariables()`: Clears all global variables.
- `environment`: A dictionary of variables in the current environment. Use `environment["foo"]` to access the value of the “foo” environment variable. **Note:** This can only be used to read the variable. Use `setEnvironmentVariable()` to set a value.
- `globals`: A dictionary of global variables. Use `globals["bar"]` to access the value of the “bar” global variable. **Note:** This can only be used to read the variable. Use `setGlobalVariable()` to set a value

## Dynamic variables

Postman also has a few dynamic variables which you can use in your requests. This is primarily an experiment right now. More functions would be added soon. Note that dynamic variables cannot be used in the Sandbox. You can only use them in the `{{..}}` format in the request URL / headers / body.

- `:v4` : Adds a v4 style guid
- `:now` : Adds the current timestamp.
- `:randint` : Adds a random integer between 0 and 1000

## Cookies

- `responseCookies {array}` Postman-only: Gets all cookies set for the domain. You will need to enable the [Interceptor](#) for this to work.
- `postman.getResponseCookie(cookieName)` Postman-only: Gets the response cookie with the given name. You will need to enable the interceptor for this to work. Check out the [blog post](#).

## Request/response related properties

- `request {object}`: Postman makes the request object available to you while writing scripts. This object is read-only. Changing properties of this object will have no effect.  
Note: Variables will NOT be resolved in the request object. The request object is composed of the following:
  - `data {object}` - this is a dictionary of form data for the request.

- (request.data["key"]=="value")
  - headers {object} - this is a dictionary of headers for the request  
(request.headers["key"]=="value")
  - method {string} - GET/POST/PUT etc.
  - url {string} - the url for the request.
- responseHeaders {object} **Deprecated, Test-only:** This is a map of the response headers. This is case-sensitive, and should not be used. Check the postman.getResponseHeader() method listed above.
- responseBody {string} **Test-only:** A string containing the raw response body text. You can use this as an input to JSON.parse, or xml2Json.
- responseTime {number} **Test-only:** The response time in milliseconds
- responseCode {object} **Test-only:** Contains three properties:
  - code {number}: The response code (200 for OK, 404 for Not Found etc)
  - name {string}: The status code text
  - detail {string}: An explanation of the response code
- tests {object} **Test-only:** This object is for you to populate. Postman will treat each property of this object as a boolean test.
- iteration {number}: Only available in the Collection Runner and Newman. Represents the current test run index. Starts from 0.

**Test-only:** This object is only available in the test script section. Using this in a pre-request script will throw an error.

## Data files

If you're using [data files](#) in the Collection Runner or in Newman, you'll have access to a data object, which is a dictionary of data values in the current test run.

## pm.\* APIs

Review [Postman Sandbox API Reference](#).

# Postman Sandbox API reference

*Note: The functionality described here is exclusive to Postman's native apps for Mac, Windows, and Linux.*

## Global functions (pm.\* )

### require

**require(moduleName:String):function → \***

The require function allows you to use the sandbox built-in library modules. The list of available libraries are listed below. The list links to their corresponding documentation.

1. [atob](#) → v2.0.3
2. [btoa](#) → v1.1.2
3. [chai](#) → v3.5.0
4. [cheerio](#) → v0.22.0
5. [crypto-js](#) → v3.1.9-1
6. [csv-parse/lib-sync](#) → 1.2.1
7. [lodash](#) → v4.17.2 (when used with require, the inbuilt \_ object is for v3.10.1)
8. [moment](#) → v2.18.1 (sans locales)
9. [postman-collection](#) → v1.2.0
10. [tv4](#) → v1.2.7
11. [uuid](#) → (the module loaded is a shim for original module)
12. [xml2js](#) → 0.4.19

A number of NodeJS modules are also available:

1. path
2. assert
3. buffer
4. util
5. url
6. punycode
7. querystring
8. string\_decoder
9. stream
10. timers
11. events

In order to use a library, simply call the require function and pass the module name as a parameter and assign the return of the function to a variable.

```
var atob = require('atob'),
  _ = require('lodash'),

arrayOfStrings,
base64Strings;

arrayOfStrings =  = ['string1', 'string2'];

base64Strings = _.map(arrayOfStrings, atob);

console.log(base64Strings);
```

## pm

### pm:Object

The pm object encloses all information pertaining to the script being executed and allows one to access a copy of the request being sent or the response received. It also allows one to get and set environment and global variables.

#### pm.info:Object

The pm.info object contains information pertaining to the script being executed. Useful information such as the request name, request Id, and iteration count are stored inside of this object.

- pm.info.eventName:String Contains information whether the script being executed is a “prerequest” or a “test” script.
- pm.info.iteration:Number Is the value of the current iteration being run.
- pm.info.iterationCount:Number Is the total number of iterations that are scheduled to run.
- pm.info.requestName:String
- pm.info.requestId:String

## pm.sendRequest

### pm.sendRequest:Function

The pm.sendRequest function allows sending HTTP/HTTPS requests asynchronously. Simply put, with asynchronous scripts, you can now execute logic in the background if you have a heavy computational task or are sending multiple requests. Instead of waiting for a

call to complete and blocking any next requests, you can designate a callback function and be notified when the underlying operation has finished.

Some things to know about pm.sendRequest():

- The method accepts a collection SDK compliant request and a callback. The callback receives 2 arguments, an error (if any) and SDK compliant response.
- It can be used in the pre-request or the test script.

```
// example with a plain string URL
pm.sendRequest('https://postman-echo.com/get', function (err, res) {
  if (err) {
    console.log(err);
  } else {
    pm.environment.set("variable_key", "new_value");
  }
});

// Example with a full fledged SDK Request
const echoPostRequest = {
  url: 'https://postman-echo.com/post',
  method: 'POST',
  header: 'headername1:value1',
  body: {
    mode: 'raw',
    raw: JSON.stringify({ key: 'this is json' })
  }
};
pm.sendRequest(echoPostRequest, function (err, res) {
  console.log(err ? err : res.json());
});

// example containing a test ** under the Tests tab only
pm.sendRequest('https://postman-echo.com/get', function (err, res) {
  if (err) { console.log(err); }
  pm.test('response should be okay to process', function () {
    pm.expect(err).to.equal(null);
    pm.expect(res).to.have.property('code', 200);
    pm.expect(res).to.have.property('status', 'OK');
  });
});
```

Extended Reference:

- [Request JSON](#)
- [Response Structure](#)

## pm.globals

## pm.globals:VariableScope

- pm.globals.has(variableName:String):function → Boolean
- pm.globals.get(variableName:String):function → \*
- pm.globals.set(variableName:String, variableValue:String):function
- pm.globals.unset(variableName:String):function
- pm.globals.clear():function
- pm.globals.toObject():function → Object

## pm.environment

s", sans-serif; font-weight: 600; line-height: 50px; color: rgb(40, 40, 40); margin-top: 20px; margin-bottom: 10px; font-size: 30px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; letter-spacing: normal; orphans: 2; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-style: initial; text-decoration-color: initial;">Postman Sandbox API reference

*Note: The functionality described here is exclusive to Postman's native apps for Mac, Windows, and Linux.*

## Global functions (pm.\* )

### require

#### require(moduleName:String):function → \*

The require function allows you to use the sandbox built-in library modules. The list of available libraries are listed below. The list links to their corresponding documentation.

1. [atob](#) → v2.0.3
2. [btoa](#) → v1.1.2
3. [chai](#) → v3.5.0
4. [cheerio](#) → v0.22.0
5. [crypto-js](#) → v3.1.9-1
6. [csv-parse/lib-sync](#) → 1.2.1
7. [lodash](#) → v4.17.2 (when used with require, the inbuilt \_ object is for v3.10.1)
8. [moment](#) → v2.18.1 (sans locales)
9. [postman-collection](#) → v1.2.0
10. [tv4](#) → v1.2.7
11. [uuid](#) → (the module loaded is a shim for original module)
12. [xml2js](#) → 0.4.19

A number of NodeJS modules are also available:

1. path
2. assert
3. buffer
4. util
5. url
6. punycode
7. querystring
8. string\_decoder
9. stream
10. timers
11. events

In order to use a library, simply call the require function and pass the module name as a parameter and assign the return of the function to a variable.

```
var atob = require('atob'),
    _ = require('lodash'),

arrayOfStrings,
base64Strings;

arrayOfStrings =  ['string1', 'string2'];

base64Strings = _.map(arrayOfStrings, atob);

console.log(base64Strings);
```

## pm

### pm:Object

The pm object encloses all information pertaining to the script being executed and allows one to access a copy of the request being sent or the response received. It also allows one to get and set environment and global variables.

#### pm.info:Object

The pm.info object contains information pertaining to the script being executed. Useful information such as the request name, request Id, and iteration count are stored inside of this object.

- pm.info.eventName:String Contains information whether the script being executed is a “prerequest” or a “test” script.
- pm.info.iteration:Number Is the value of the current iteration being run.
- pm.info.iterationCount:Number Is the total number of iterations that are scheduled to

run.

- pm.info.requestName:String
- pm.info.requestId:String

## pm.sendRequest

### pm.sendRequest:Function

The pm.sendRequest function allows sending HTTP/HTTPS requests asynchronously. Simply put, with asynchronous scripts, you can now execute logic in the background if you have a heavy computational task or are sending multiple requests. Instead of waiting for a call to complete and blocking any next requests, you can designate a callback function and be notified when the underlying operation has finished.

Some things to know about pm.sendRequest():

- The method accepts a collection SDK compliant request and a callback. The callback receives 2 arguments, an error (if any) and SDK compliant response.
- It can be used in the pre-request or the test script.

```

// example with a plain string URL
pm.sendRequest('https://postman-echo.com/get', function (err, res) {
  if (err) {
    console.log(err);
  } else {
    pm.environment.set("variable_key", "new_value");
  }
});

// Example with a full fledged SDK Request
const echoPostRequest = {
  url: 'https://postman-echo.com/post',
  method: 'POST',
  header: 'headername1:value1',
  body: {
    mode: 'raw',
    raw: JSON.stringify({ key: 'this is json' })
  }
};
pm.sendRequest(echoPostRequest, function (err, res) {
  console.log(err ? err : res.json());
});

// example containing a test ** under the Tests tab only
pm.sendRequest('https://postman-echo.com/get', function (err, res) {
  if (err) { console.log(err); }
  pm.test('response should be okay to process', function () {
    pm.expect(err).to.equal(null);
    pm.expect(res).to.have.property('code', 200);
    pm.expect(res).to.have.property('status', 'OK');
  });
});

```

## Extended Reference:

- [Request JSON](#)
- [Response Structure](#)

## **pm.globals**

### **pm.globals:VariableScope**

- [pm.globals.has\(variableName:String\):function → Boolean](#)
- [pm.globals.get\(variableName:String\):function → \\*](#)
- [pm.globals.set\(variableName:String, variableValue:String\):function](#)
- [pm.globals.unset\(variableName:String\):function](#)
- [pm.globals.clear\(\):function](#)
- [pm.globals.toObject\(\):function → Object](#)

## **pm.environment**

# Variables

## What are variables?

Variables are symbols that can take different values. You might be familiar with variables from other languages from your prior programming experience. Variables in Postman work the same way.

## Why use variables?

Variables allow you to reuse values in multiple places so you can keep your code DRY (Don't Repeat Yourself). Also, if you want to change the value, you can change the variable once with the impact cascading through the rest of your code.

Let's say you have 3 API endpoints that use the same domain - `your-domain.com`. You can save this domain as a variable and instead of repeating the value, you can use `/endpoint1` and `/endpoint2` in the request builder. Now, if your domain changes to another-`domain.com`, you just have to change this value once.

With Postman's scripting engine you can set variable values, copy data from one request and use it into another request, and more.

## Variable scopes

The following scopes are available to you:

1. Global
2. Environment
3. Local
4. Data

Scopes can be viewed as different kinds of buckets in which values reside. If a variable is in two different scopes, the scope with a higher priority wins and the variable gets its value from there. Postman resolves scopes using this hierarchy progressing from broad to narrow scope.

If a variable from the currently active environment shares its name with a global variable, the environment variable will take priority. In other words, global variables are overridden by environment variables, which are overridden by [data variables](#) (only available in the [collection runner](#)).

## Accessing variables in the request builder

Variables can be used in the following form in the Postman user interface - . The string will be replaced with its corresponding value when Postman resolves the variable. For example, for an environment variable ‘url’ with the value ‘<http://localhost>’ , you will have to use in the request URL field. will be replaced by <http://localhost> when the request is sent.

Since variables in the request builder are accessed using string substitution, they can be used everywhere in the request builder where you can add text. This includes the URL, URL parameters, headers, authorization, request body and header presets. Postman evaluates the variables according to scoping rules as discussed in the Variable Scopes section and sends them to the server.

## Accessing variables through scripts

Variables can also be used in pre-request and test scripts. Since these sections for scripts are written in JavaScript, you will initialize and retrieve these variables in a different manner. You can initialize variables in scripts and put them in a particular scope.

1. Defining a variable in a script:
  - To set a variable in a script, use the `setEnvironmentVariable()` method or `setGlobalVariable()` method depending on the desired scope. The method requires the variable key and value as parameters to set the variable. When you send the request, the script will be evaluated and the value will be stored as the variable.
2. Fetching a pre-defined variable:
  - Once a variable has been set, use the `getEnvironmentVariable()` method or `getGlobalVariable()` method depending on the appropriate scope to fetch the variable. The method requires the variable name as a parameter to retrieve the stored value in a script.
3. Setting a variable in a scope:
  - Environment variables can be accessed with the corresponding environment template. Global variables can be accessed broadly regardless of the selected environment.

## Logging variables

Often while using variables in scripts, you will need to see the values they obtain. You can use the [Postman Console](#) to do this easily. From the application menu, select “View” and then “Show Postman Console”. To log the value of a variable, you can use `console.log(foo);`

in your script. When you send a request, the script will be evaluated and the value of the variable will be logged in the Postman Console.

## Data variables

The Collection Runner lets you import a CSV or a JSON file, and then use the values from the data file inside HTTP requests and scripts. We call these data variables. To use them inside Postman, follow the same syntax as environment or global variables.

### Data variables in requests

Variables inside the Postman UI are enclosed inside curly braces. For example, in the screenshot below, and inside URL parameters would be replaced by corresponding values from the data file:

The screenshot shows the Postman Request tab with the following details:

- Method:** GET
- URL:** echo.getpostman.com/get?username={{username}}&password={{password}}
- Params:** An empty table with columns for key and value.
- Send:** A blue button.
- Save:** A dropdown menu.
- Authorization:** Set to No Auth.
- Headers:** None listed.
- Body:** None listed.
- Pre-request Script:** None listed.
- Tests:** None listed.
- Type:** No Auth.

### Data variables in pre-request and test scripts

Inside pre-request and test scripts, the special `data` object contains values loaded from the data file for a specific iteration. For example `data.username` or `data["username"]` would let you access the value of the `username` variable from a data file.

The screenshot shows the Postman Request tab with the following details:

- Method:** GET
- URL:** echo.getpostman.com/get?username={{username}}&password={{password}}
- Params:** An empty table with columns for key and value.
- Send:** A blue button.
- Save:** A dropdown menu.
- Authorization:** Set to No Auth.
- Headers:** None listed.
- Body:** None listed.
- Pre-request Script:** Contains the following code:
 

```
1 console.log("Username that will be sent: " + data.username);
2 console.log("Password that will be sent: " + data["password"]);
```
- Tests:** None listed.
- Type:** No Auth.

Learn more about [working with data files](#).

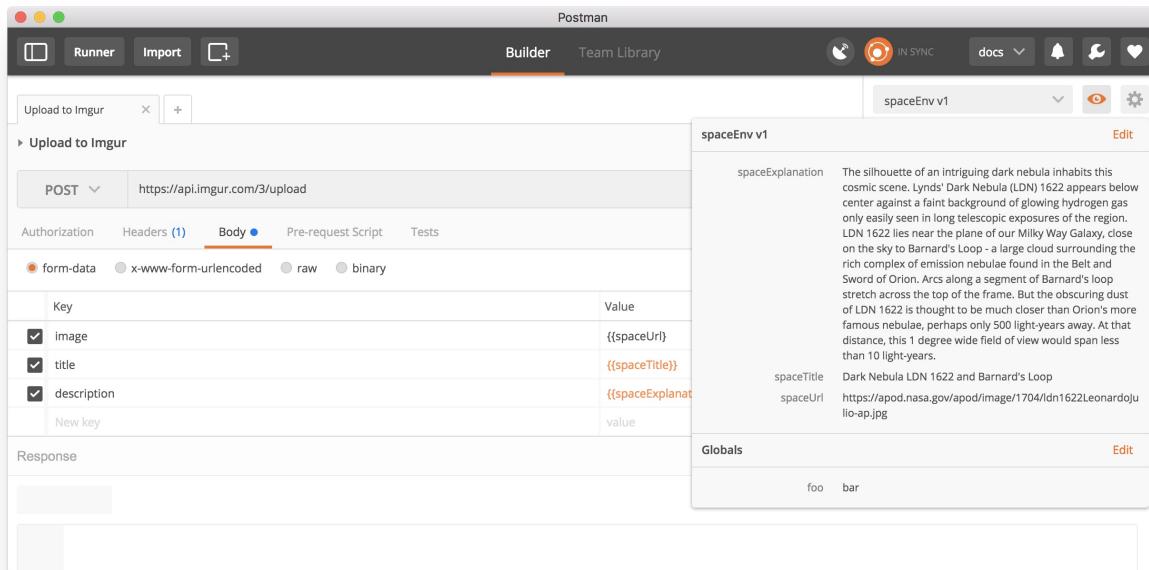
## Dynamic variables

Postman has a few dynamic variables which you can use in your requests. Dynamic variables cannot be used in the Sandbox. You can only use them in the `{{..}}` format in the request URL / headers / body.

- `:v4` : Adds a v4 style guid
- `:now` : Adds the current timestamp
- `:rand(1000)` : Adds a random integer between 0 and 1000

## Quick Look for variables

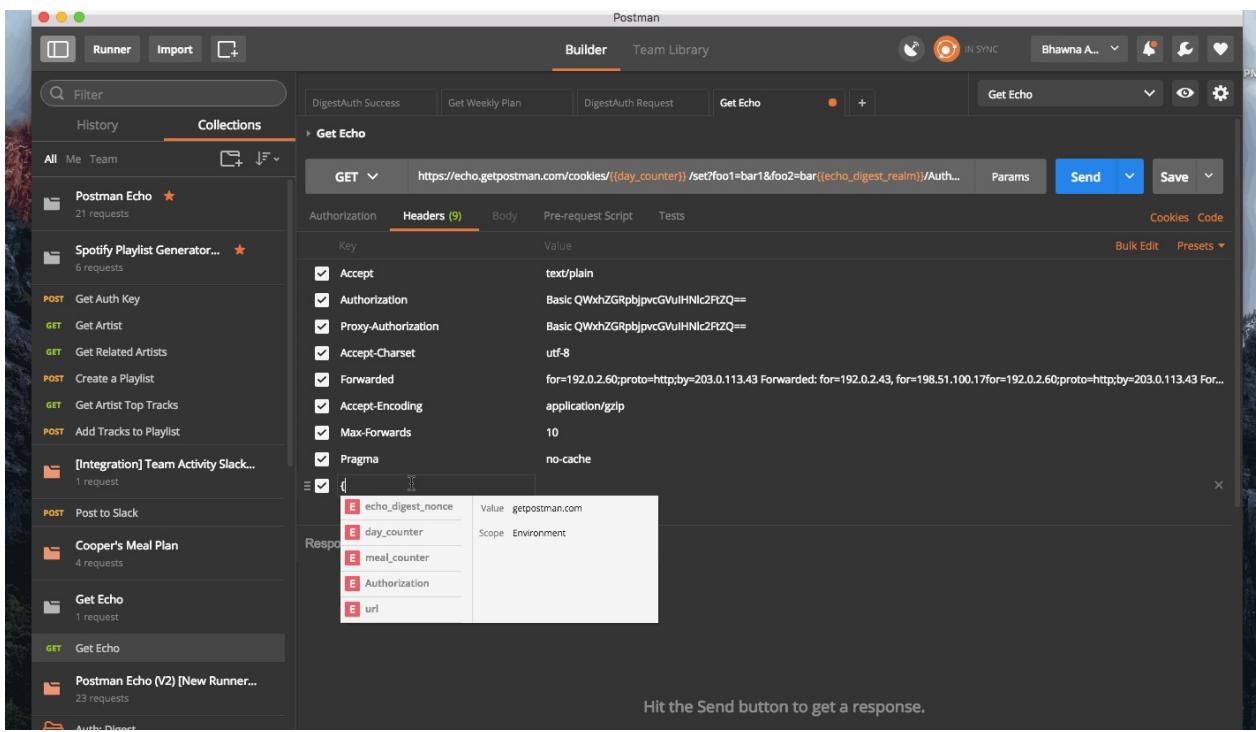
Quick Look is a quick preview feature displaying all your environment and global variables in one place. Click on the “eye” icon in the top right to toggle the display, or typing the keyboard shortcut (**CMD/CTRL + ALT + E**).



## Autocomplete and tooltips for variables

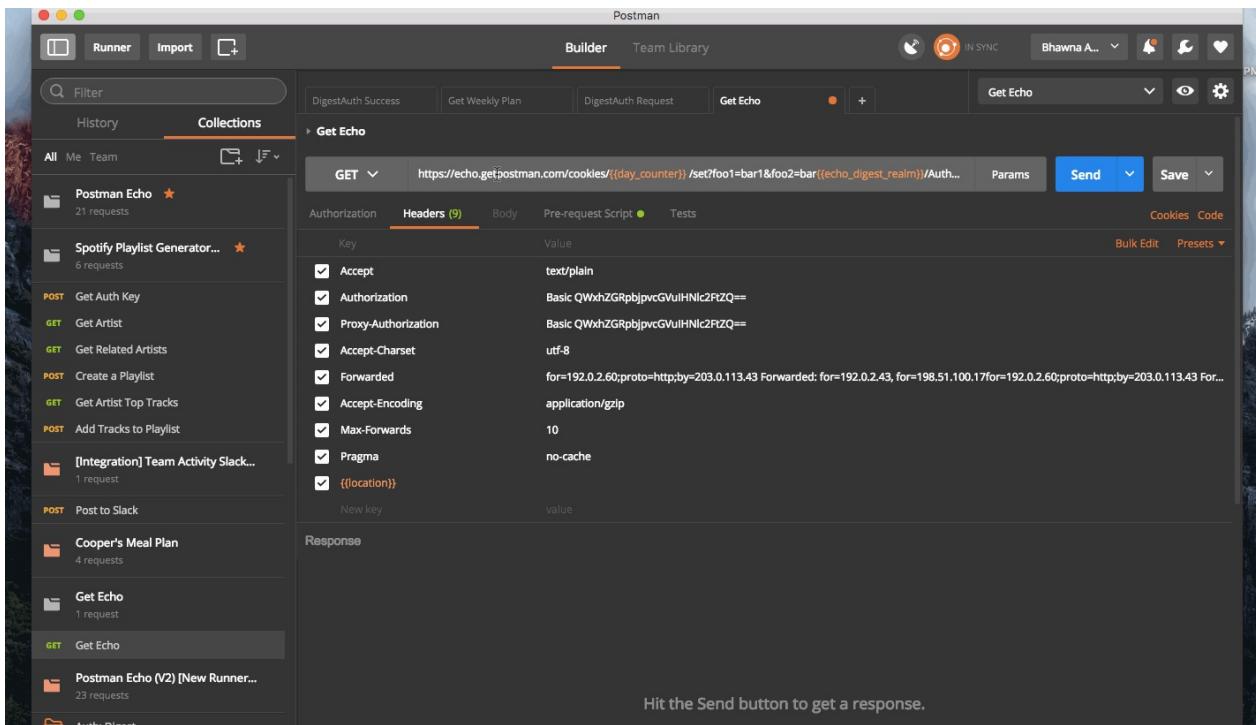
Postman variables are very powerful, and two features - autocomplete and tool tips - make them even more convenient.

### Autocomplete for variables



Type an open curly bracket to bring up the autocomplete menu. For the pre-request and test scripts section, which [uses the data editor](#), entering the first letter of a variable triggers the autocomplete. The menu contains a list of all the variables in the current environment, followed by globals. Navigating through the list also shows the current value and scope for each variable, along with a feedback for overridden variables.

## Variable highlighting and tooltip on hover



Variables are highlighted in orange, with unresolved variables shown in red colour. Hovering over a variable shows its current value and the scope. If a variable is unresolved - i.e., no value in the current environment - the tooltip shows the appropriate feedback.

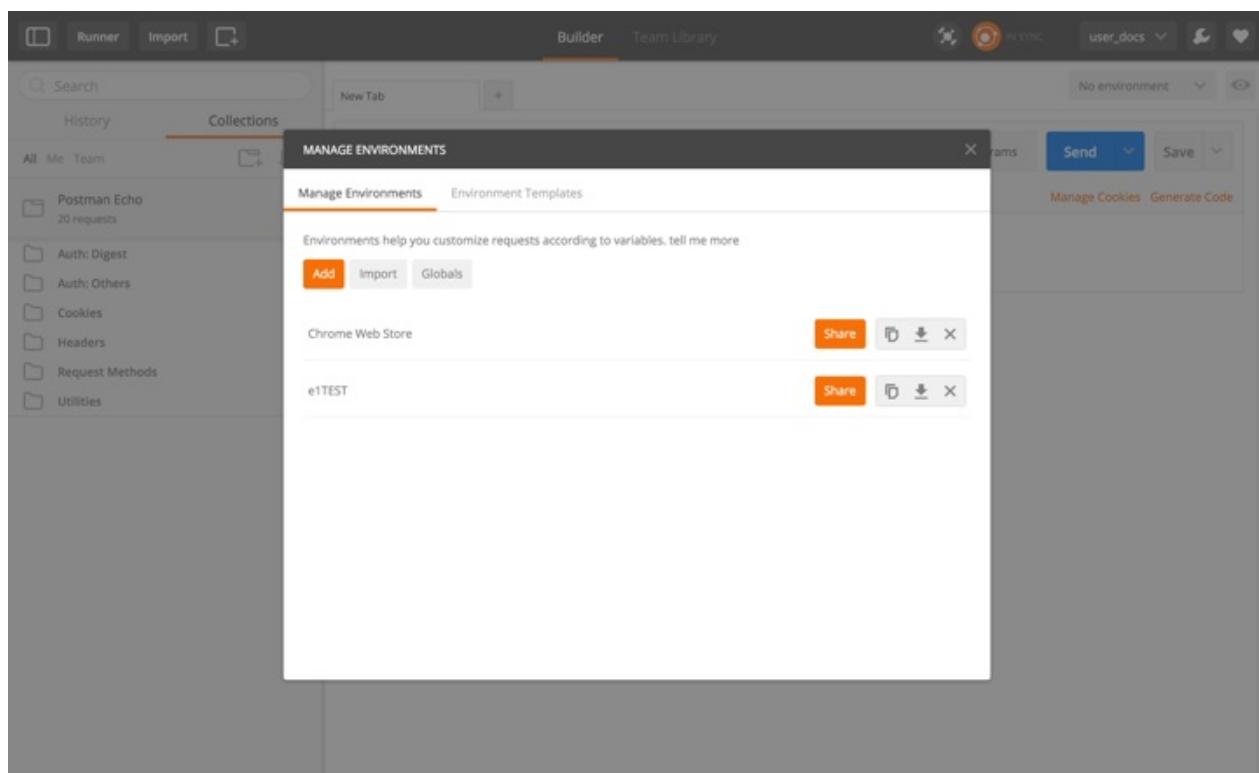
# Manage environments

Each environment is a set of key-value pairs, with the key as the variable name. These can be edited using the [data editor](#).

Environment and global variables will always be stored as strings. If you're storing objects/arrays, be sure to `JSON.stringify()` them before storing, and `JSON.parse()` them while retrieving.

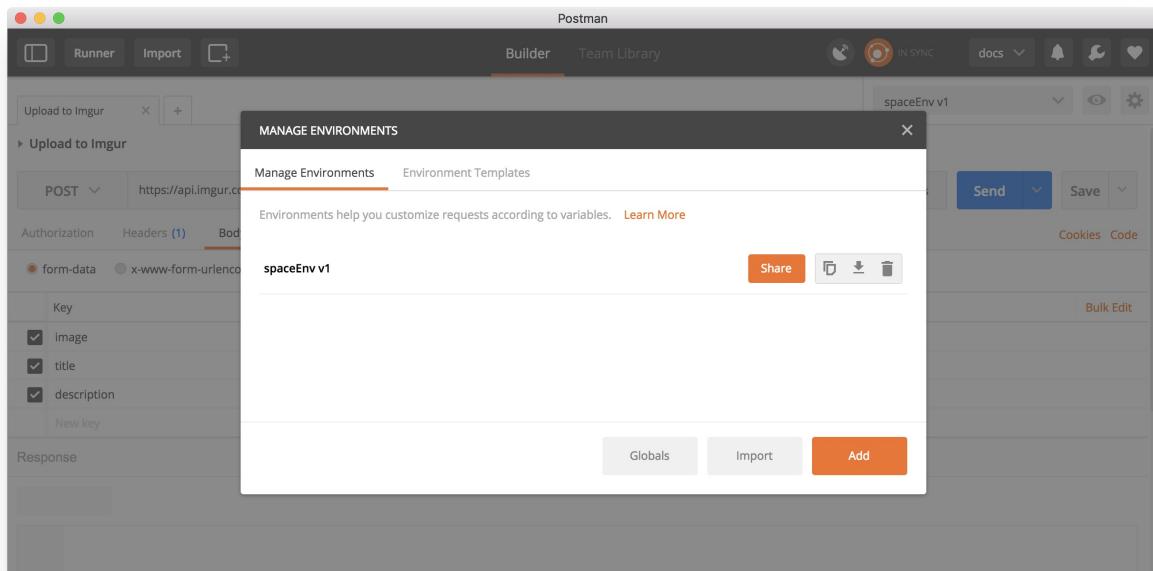
## What are environments?

While working with APIs, you will often need to have different setups. For example, your local machine, the development server, or the production API. Environments give you the ability to customize requests using variables. This way you can easily switch between different setups without changing your requests. You won't have to worry about remembering all those values once they are in Postman. Environments can be downloaded and saved as JSON files and uploaded later.



## Create a new environment

Click the gear icon in the upper right corner of the Postman app and select “Manage Environments”. Click the **Add** button to create a new environment.

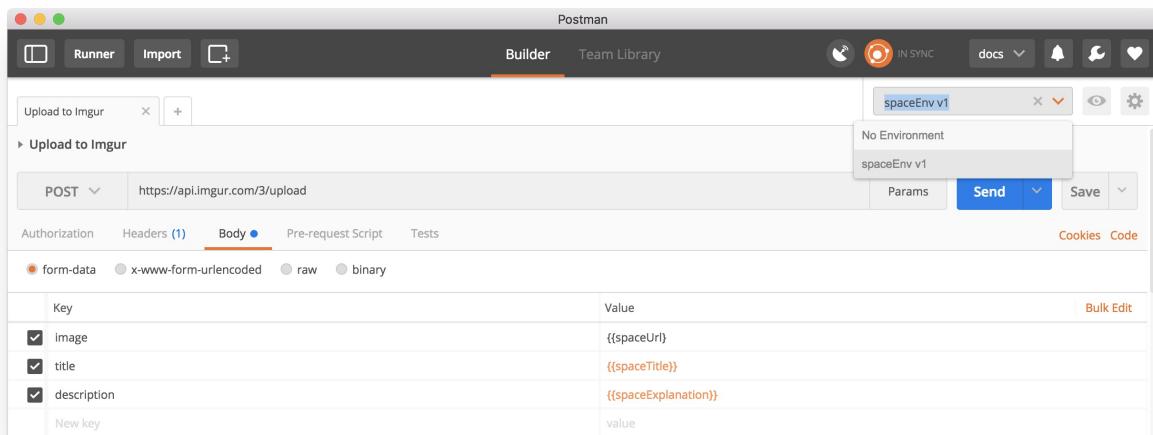


### Manage environments

Click the gear icon in the upper right corner of the Postman app and select “Manage Environments”. In addition to creating and sharing an environment, you can also duplicate, export, and delete an environment. This is also where you can import an environment as a single JSON file.

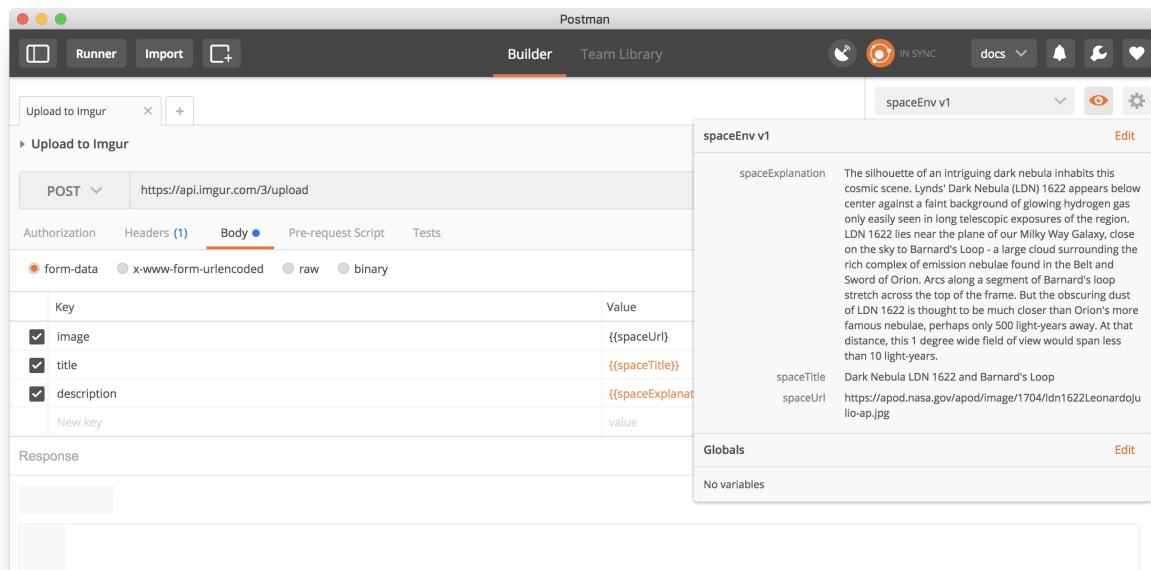
### Select an active environment

Click the dropdown in the upper right corner of the Postman app to select an active environment, or type in the environment name. Once you select an environment, you can access variables within the active environmental scope.



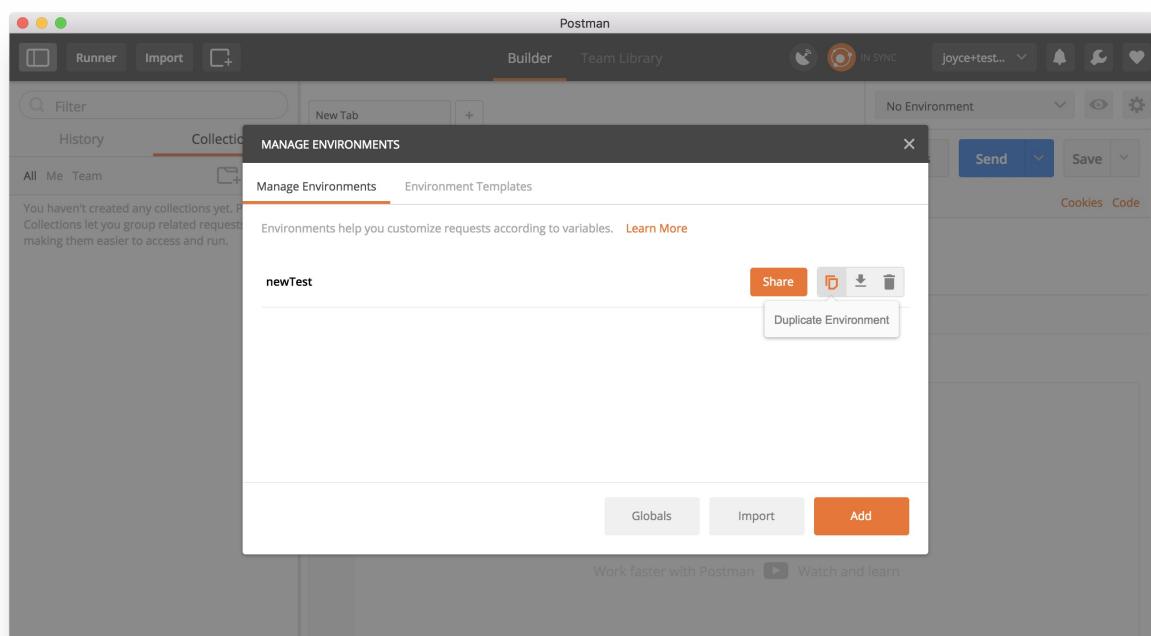
### Edit an active environment

Click the Quick Look icon in the upper right corner of the Postman app to display the environment and global variables. Clicking on the **Edit** link will open a modal for editing keys and values.



## Share an environment

Click the gear icon in the upper right corner of the Postman app and select “Manage Environments”. Click the **Duplicate Environment** icon next to the environment you want to share.



It's a best practice to create a duplicate, remove any sensitive values like passwords and access tokens before downloading the copy to share with someone else. When someone else imports the environment, or accesses the shared template, they can input their own personal information within their own version of the template.

For Postman Pro and Enterprise users, learn how to [share environment templates](#) with team members.

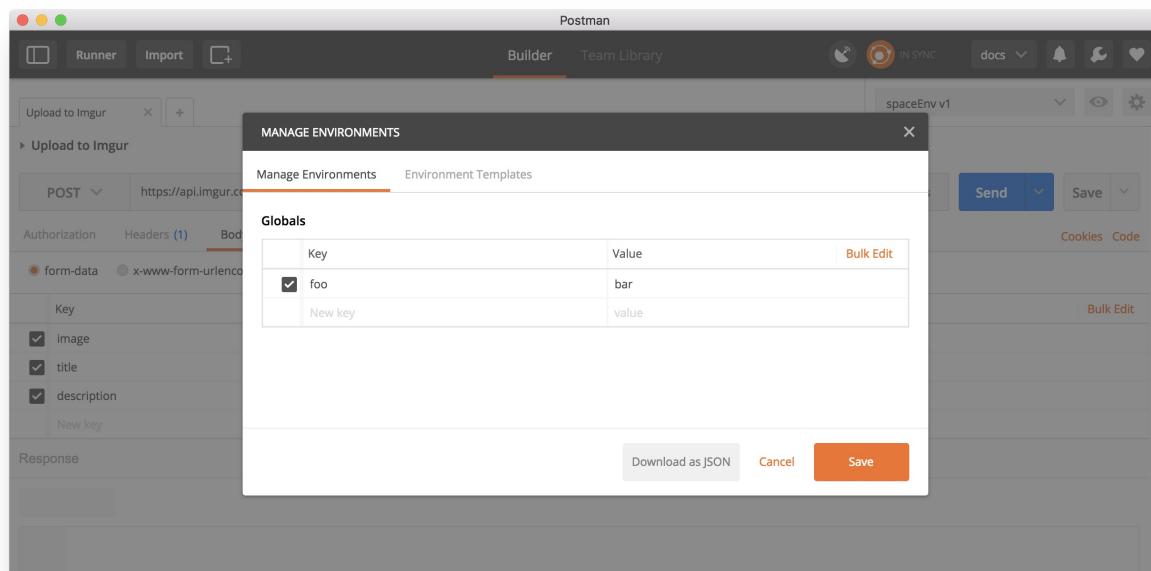
# Manage globals

Global variables provide a set of variables that are always available to you within all scopes. You can have multiple environments, and only one can be active at a time. But you'll have only one set of global variables, and they'll always be available.

**Environment and global variables will always be stored as strings. If you're storing objects/arrays, be sure to `JSON.stringify()` them before storing, and `JSON.parse()` them while retrieving.**

## Manage global variables

Click the gear icon in the upper right corner of the Postman app and select “Manage Environments”. Click on the **Globals** button at the bottom of the modal to reveal a key-value editor to add, edit, and delete global variables. This is also where you can download your global variables as a single JSON file.



## View global variables

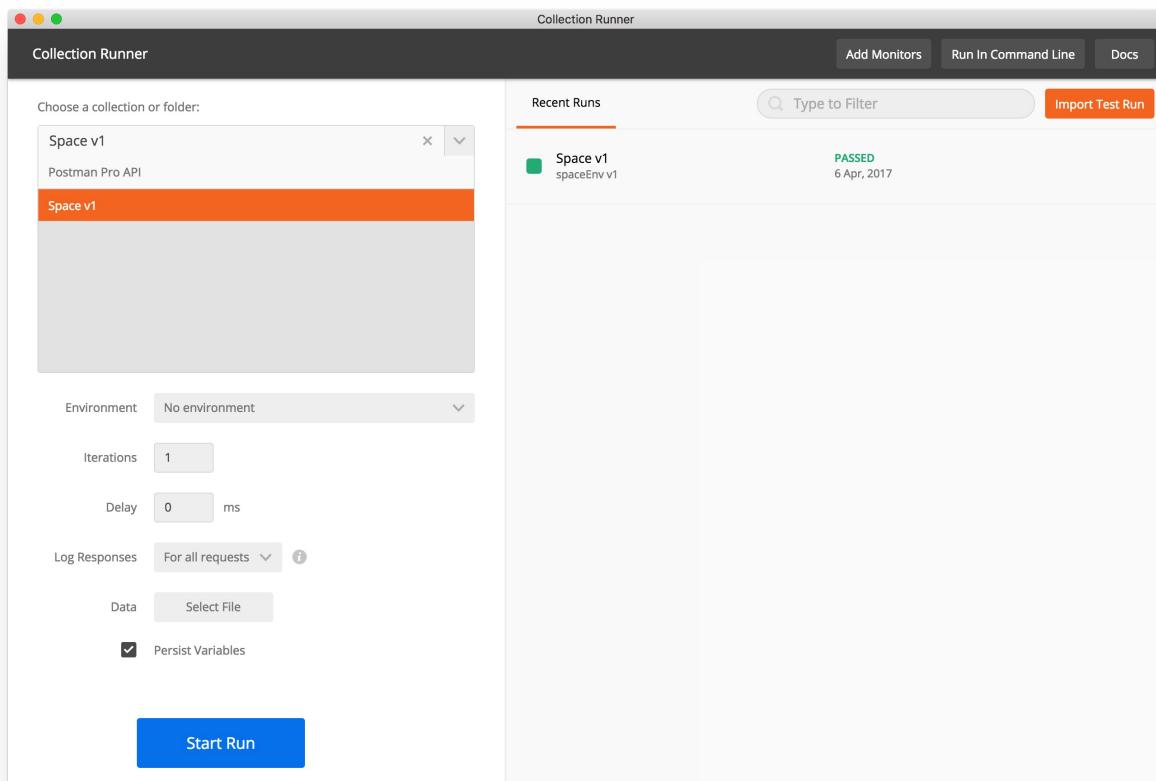
Click the Quick Look icon in the upper right corner of the Postman app to display the environment and global variables. Clicking on the **Edit** link will open a modal for editing keys and values.

# Starting a collection run

Collections are groups of requests that can be run together as a series of requests, against a corresponding environment. Using scripts, you can build integration test suites, pass data between API requests, and build workflows that mirror your actual use case of APIs.

Collections can be run within the Postman app using the collection runner, from the command line using Postman's [Newman](#) tool, or on scheduled intervals using [Postman Monitors](#).

Running a collection is useful when you want to automate API testing. When you run a collection, you're essentially sending all requests in your collection one after another. Let's go over several parameters that you can configure for a collection run.



## Collection / Folder

This is the collection or folder that you want to run. When you run a collection, all requests in the collection are sent in the order in which they appear in the main app. This means each folder is run, and each request inside the folder, is sequentially executed. You can, however, change this order to more closely mirror your workflow by using the `setNextRequest()` method. Read more about [building workflows](#).

When you select a folder here, only that folder is executed, which means only requests inside the folder are sent.

## Environment

This is the environment that will be used when the collection is run. Read more about [using environments in collection runs](#).

## Iterations

This is the number of times your collection will be run. Higher iteration counts are usually run to ensure stability of your APIs by sending different data in each iteration. Read more about [running multiple iterations](#).

## Delay

This is the interval (in ms) between each request in your collection run.

## Log Responses

This is used to limit response logging when the collection is run. By default, all responses are logged for debugging purposes, but for large collections, this can be changed to improve performance. Read more about [debugging collection requests](#).

- For all requests, responses for all requests will be logged.
- For failed requests, only responses for requests with at least one failing test will be logged.
- For no requests, no responses will be logged.

## Data

This is used to supply a data file to be used for the collection run. Read more about [data files](#).

## Persist variables

By default, any environment changes in the collection runner are not reflected back in the request builder. Read more about [using environments in collection runs](#).

# Using environments in collection runs

Environments allow you to create robust requests that can be re-used. Read more about [using variables and environments](#).

Environments can also be used in the Collection Runner. Let's look at an example collection, with one POST request that uses environment variables in its URL, body, and test script. Download the sample collection: [collection.json](#).

The screenshot shows the Postman Builder interface. A POST request is defined to `postman-echo.com/{{path}}`. In the Body tab, there is a single form-data entry named `foo` with the value `{{foo}}`. In the Tests tab, there is a snippet of JavaScript code:

```

1 let jsonData = JSON.parse(responseBody);
2 tests['Correct value is returned'] = jsonData.form.foo === 'bar'
3 // Calculate the mass of the sun, then set the variable 'foo' to 'bar2'
4
5 postman.setEnvironmentVariable('foo', 'bar2')

```

The test expects the value of `foo` in the response body to be equal to `bar`. Let us also assume that after some computation, we're resetting the value of this variable to `bar2`.

The screenshot shows the Postman Builder interface. A POST request is defined to `postman-echo.com/{{path}}`. In the Body tab, there is a single form-data entry named `foo` with the value `{{foo}}`. In the Tests tab, there is a snippet of JavaScript code:

```

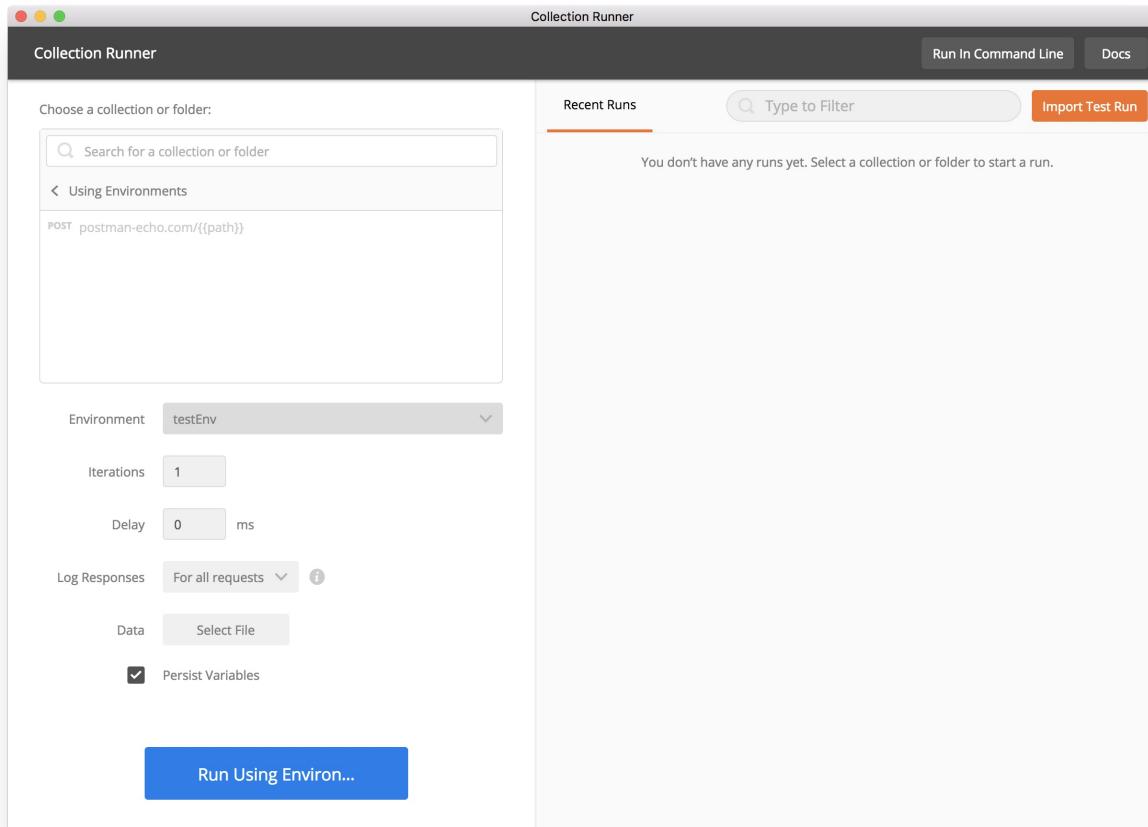
1 let jsonData = JSON.parse(responseBody);
2 tests['Correct value is returned'] = jsonData.form.foo === 'bar'
3 // Calculate the mass of the sun, then set the variable 'foo' to 'bar2'
4
5 postman.setEnvironmentVariable('foo', 'bar2')

```

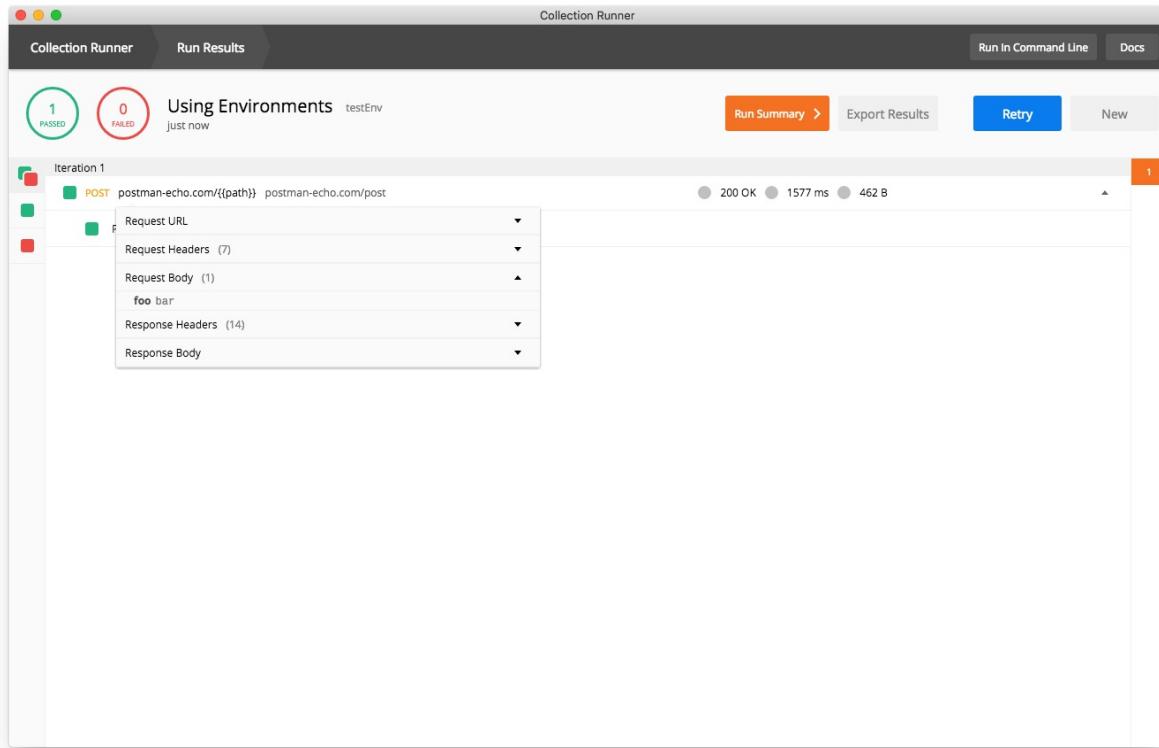
The right sidebar displays a list of snippets:

- SNIPPETS >
- Clear a global variable
- Clear an environment variable
- Response body: Contains string
- Response body: Convert XML body to a JSON Object
- Response body: Is equal to a string
- Response body: JSON value check
- Response headers: Content-Type header check

To run this collection correctly in the Collection Runner, you need to supply it the corresponding environment. Download the sample environment: [environment.json](#). In the Collection Runner, if we select our test environment from the environment dropdown on the left and run the collection, you'll see the tests pass.

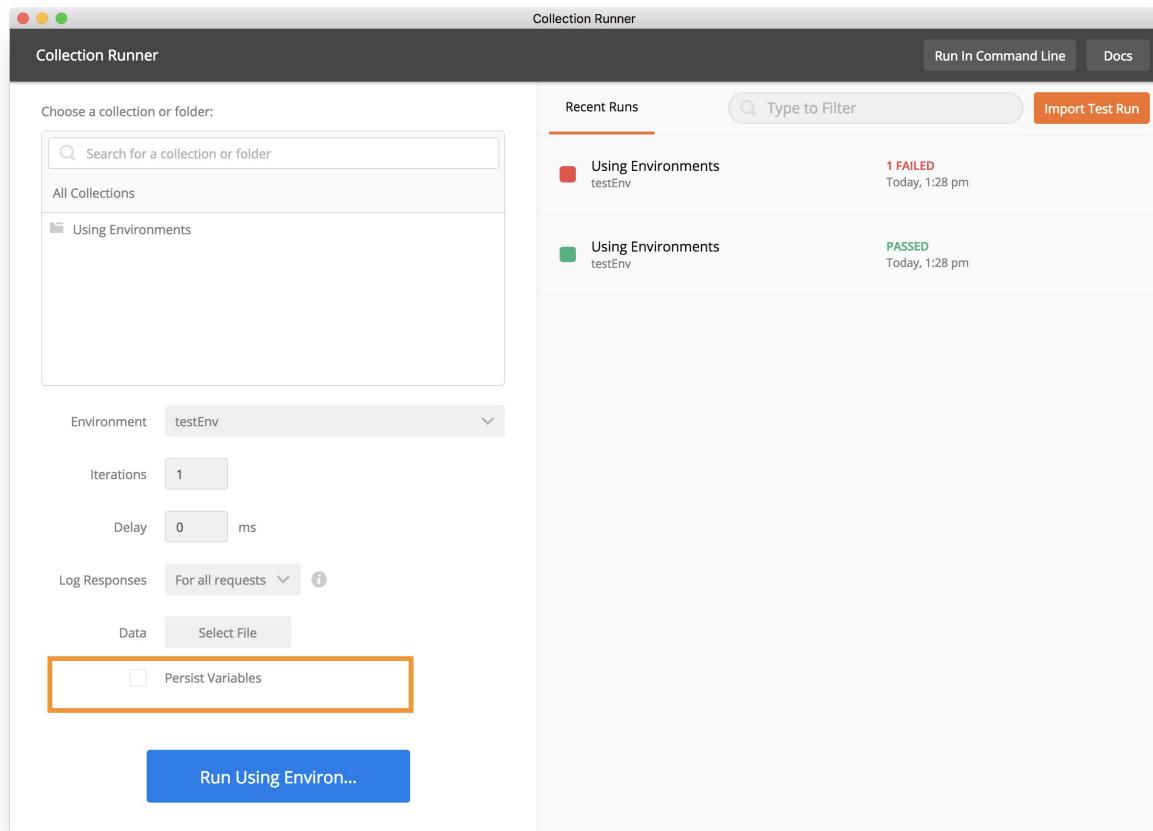


If you switch back to the main Postman app window and check the value of the variable foo, you'll see that it is now bar2.



This is because, by default, any variable changes in the environment (or globals) in the Collection Runner will be reflected in the main Postman app window since Persist Variables is checked in the options. In fact, if you run the collection once again, you'll see that it will now fail, since we changed the value of the variable foo.

By default, Persist Variables is checked the first time you open the Collection Runner. If you do not want variables to be updated during the run, deselect the Persist Variables checkbox. In this case, think of it as the Collection Runner saving the initial state of the environment (and globals), and restoring it after the run is complete. This is useful when you reuse the same variables in your requests and want to run the same collection multiple times. This will also make sure that the environment (and globals) state is not affected by a collection run.



**Download the collection and environment used in this example:**

- [collection.json](#)
- [environment.json](#)

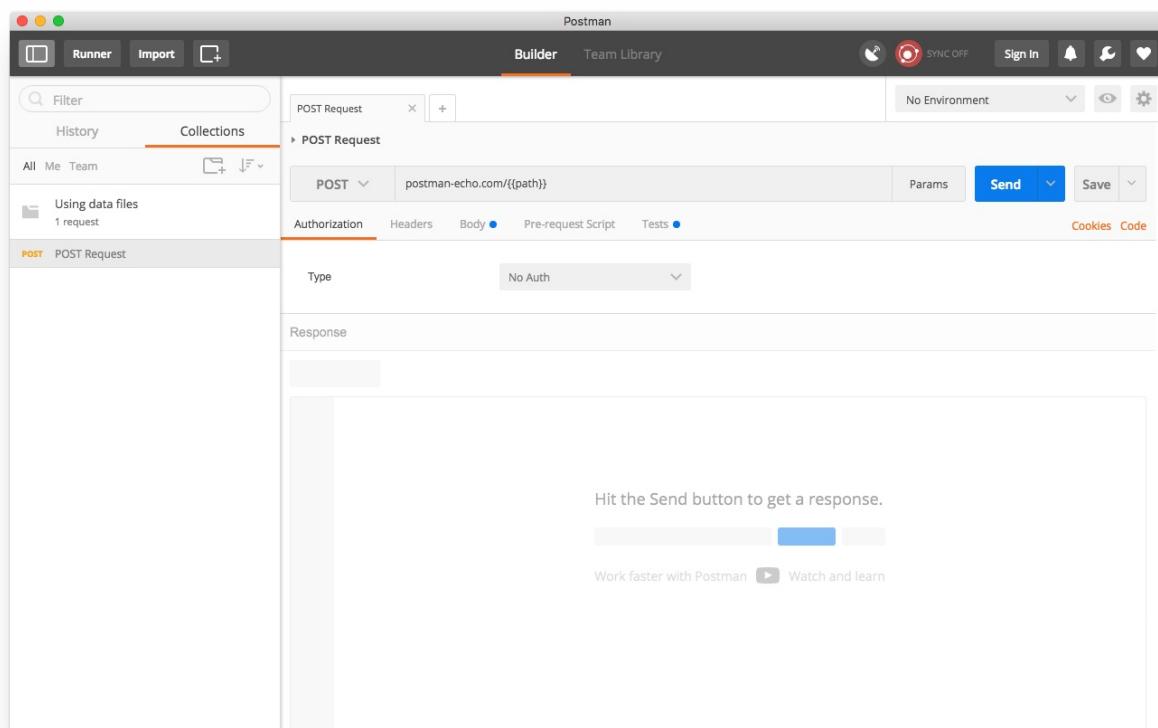
# Working with data files

Data files are extremely powerful ways to test your APIs with varying data to check if they behave properly under unexpected circumstances.

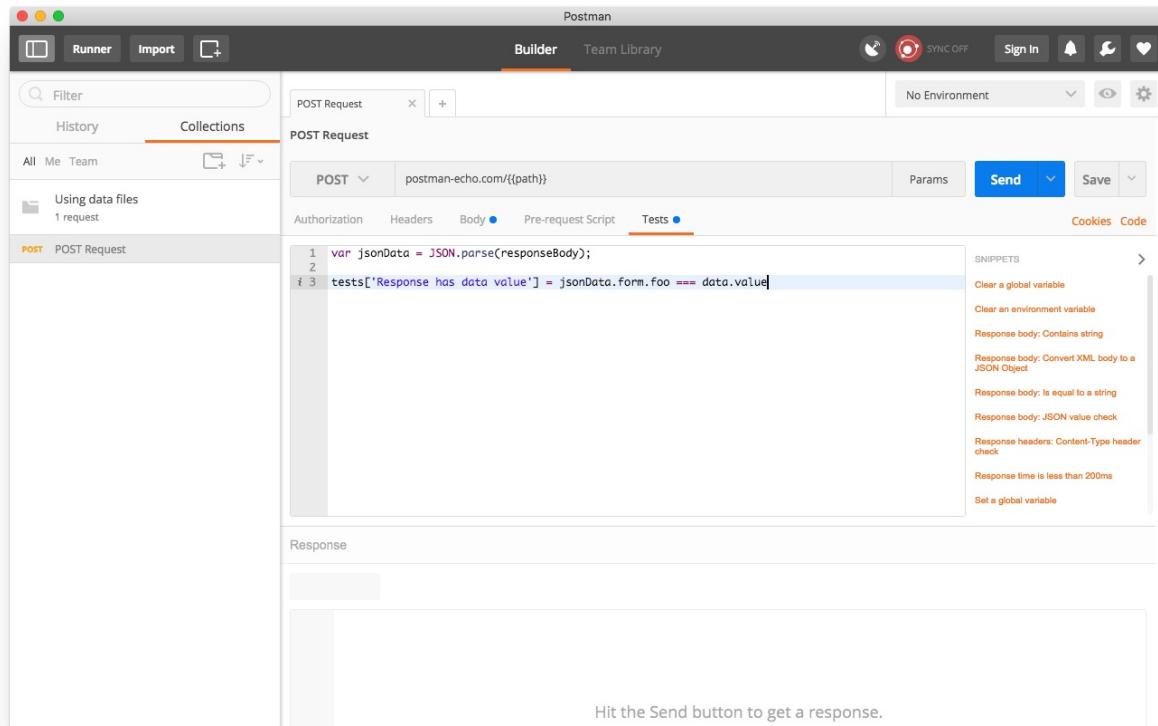
We can think of data files as parameters for each iteration of a collection run. Let's walk through an example.

**Download the collection and data files used in this example:**

- [Collection.json](#)
- [JSON, CSV](#)



Here, we have a simple collection with a single POST request. If you open up this request, you'll see two variables used in the request, path (in the URL) & value (in the request body). These are used just like environment variables. We will supply the value to these variables using a JSON / CSV file. On opening the test script, you'll see we're using some variables in the test script -data specifically. This isn't defined in the script itself. The Postman Sandbox initializes the data variable from the JSON/CSV file that we will select in the collection run.



Let's investigate the data files first. We currently support JSON & CSV files.

The JSON data file looks like this:

```
[{
  "path": "post",
  "value": "1"
}, {
  "path": "post",
  "value": "2"
}, {
  "path": "post",
  "value": "3"
}, {
  "path": "post",
  "value": "4"
}]
```

This is an array of objects. Each object represents the variable values for one iteration. Each member of this object represents a variable. In this way, in the first iteration, the variable called path will have the value post, and the variable value will have the value 1. Similarly, in the second iteration, path will still be post and value will be 2. In this example, the variable path does not change its value over iterations, but value does. This is totally up to you.

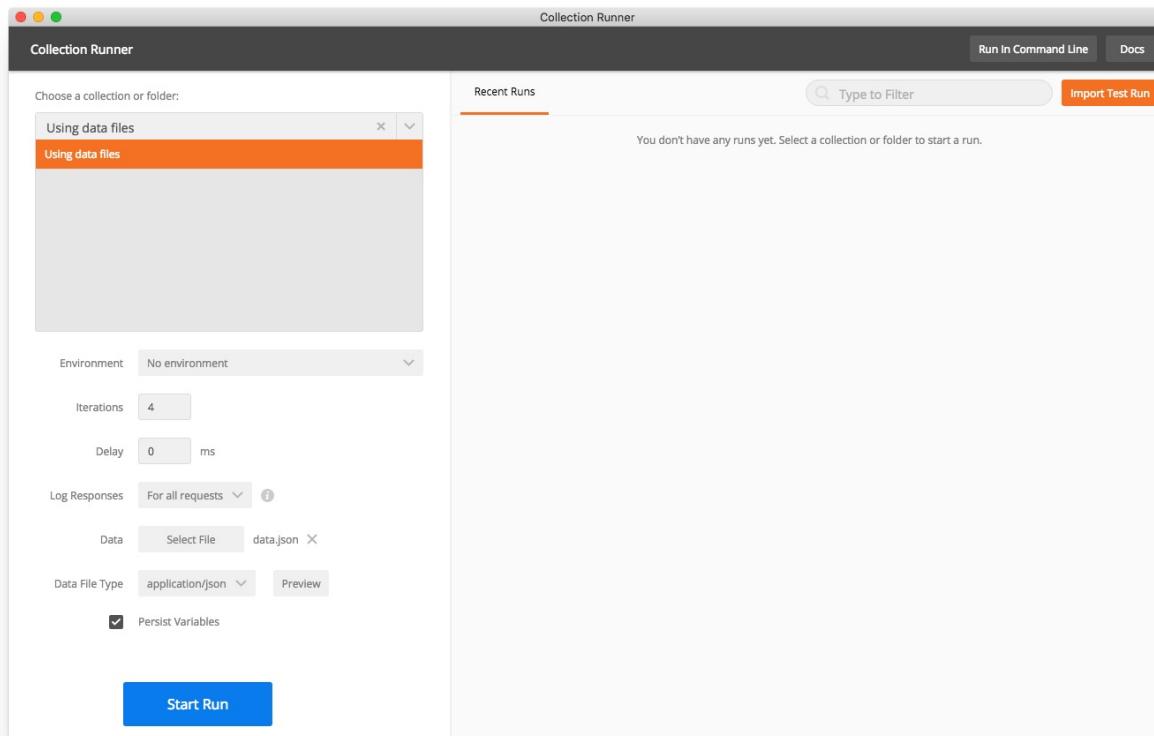
The data file can also be a CSV. The example CSV looks like this:

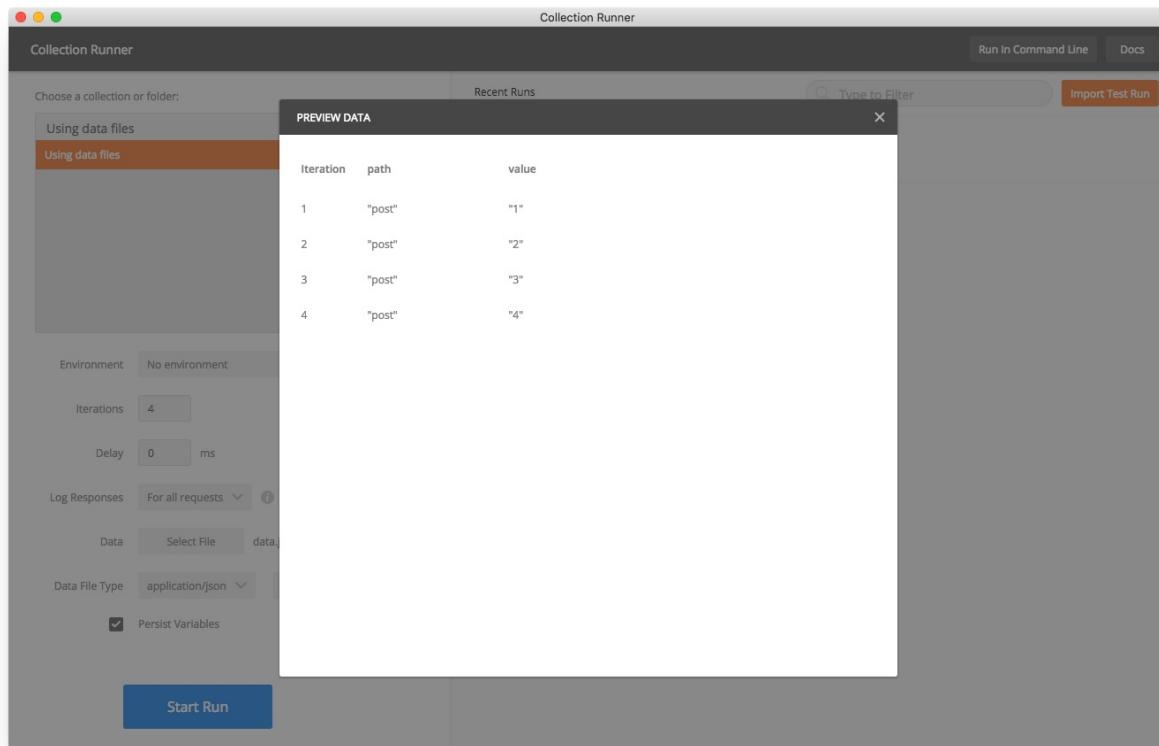
```
path, value
post, 1
post, 2
post, 3
post, 4
```

In typical CSV fashion, the first row represents all variable names, and subsequent rows represent values for these variables for each iteration. For iteration 1, path has value post, and value is 1. For the second iteration, path is still post, but value is 1.

Do note that you can only use one data file for one run.

Now that you understand how to construct data files, let's supply this data file to a Collection Run. Click Select File in the Runner, and select one of these files. You can also preview what values each variable has in each iteration by clicking on Preview next to the file name.





Let's run our collection now. You'll see that all tests pass now. If you open up the request debug tooltip, and expand Request Body, you'll see that the variable was replaced by the value, as dictated by the data file. Read more about [debugging requests](#). In fact, for different iterations, this value is different. This way, we've thrown different kinds of data to our API and have ensured that it works correctly for each case.

The screenshot shows the Postman Collection Runner interface. At the top, it displays 'Collection Runner' and 'Run Results'. Below that, a summary shows 4 PASSED and 0 FAILED tests, run 'just now'. The main area is titled 'Using data files' under 'No Environment'. It lists four iterations of a POST request to 'postman-echo.com/post'. Each iteration includes a 'PASS' assertion for 'Response has data value'. The response details for each iteration show a 200 OK status with 1613 ms latency and 548 B size. The response body is a JSON object with 'foo' set to '3'. The interface also includes 'Run Summary', 'Export Results', 'Retry', and 'New' buttons.

Let's also take a look at our test scripts once again. The variable data is a predefined variable that gets the values from the data file. With each iteration, its value is updated with new data from our file. data is an object with all variables you defined in your file as its keys. Since this API echoes back whatever is sent to it, we're asserting that the returned value from Echo is the same as the one dictated by our file.

Data variables can be used in all places that environment variables can be used, in the exact same way, except in pre-request & test scripts.

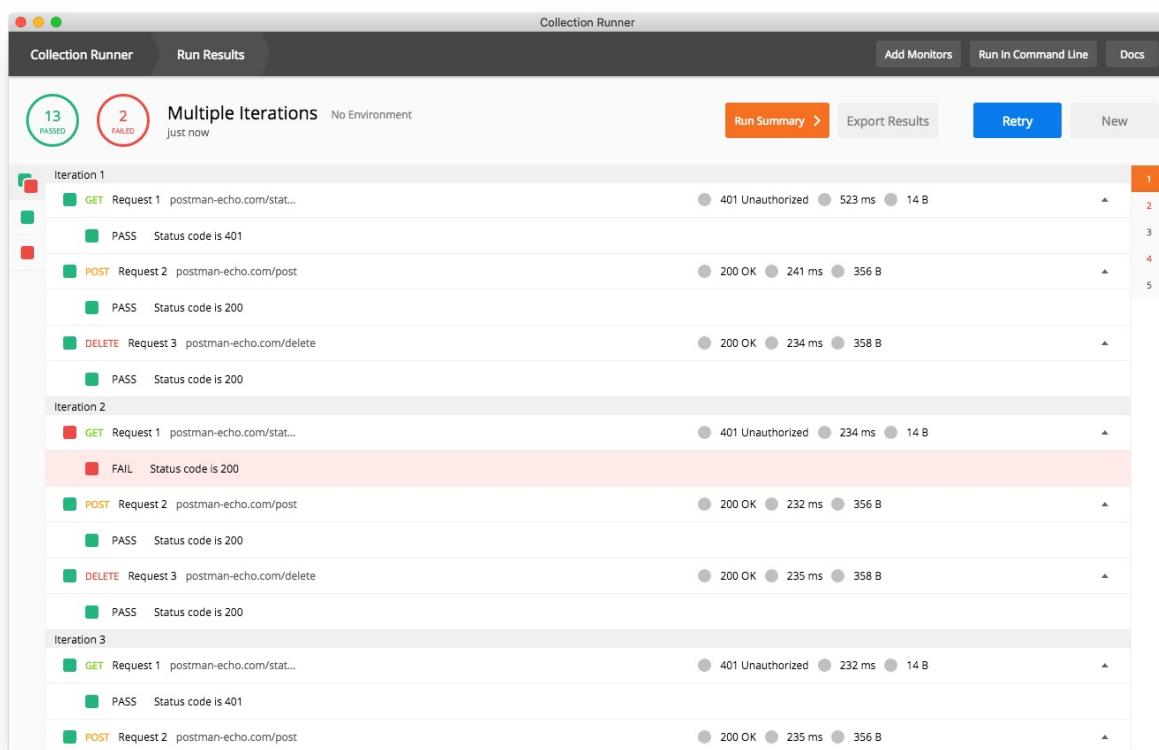
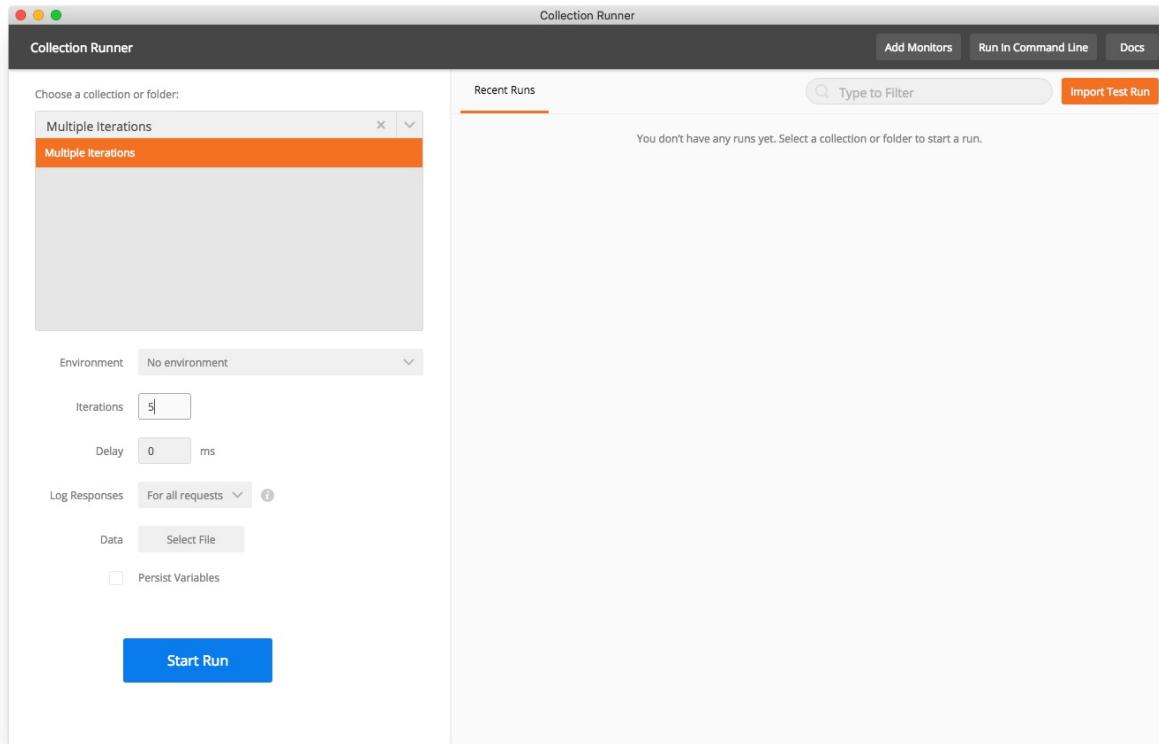
# Running multiple iterations

**Download the collection used in this example:**

- [collection.json](#)

The iterations of a collection run reflect how many times the collection will run. Here we have a collection that is run with 5 iterations.

## Running multiple iterations

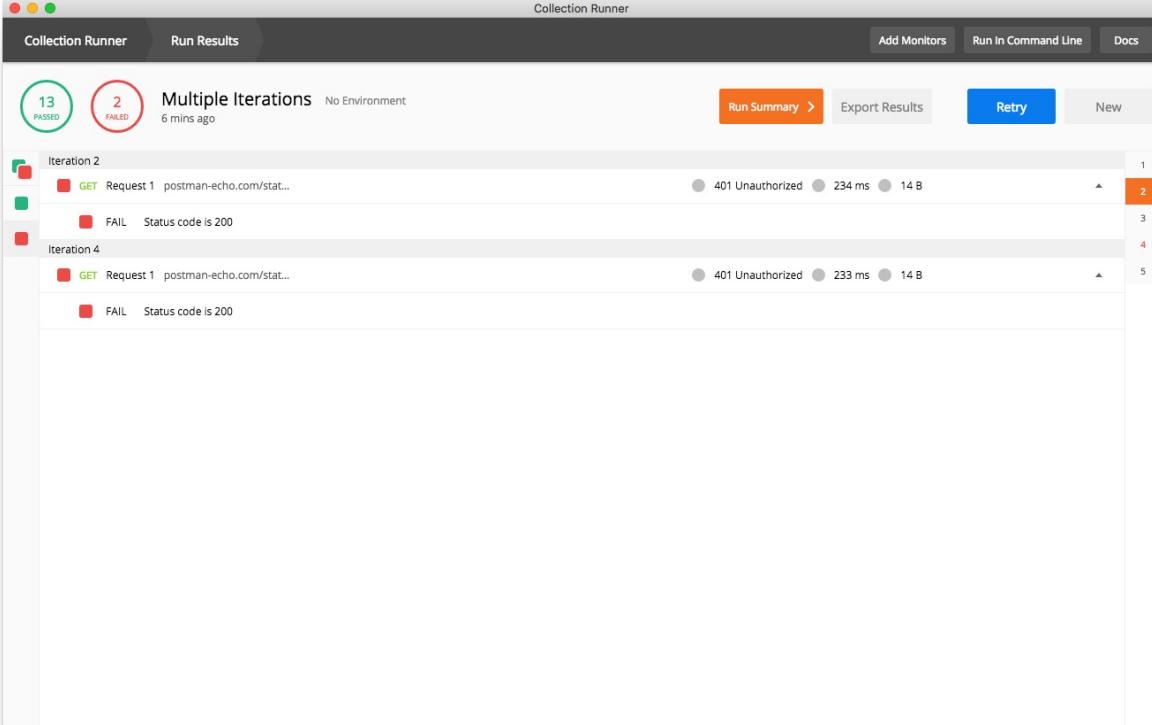


## Switching between iterations

To quickly jump between iterations, you can click on one of the numbers on the right sidebar, each of which represents one iteration.

## Using green and red filters

The left sidebar contains three filters, which can be used to show all, passed, or failed tests. This is super useful when trying to look for tests that failed so that you can quickly find bugs in your API.



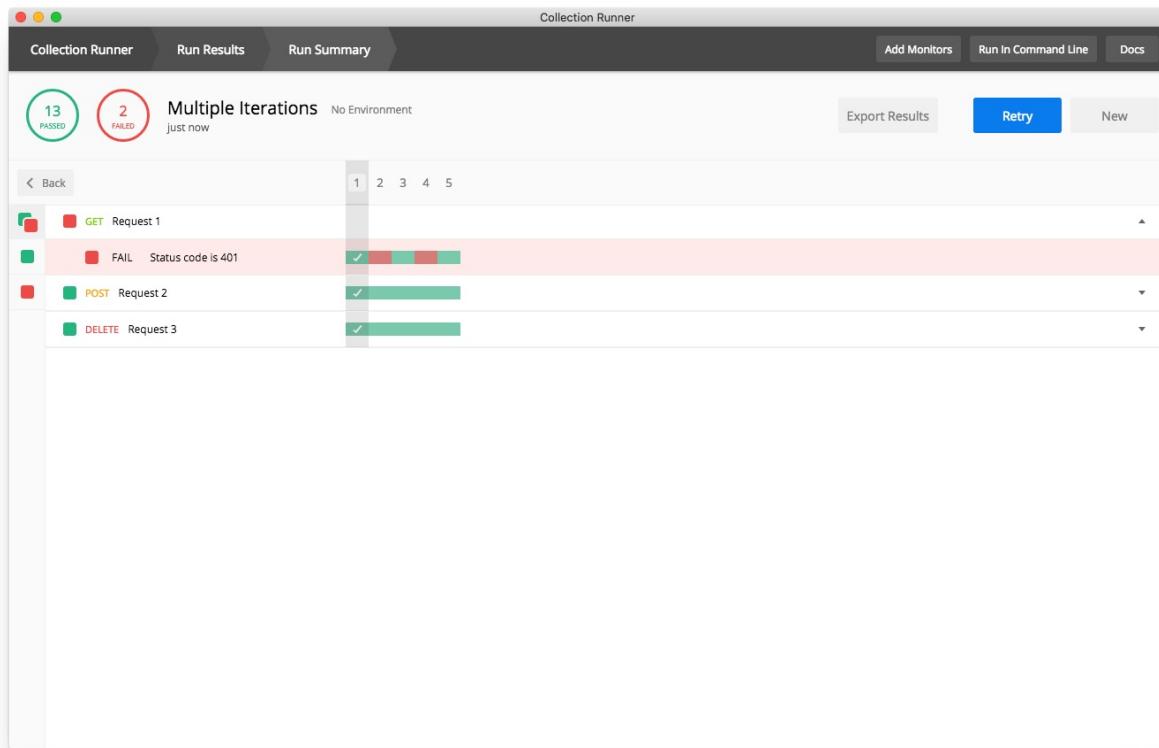
The screenshot shows the Postman Collection Runner interface. At the top, there are tabs for 'Collection Runner' and 'Run Results'. On the right, there are buttons for 'Add Monitors', 'Run In Command Line', and 'Docs'. Below the tabs, a summary shows '13 PASSED' and '2 FAILED'. The main area is titled 'Multiple Iterations' with 'No Environment' and a timestamp '6 mins ago'. It displays two iterations:

- Iteration 2:** Contains one successful 'GET' request to 'postman-echo.com/stat...' and one failing 'FAIL' request 'Status code is 200'. The failing request has a status of '401 Unauthorized' and took '234 ms' with '14 B' body size. A red error icon is next to it.
- Iteration 4:** Contains one successful 'GET' request to 'postman-echo.com/stat...' and one failing 'FAIL' request 'Status code is 200'. The failing request has a status of '401 Unauthorized' and took '233 ms' with '14 B' body size. A red error icon is next to it.

On the far right of each iteration row, there are numerical indicators (1, 2, 3, 4, 5) and small up/down arrows, likely for sorting or filtering.

## Debugging with multiple iterations

When working with multiple iterations, it can quickly become tedious to switch between them to check if everything worked as you'd expect. For this reason, there's a third screen in the collection runner, which is the **Run Summary** screen. When a run is finished (or stopped), you can open up the **Run Summary** screen by hitting the orange button that says Run Summary.



This screen is, as the name suggests, an overview of your run. Here, you can see each request, and its pass/fail status as a timeline. A request is treated as Passed if all tests inside it pass. Similarly, if one or more tests fail, the request is marked as Failed.

The numbers in the header represent the iteration you are working with. It becomes very easy to pinpoint the test that is misbehaving. Clicking on an iteration in the header will take you to that iteration, so you can further investigate what might be going wrong.

Iterations in the collection runner are 1-indexed with the first iteration beginning with a count of 1. Note that this is different than the iteration count accessible programmatically in the [Postman sandbox](#), which is 0-indexed with the first iteration beginning with a count of zero.

Read more about [debugging collection runs](#).

# Building workflows

**Download the collection used in this example:**

- [collection.json](#)

## Basic usage

When you start a collection run, all requests are run in the order you see them in the main app. This means that all requests inside are executed first, by order of the folder they are in, and then any requests that are in the root of the collection. However, you can override this behavior using a [built-in function](#) called `setNextRequest()`.

`setNextRequest()`, as the name suggests, will allow you to specify which request will be run next. The easiest way to understand this is to look at a sample collection.

Let's assume that we have a collection with four requests. If you run this collection directly, the collection runner will run all four requests in order.

## 创建工作流

The screenshot shows the Postman application interface. On the left, there's a sidebar with a 'Collections' tab selected, displaying various API collections like 'Sync API v2', 'Alcohol', 'BitBucket Repository Tests', etc. In the main area, 'Request 1' is selected. The request details show a GET method to 'postman-echo.com/get'. The 'Tests' tab is active, containing a snippet of JavaScript code:

```
// Some code here
// postman.setNextRequest('Request 4')
// Some code here
```

Below the code editor is a 'Response' section with a note: 'Hit the Send button to get a response.'

The screenshot shows the Collection Runner interface. It displays a summary with 0 passed and 0 failed tests. The 'Run Results' tab is selected, showing the execution of 'Iteration 1' with four requests:

- Request 1: GET postman-echo.com/get - Status: 200 OK, Time: 598 ms, Body: 180 B
- Request 2: GET postman-echo.com/get - Status: 200 OK, Time: 257 ms, Body: 282 B
- Request 3: GET postman-echo.com/get - Status: 200 OK, Time: 258 ms, Body: 282 B
- Request 4: POST postman-echo.com/post - Status: 200 OK, Time: 253 ms, Body: 347 B

Let's now add `postman.setNextRequest()` to Request 1's test script, as shown.

`postman.setNextRequest()` is a function with one argument, which is the name or ID of the request you want to run next. In the example, we're setting the next request to Request 4 in

the test script for Request 1. This means the execution will jump to Request 4 after Request 1 has completed. If we run the same collection now, you'll see that only two requests are run now.

Postman Builder interface showing a collection named "Request 1". The collection contains four requests:

- Request 1:** GET postman-echo.com/get. Tests tab contains:
 

```
1 // Some code here
2
i 3 postman.setNextRequest('Request 4')
4
5 // Some code here
```
- Request 2:** GET postman-echo.com/get
- Request 3:** GET postman-echo.com/get
- Request 4:** POST postman-echo.com/post

The sidebar shows other collections like "Sync API v2", "Alcohol", "BitBucket Repository Tests", etc. A message at the bottom says "Hit the Send button to get a response."

Collection Runner interface showing the results of running the "Request 1" collection. The results are:

- PASSED: 0**
- FAILED: 0**

The report details the execution of Iteration 1:

- Request 1:** GET postman-echo.com/get. Status: 200 OK, Duration: 251 ms, Size: 282 B. Note: This request does not have any tests.
- Request 4:** POST postman-echo.com/post. Status: 200 OK, Duration: 257 ms, Size: 347 B. Note: This request does not have any tests.

Note that `setNextRequest()` will only work with the collection runner and Newman where the intent is to run a collection, as opposed to sending a single request.

## Advanced usage

Now that we have a good understanding of how `setNextRequest()` works, we can do some pretty advanced stuff with it. Since you are no longer restricted by the order in which you define your requests, you can jump around your collection, establish conditional logic, or skip unnecessary requests. This [blog post](#) explains how you can write a collection that will generate Spotify playlists for you based on your favorite musical artists.

There are some gotchas to keep in mind:

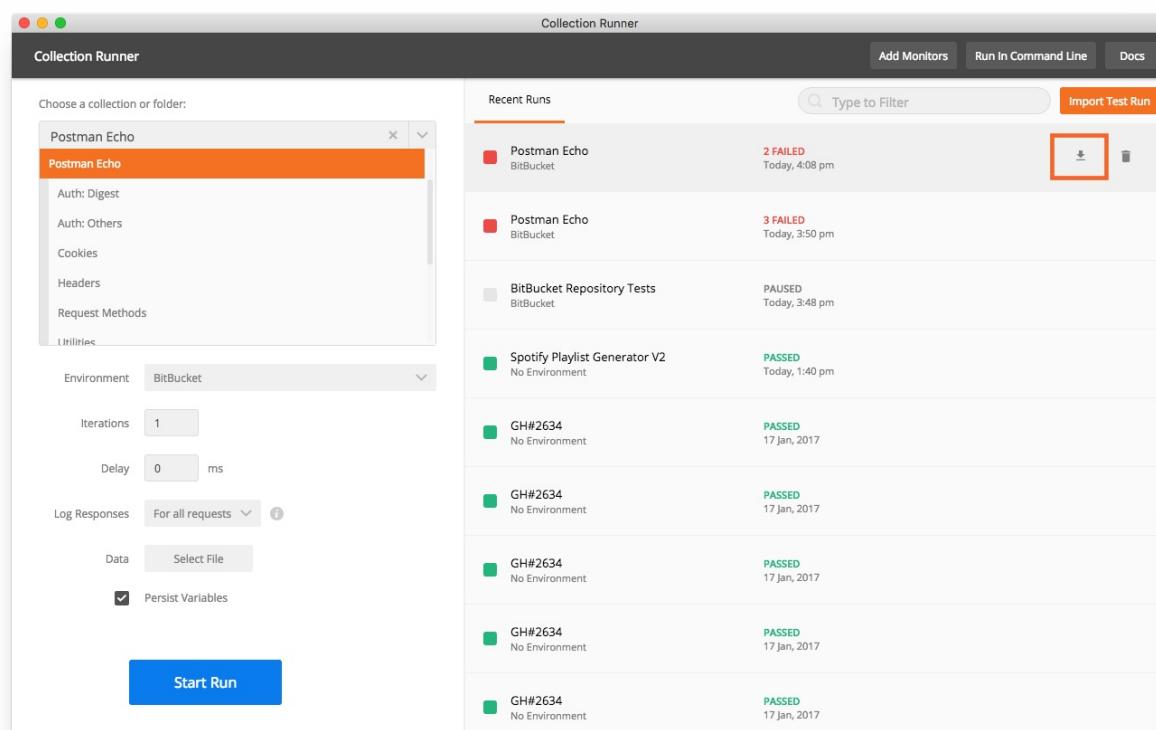
- `setNextRequest()` is always executed at the end of the current script. This means that if you put `setNextRequest()` before other code blocks, these blocks will still be executed.
- `setNextRequest()` has a scope, which is the source of your collection run. This means that if you run a collection, you can jump to any request in the collection (even requests inside folders, using the same syntax). However, if you run a folder, the scope of `setNextRequest()` is limited to that folder. This means that you can jump to any request within this folder, but not ones that are outside of the folder. This includes requests inside other folders, and also root-level requests in the collection. To read more about [running collections or folders](#).

# Sharing a collection run

Sharing a collection run is simple. You just export the collection run from the collection runner and the recipient imports it into their Postman app.

## Exporting a run

To export a collection run, click on the download icon that appears when you hover over over a run. You can then save the generated JSON file wherever you want.



If the collection run name isn't enough to decide which run you want to export, open up the run by clicking on the collection run. Then, click on the Export Results button in the Runner's header. You can then save the generated JSON file wherever you want.

The screenshot shows the Postman Collection Runner interface. At the top, there are tabs for 'Collection Runner' and 'Run Results'. Below that, a summary shows 64 PASSED and 2 FAILED tests. The title 'Postman Echo' and 'BitBucket' is followed by a timestamp '3 hrs ago'. On the right, there are buttons for 'Run Summary', 'Export Results' (which is highlighted with a red box), 'Retry', and 'New'. The main area displays a list of test iterations. Iteration 1 contains several test steps: 'DigestAuth Request https://echo.getpostman...', 'DigestAuth Success https://echo.getpostman...', 'DigestAuth Success copy https://echo.getpostman...', 'Set Cookies https://echo.getpostman...', and 'Delete Cookies https://echo.getpostman...'. Each step includes status indicators (green for PASS, red for FAIL) and details like response codes (e.g., 401 Unauthorized, 200 OK), execution time (e.g., 1114 ms, 271 ms), and body size (e.g., 12 B, 22 B). A small orange box in the top right corner indicates there is one more iteration.

## Importing a run

To import a run that someone shared with you, you'll have to click on the orange Import Test Runbutton on the Runner's selection screen. This will then open up an explorer that you can use to navigate and import a JSON collection run.

# Debugging a collection run

Oftentimes, things don't go according to plan and your collection tests will fail even when you expect them all to pass. When this happens, there are two ways you can debug your requests.

In this example, we're running the [Postman Echo collection](#).

Request	Test	Status	Details
DELETE /cookies	Body contains cookie foo1	FAIL	Body contains cookie foo1
DELETE /cookies	Body contains cookies	PASS	Body contains cookies
DELETE /cookies	Body does not contain cookie foo2	FAIL	Body does not contain cookie foo2
GET /cookies	Status code is 200	PASS	Status code is 200
GET /cookies	Body contains cookies	PASS	Body contains cookies
GET /cookies	Cookies object is empty	PASS	Cookies object is empty
GET /cookies	Status code is 200	PASS	Status code is 200
GET /headers	Body contains headers	PASS	Body contains headers
GET /headers	Header contains host	PASS	Header contains host
GET /headers	Header contains test parameter sent as part of request header	PASS	Header contains test parameter sent as part of request header
GET /response-headers	Body contains Content-Type	PASS	Body contains Content-Type

In the Delete Cookies request, we expect a certain cookie to be returned by the server, and this is what the test checks as well. Postman Echo's [Cookies](#) endpoint returns whatever cookies are sent to it. It also sends a JSON representation of these in the response body. This is what we're using to check if a certain cookie was returned.

As we can see, this test is failing. Let's investigate why.

## Debugging using the Request & Response body

Collection Runner

Collection Runner Run Results

**Postman Echo** BitBucket just now

Iteration 1

- PASS Status code is 200
- GET Delete Cookies https://echo.getpostman...
- Request URL
- Request Headers (5)
- Request Body
- Response Headers (13)
  - 200 OK 523 ms 26 B
- Response Body
  - 200 OK 257 ms 26 B
- PASS Cookies object is empty
- PASS Status code is 200
- GET Request Headers https://echo.getpostman...
- PASS Body contains headers
- PASS Header contains host
- PASS Header contains test parameter sent as part of request header
- GET Response Headers https://echo.getpostman...
- PASS Body contains Content-Type

Run Summary > Export Results Retry New

Collection Runner

Collection Runner Run Results

**Postman Echo** BitBucket just now

Iteration 1

- GET Delete Cookies https://echo.getpostman...
  - Request Headers (2)
  - Request Body
  - Response Headers (13)
    - Access-Control-Allow-Credentials
    - Access-Control-Allow-Headers
    - Access-Control-Allow-Methods
    - Access-Control-Allow-Origin
    - Access-Control-Expose-Headers
    - Connection keep-alive
    - Content-Encoding gzip
    - Content-Length 48
    - Content-Type application/json; charset=utf-8
    - Date Fri, 31 Mar 2017 10:20:49 GMT
    - ETag W/"1a-NGunheS3XF0tLxz33frD4Q"
    - Server nginx/1.10.1
  - Vary Accept-Encoding
- PASS Body contains headers
- PASS Header contains host
- PASS Header contains test parameter sent as part of request header
- GET Response Headers https://echo.getpostman...
- PASS Body contains Content-Type

Run Summary > Export Results Retry New

As the test says, we're expecting a cookie named foo1 to be returned as part of the response.

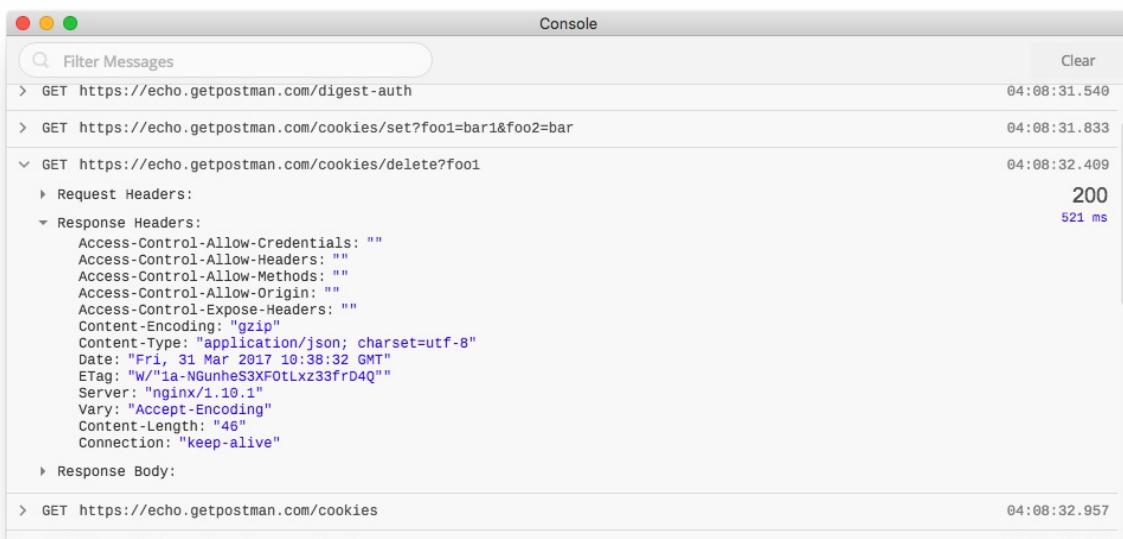
If you click on any request name in your collection run, you'll notice a tooltip appear. This has useful information pertaining to your request, information you might need when figuring out what went wrong. Expanding the Response Body section, we can see clearly that the

response does not contain the cookie we expect. Moreover, upon expanding the Response Headers section, we see that the cookie was not sent at all. We infer that something must be wrong with the way Postman Echo handles cookies. We can now go ahead and patch this up in our API and try again.

Note that only response bodies less than 300KBs are attempted to be displayed. Your response headers and bodies are never synced for security reasons. You can control which bodies show up in this tooltip by using the Log responses dropdown when [starting a collection run](#).

## Debugging using the Postman Console

Debugging using the Postman Console requires you to have the console open before you start your run. You can read about the [Postman Console](#).



The Postman Console will record all requests and display them in a list.

Let's find the request that's causing problems here and expand its response headers. Here too, we see that the Postman Echo endpoint did not return a cookie. This must be why our test is failing. We can then infer that the endpoint is misbehaving and needs to be looked at.

Any console.logs that you have in your test scripts will also appear here, so you can log things in the console if you're debugging a complex test script.

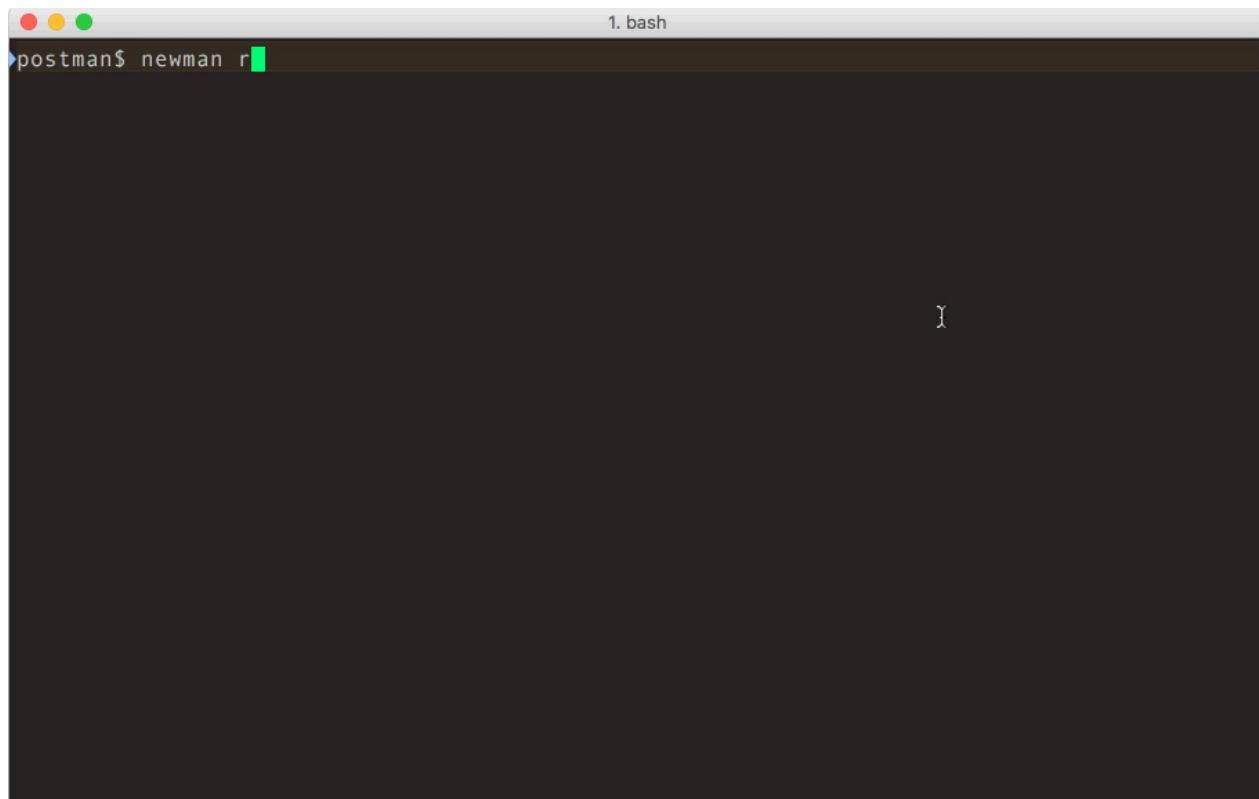
# Command line integration with Newman

- 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 dev_environment.json
```

## 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 with code=0	Continue running tests even after a failure, but exit
--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 ``--bail flag 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 outfile.json
```

**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 fields",  
    "schema": "https://schema.getpostman.com/json/collection/v2.0.0/collection.json"  
  },  
  "item": [  
    {  
      "name": "Form data upload",  
      "event": [  
        {  
          "listen": "test",  
          "script": {  
            "type": "text/javascript",  
            "exec": [  
              "var response = JSON.parse(responseBody).files[\"sample-file.txt\"];",  
              "",  
              "tests[\"Status code is 200\"] = responseCode.code === 200;  
              ",  
              "tests[\"File was uploaded correctly\"] = /^data:application/octet-stream;base64/.test(response);",  
              ""  
            ]  
          }  
        }  
      ],  
      "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://echo.getpostman.com/post` via a `POST` request."  
      },  
      "response": []  
    }  
  ]  
}
```

The file `sample-file.txt` must be present in the same directory as the collection. The collection can 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 its tests) fail, and so on.

## Building custom reporters

A custom reporter is a Node module with a name of the form `newman-reporter-`. 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 an event emitter that triggers the following events: https://github.com/postmanlabs/newman#newmanrunevents
  // reporterOptions is an object of the reporter specific options. See usage examples below for more details.
  // collectionRunOptions is an object of all the collection run options: https://github.com/postmanlabs/newman#newmanrunoptions-object--callback-function--run-emitter
};
```

3. Publish your reporter using npm publish, or use your reporter locally [see usage instructions](#).

Scoped reporter package names like @myorg/newman-reporter- 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-, where 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 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-name>
<option-value> # The option is optional
```

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](#).

# Integration with Jenkins

- Newman

Postman contains a full-featured [testing sandbox](#) that lets you write and execute JavaScript based tests for your API. You can then hook up Postman with your build system using [Newman](#), the command line collection runner for Postman. Newman allows you to run and test a Postman Collection.

Newman and Jenkins are a perfect match. Let's start setting this up. We are using Ubuntu as a target OS as in most cases your CI server would be running on a remote Linux machine.

## Installation

1. [Install Jenkins](#).
2. Install NodeJS and npm. Newman is written in NodeJS and we distribute the official copy through npm. Install [nodejs and npm for Linux](#).
3. Install Newman globally, to set up Newman as a command line tool in Ubuntu.

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

## Run a collection in Postman

We are assuming that you already have a Postman Collection with some tests. Run the collection in the Postman app. This is what the output looks like in Postman's collection runner.

The screenshot shows the 'COLLECTION RUNNER' interface in Postman. On the left, a sidebar lists 'Previous Runs' with entries like 'Jenkins demo' (a few seconds ago), 'Jenkins demo' (a minute ago), 'Jenkins demo' (2 minutes ago), 'Jenkins demo' (5 minutes ago), and 'Postman Echo' (21 minutes ago). A prominent orange button labeled 'Import Test Run' is located at the top of this sidebar. The main area is divided into 'CURRENT RUN' and 'RESULTS'. In 'CURRENT RUN', a search bar contains 'jenkins demo' and highlights 'Jenkins demo'. Below it, there's a 'Postman Echo' section with fields for 'Environment' (set to 'No environment'), 'Iteration' (set to 1), and 'Delay' (set to 0). A 'Data File' field with a 'Choose Files' button and the message 'No file chosen' is also present. A large orange 'Start Test' button is centered below these fields. To the right, the 'RESULTS' panel shows a summary: '0 passed' and '4 failed' in 1069 ms. It details two failing tests: 'GET copy' to 'dump.getpostman.com/get' (Body contains username) and 'Default copy' to 'http://dump.getpostman.com/post' (Body contains html, Response time is less than 200ms).

Some of my tests are failing intentionally in the screenshot so we can show you the instructions for troubleshooting.

## Run a collection using Newman

Run this collection inside Newman, using the command below. If everything is set up nicely, you should see the output below.

```
prakhar@Prakhars-MBP ~> newman run Desktop/jenkins_demo.postman_collection.json
newman

jenkins_demo

→ Get copy
GET https://postman-echo.com/cookies [200 OK, 532B, 1458ms]
1. Status code is 200

→ Defualt Copy
GET https://postman-echo.com/headers [200 OK, 812B, 249ms]
2. Status code is 200



|                    | executed | failed |
|--------------------|----------|--------|
| iterations         | 1        | 0      |
| requests           | 2        | 0      |
| test-scripts       | 2        | 0      |
| prerequest-scripts | 0        | 0      |
| assertions         | 2        | 2      |


total run duration: 1811ms
total data received: 283B (approx)
average response time: 853ms

# failure
# detail

1. AssertionFailure      Status code is 200
                           at assertion:1 in test-script
                           inside "Get copy" of "jenkins_demo"

2. AssertionFailure      Status code is 200
                           at assertion:1 in test-script
                           inside "Defualt Copy" of "jenkins_demo"
prakhar@Prakhars-MBP ~>
```

## Set up Jenkins

Jenkins exposes an interface at <http://localhost:8080>.

The screenshot shows the Jenkins dashboard. On the left sidebar, there are links for "New Item", "People", "Build History", "Manage Jenkins", and "My Views". The main area displays a "Welcome to Jenkins!" message with a call to action: "Please [create new jobs](#) to get started." Below this, there are two collapsed sections: "Build Queue" (No builds in the queue) and "Build Executor Status" (1 Idle, 2 Idle).

Create a new job by clicking on the “New Item” link on the left sidebar > Select a “Freestyle Project” from the options > Name your project.

The screenshot shows a modal dialog box titled "Enter an item name". A text input field contains "Jenkins\_Newman\_Test". Below the input field, a note says "» Required field". A "Freestyle project" section is shown with a description: "This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build." At the bottom of the dialog is an "OK" button.

Add a build step in the project. The build step executes a Shell command.

The screenshot shows the Jenkins configuration interface for a project named "Jenkins\_Newman\_Test". The "Source Code Management" tab is active. In the "Build Triggers" section, the "Execute shell" option is selected. At the bottom, the "Save" button is highlighted.

The command is:

```
$ newman jenkins_demo.postman_collection --exitCode 1
```

Note here that we are using the Newman command parameter “exitCode” with the value 1. This denotes that Newman is going to exit with this code that will tell Jenkins that everything did not go well.

Click the **Save** button to finish creating the project.

The screenshot shows the Jenkins configuration page for a job. The top navigation bar includes tabs for General, Source Code Management (which is selected), Build Triggers, Build, and Post-build Actions. The Source Code Management section is currently active, showing a radio button for "None". The Build Triggers section lists four options: Trigger builds remotely (e.g., from scripts), Build after other projects are built, Build periodically, and Poll SCM, each with a corresponding help icon. The Build section contains an Execute shell step with the command "newman run /Users/prakhar/Desktop/jenkins\_demo.postman\_collection.json". There are "Save" and "Apply" buttons at the bottom left, and an "Advanced..." button at the bottom right.

## Troubleshooting

Run this build test manually by clicking on the “Build Now” link in the sidebar.

**Project Jenkins\_Newman\_Test**

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Project
- Configure

**Workspace**

**Recent Changes**

**Build History**

find

RSS for all RSS for failures

**Permalinks**

Jenkins indicates that the build has failed with a red dot in the title. We can check why with the console output from Newman.

**Build #1 (Apr 14, 2017 3:45:32 PM)**

No changes.

Started by user admin

add description

Click on the “Console Output” link in the sidebar to see what Newman returned.

## Integration with Jenkins

Jenkins > Jenkins\_Newman\_Test > #1

### Console Output

```
Started by user admin
Building in workspace /Users/prakhar/.jenkins/workspace/Jenkins_Newman_Test
[Jenkins_Newman_Test] $ /bin/sh -xe /var/folders/6t/1jrhmt927x2lnql4g992wrh0000gn/T/jenkins3893355066503334401.sh
+ newman run /Users/prakhar/Desktop/jenkins_demo.postman_collection.json
newman
jenkins_demo

→ Get copy
GET https://postman-echo.com/cookies [200 OK, 532B, 1880ms]
1. Status code is 200

→ Default Copy
GET https://postman-echo.com/headers [200 OK, 691B, 293ms]
2. Status code is 200



|                    | executed | failed |
|--------------------|----------|--------|
| iterations         | 1        | 0      |
| requests           | 2        | 0      |
| test-scripts       | 2        | 0      |
| prerequest-scripts | 0        | 0      |
| assertions         | 2        | 2      |


total run duration: 2.3s
total data received: 286B (approx)
average response time: 1086ms

# failure          detail
1. AssertionFailure Status code is 200
                     at assertion:1 in test-script
                     inside "Get copy" of "jenkins_demo"

2. AssertionFailure Status code is 200
                     at assertion:1 in test-script
                     inside "Default Copy" of "jenkins_demo"
Build step 'Execute shell' marked build as failure
Finished: FAILURE
```

Fix these tests inside Postman and then try again.

COLLECTION RUNNER

Runs Statistics Run in command line Docs

Previous Runs Import Test Run

Jenkins demo a few seconds ago

Jenkins demo a minute ago

Jenkins demo 4 minutes ago

Postman Echo 20 minutes ago

CURRENT RUN

RESULTS

3 passed 1 failed 1115 ms

GET copy dump.getpostman.com/get 200 OK 597 ms

PASS Body contains username 1:0

PASS Body contains URL 1:0

Default copy http://dump.getpostman.com/post 200 OK 518 ms

PASS Body contains html 1:0

FAIL Response time is less than 200ms 0:1

Environment No environment

Iteration 1

Delay 0

Data File Choose Files No file chosen

Start Test

You can move on once you see green pass icons for all your tests like the screenshot above.

The screenshot shows the Jenkins interface for a project named "Jenkins\_Newman\_Test". The "Console Output" tab is selected. The output shows a Newman test run completed successfully. It includes logs for "jenkins\_demo" and "Default Copy" requests, a summary table of iterations, requests, and assertions, and a final message "Finished: SUCCESS".

```

Started by user admin
Building in workspace /Users/prakhar/.jenkins/workspace/Jenkins_Newman_Test
[Jenkins_Newman_Test] $ /bin/sh -xe /var/folders/6t/ljrhmt927x2lndq14g992wrh0000gn/T/jenkins5725234796952378014.sh
+ newman run /Users/prakhar/Desktop/jenkins_demo.postman_collection.json
newman

jenkins_demo

+ Get copy
GET https://postman-echo.com/cookies [200 OK, 528B, 1252ms]
✓ Status code is 200

+ Default Copy
GET https://postman-echo.com/header [200 OK, 810B, 238ms]
✓ Status code is 200

|           | executed | failed |
|iterations| 1       | 0      |
|requests  | 2       | 0      |
|test-scripts| 2       | 0      |
|prerequest-scripts| 0       | 0      |
|assertions| 2       | 0      |
|total run duration: 1580ms|
|total data received: 281B (approx)|
|average response time: 745ms|
|                                         |

Finished: SUCCESS

```

Jenkins indicates that the build succeeded with a blue ball.

## Configure frequency of runs

To set up the frequency with which Jenkins runs Newman, click on “Configure project” in the main project window and then scroll down.=. The syntax for setting the frequency is H/(30) .

The screenshot shows the "Configure Project" screen for the "Jenkins\_Newman\_Test" project. The "Build Triggers" section is active, showing a "Build periodically" trigger with a schedule of "H/30 \* \* \* \*". The "Build" section contains an "Execute shell" step with the command "newman run /Users/prakhar/Desktop/jenkins\_demo.postman\_collection.json". The "Post-build Actions" section is currently empty, with "Save" and "Apply" buttons visible.

**Note:** 30 can be replaced with another number

Jenkins will now run Newman at your desired frequency and will tell you whether the build failed or succeeded. In a bigger setup, Newman will be part of your build process and probably not the entire process. You can set up notifications and customize Jenkins as per

your needs.

You can use a wide variety of other configurations to make your collection more dynamic.

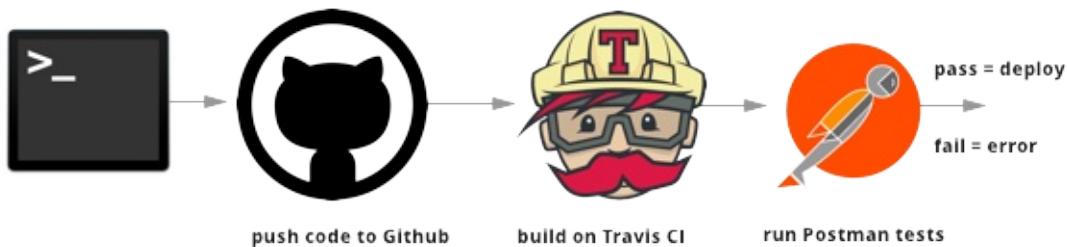
# Integration with Travis CI

- Newman

**Continuous Integration (CI)** is a practice that requires developers to **integrate** code into a shared repository several times a day. By committing early and often, the team avoids a ton of technical debt by allowing teams to detect problems early while conflicts are relatively easy to fix. Every check-in kicks off an automated build process which typically includes testing and (if your commit hasn't broken anything) might include deployment too.

In general, integrating your [Postman tests](#) with your favorite continuous integration service is the same process if you're [running on Jenkins](#), Travis CI, AppVeyor, or any other build system. You will set up your CI configuration to run a shell command upon kicking off your build. The command is a [Newman script that runs your collection](#) with the tests, returning a pass or fail exit code that's logged in your CI system.

In this example, we'll walk through how to integrate Postman with [Travis CI](#), a continuous integration service used to build and test projects hosted on GitHub. Travis CI will run your tests every time you commit to your GitHub repo, submit a pull request, or some other specified configuration.



## Before we get started:

[▶ Run in Postman](#)

1. **Start with a Postman collection with tests:** For now, let's assume you already have a Postman collection with tests. Download the sample collection and environment by clicking the Run in Postman button if you want to follow along with this example.
2. **Set up a GitHub repository:** Travis CI is free for open source projects on GitHub, so in this example, we will keep our Postman tests in a public GitHub repo.
3. **Set up Travis CI:** Getting started with Travis CI is simple and will take a few minutes. Follow the [Travis CI getting started guide](#) for the complete walk through. [Sign in to Travis CI](#) with your GitHub account. Go to your [profile page](#) and enable Travis CI for the

public GitHub repo we set up in the previous step.

## Hooking up Postman to Travis CI

1. Export the Postman Collection as a JSON file and move the file to your project directory.

If you're using an environment like in this example, [download the Postman environment as a JSON file](#) and move the file to your project directory as well. In this example, I've moved both files into a directory called tests placed in the root of the project repository. Remember to add and commit these 2 files to your repo.

```
~/repos/myExampleCodebase $ [master L|+ 1]
[13:26 $ tree
.
└── tests
    ├── bitcoinz.postman_collection.json
    └── tests.postman_environment.json
```

2. Create a new file called .travis.yml and move it to the root of your project repository.

Remember to add and commit it to your repo. This file tells Travis CI the programming language for your project and how to build it. Any step of the build [can be customized](#). These scripts will execute the next time you commit and push a change to your repo.

```
.
├── .travis.yml
└── tests
    ├── bitcoinz.postman_collection.json
    └── tests.postman_environment.json
```

In the .travis.yml file, add a command to install Newman in the CI environment, and then add a script telling Newman to run the Postman tests (which I've placed in the tests directory). Since Travis CI doesn't know where Newman is located, let's update the PATH. In this node.js example, the newman tool is located in my .bin directory which is located in my node\_modules directory.

Now, my .travis.yml file looks like this for this node.js example:

```
language: node_js
node_js:
  - "8.2.1"

install:
  - npm install newman

before_script:
  - node --version
  - npm --version
  - node_modules/.bin/newman --version

script:
  - node_modules/.bin/newman run tests/bitcoinz.postman_collection.json -e tests/tests.postman_environment.json
```

Travis CI is now set up to run your Postman tests every time you trigger a build, for example, by pushing a commit to your repo.

Let's try it out. The [Travis CI build status page](#) will show if the build passes or fails:

```
491
492 The command "node_modules/.bin/newman run tests/bitcoinz.postman_collection.json -e
      tests/tests.postman_environment.json" exited with 1.
493
494 Done. Your build exited with 1.
```

Travis CI is running our Newman command, but we see a failed exit code (1). Boo.

Stay calm. Let's review the logs in Travis CI. Newman ran our tests, we see the first and second tests passed, but the last test Updated in the last day failed.

## Integration with Travis CI

```
457
458 → Get Bitcoin Exchange Rate
459   GET https://api.coindesk.com/v1/bpi/currentprice.json [200 OK, 1.15KB, 91ms]
460   ✓ Status code is 200
461   ✓ Bitcoin rate is a float
462   1. Updated in the last day
463
464
465 |           | executed | failed |
466 |       iterations |    1 |     0 |
467 |       requests |    1 |     0 |
468 |       test-scripts |    1 |     0 |
469 |       prerequest-scripts |    1 |     0 |
470 |       assertions |    3 |     1 |
471 |   total run duration: 271ms
472 |   total data received: 672B (approx)
473 |   average response time: 91ms
474
475
476
477
478
479
480
481
482
483
484 # failure      detail
485
486 1. AssertionFailure Updated in the last day
487         at assertion:3 in test-script
488         inside "Get Bitcoin Exchange Rate"
489
490
491 The command "node_modules/.bin/newman run tests/bitcoinz.postman_collection.json -e tests/tests.postman_environment.json" exited with
492 1.
493 Done. Your build exited with 1.
```

Top ↵

Let's go back to our Postman collection and fix our Updated in the last day test.

The screenshot shows the Postman application interface. At the top, there are tabs for Runner, Import, Builder, Team Library, and joycifer. Below the tabs, there's a search bar with the text 'tests' and a dropdown menu labeled 'Examples (0)'. The main workspace shows a collection named 'Get Bitcoin Exchange' with one item: 'Get Bitcoin Exchange Rate'. The item details show a GET request to 'https://api.coindesk.com/v1/bpi/currentprice.json'. The 'Tests' tab is currently selected, displaying the following code:

```
1 // ensure healthy response code
2 tests["Status code is 200"] = responseCode.code === 200;
3
4 // parse the response object
5 var response = JSON.parse(responseBody);
6
7 // pull key values from the response
8 var usdRate = response.bpi.USD.rate;
9 var bitcoinTime = response.time.updatedISO;
10
11 // ensure the rate in the response is a float, and not divisible evenly by 1
12 tests["Bitcoin rate is a float"] = usdRate % 1 !== 0;
13
14 // get environment variable from the pre-request script
15 var lastDay = postman.getEnvironmentVariable('lastDay');
16
17 // Oops! There is an error in this test. Fix it so that `<=` is `>=`
18 // to ensure the response was updated in the last day
19 tests["Updated in the last day"] = bitcoinTime <= lastDay;
```

A snippet of code from the 'SNIPPETS' panel is shown on the right side of the interface:

- Clear a global variable
- Clear an environment variable
- Response body: Contains string
- Response body: Convert XML body to a JSON Object
- Response body: Is equal to a string
- Response body: JSON value check
- Response headers: Content-Type header check
- Response time is less than 200ms
- Set a global variable
- Set an environment variable
- Status code: Code is 200

Once we fix the mistake in our test, let's save the changes, update the repo with the latest collection file, and then trigger a Travis CI build once again by committing and pushing the change.

```
454 $ node_modules/.bin/newman run tests/bitcoinz.postman_collection_correct.json -e tests/tests.postman_environment.json      3.81s
455 newman
456
457 bitcoinz
458
459 → Get Bitcoin Exchange Rate
460   GET https://api.coindesk.com/v1/bpi/currentprice.json [200 OK, 1.15KB, 107ms]
461     ✓ Status code is 200
462     ✓ Bitcoin rate is a float
463     ✓ Updated in the last day
464
465   ┌─────────────────┐ ┌──────────┐ ┌────────┐
466   |                 | | executed | | failed |
467   └────────────────┘ |           | |       |
468   |   iterations   | |   1   | |   0   |
469   |                 | |           | |       |
470   |   requests    | |   1   | |   0   |
471   |                 | |           | |       |
472   |   test-scripts| |   1   | |   0   |
473   |                 | |           | |       |
474   |   prerequest-scripts| |   1   | |   0   |
475   |                 | |           | |       |
476   |   assertions   | |   3   | |   0   |
477   |                 | |           | |       |
478   | total run duration: 275ms
479   |                 |
480   | total data received: 673B (approx)
481   |                 |
482   | average response time: 107ms
483   └─────────────────┘ └──────────┘ └────────┘
484
485
486 The command "node_modules/.bin/newman run tests/bitcoinz.postman_collection_correct.json -e tests/tests.postman_environment.json"
487 exited with 0.
488 Done. Your build exited with 0.
```

And it's working! All our tests passed and the command exited with a successful exit code (0).

# Newman with Docker

- Newman

## For Mac and Ubuntu

To run [Newman in Docker](#),

1. Go to Docker Hub and pull your copy [here](#).
2. Ensure you have Docker installed and running in your system. Docker has extensive installation guideline for popular operating systems. Choose your operating system and follow the instructions. A quick test to see if docker is installed correctly is to execute the following command, ensuring that it runs without errors.

```
$ docker run hello-world
```

1. Pull the Newman docker image.

```
$ docker pull postman/newman_ubuntu1404
```

1. Run Newman commands on the image.

```
$ docker run -t postman/newman_ubuntu1404 --url="https://www.getpostman.com/collection  
s/8a0c9bc08f062d12dcda"
```

At this stage, you should see Newman running the collection and the output being visible on the terminal. The entrypoint to the docker image is Newman, and as such, all command line parameters of Newman can be used here. You can also run locally stored collection files. The README of the image outlines the procedure of mounting shared data volumes to achieve this.

## For Windows

Check out our [blog post](#) on how to run Newman in Docker for Windows.

# Setting up a team library

- Pro
  - Enterprise

Postman Pro and Enterprise users can use the Team Library to enhance collaboration among teammates. This feature enhances collaboration among teammates in several ways.

- **Share collections** You can [share collections](#) with your team. Collections contain groups of API requests you can organize, add descriptions, attach test scripts, and build conditional workflows.
- **Share environment templates** Teams can use [environment templates](#) to create and [share a snapshot of a local environment](#). Users might have different environment variable values, so updates to these values are not synced between shared environments.
- **See the activity feed in the Team Library** You can see changes to shared collections in the [activity feed](#).
- **Subscribe to shared collections** When you [subscribe to a collection](#), you get a synced copy of this collection in your Postman app. If you have edit permissions for the collection, you can make changes that other subscribers to the collection can see.

## Setting up the Team Library

To use your Team Library, open the Postman app and click the **Team Library** tab on the header bar.

## Setting up a Team Library

The screenshot shows the Postman application window with the 'Team Library' tab selected. The main area displays a list of API collections:

- [Cloud API] Authority Testing
- [Data] API Gateway (Prediction Scores)
- [Data] API Gateway Analytics
- [Data] Elastic Search Indexes Querying
- [Data] Elastic Search Indexes Querying - Me
- [Data] Marketo APIs
- [Data] SalesForce APIs
- [Data] SalesForce APIs JSON Sandbox
- [Integration-Anuhya] Pagerduty #monitor
- [Integration] - Send mail using Sendgrid
- [Integration] Collection to Dropbox

Each collection entry includes a description, the number of requests and subscribers, and user information (e.g., Ankit Jaggi, View Only). Buttons for 'View in web' and 'Subscribe' are also present.

## New Postman Pro or Enterprise users

If you are visiting the Team Library for the first time, you see this screen.

The screenshot shows the Postman application window with the 'Team Library' tab selected. A central message reads:

**Collaborate on your APIs with your entire team**  
With the Postman Pro trial, you can now collaborate on your API collections with your teammates.  
Select the collections you want to share and invite your team to get started. [Learn more about Pro](#)

The interface is divided into two main sections:

- 1 Choose which collections to share**: A list of collections to be shared:
  - All your collections
  - Postman Echo
- 2 Invite Users**: A field to enter user emails:
  - + (button to add users)

A button at the bottom center says "Activate your trial and share these collections".

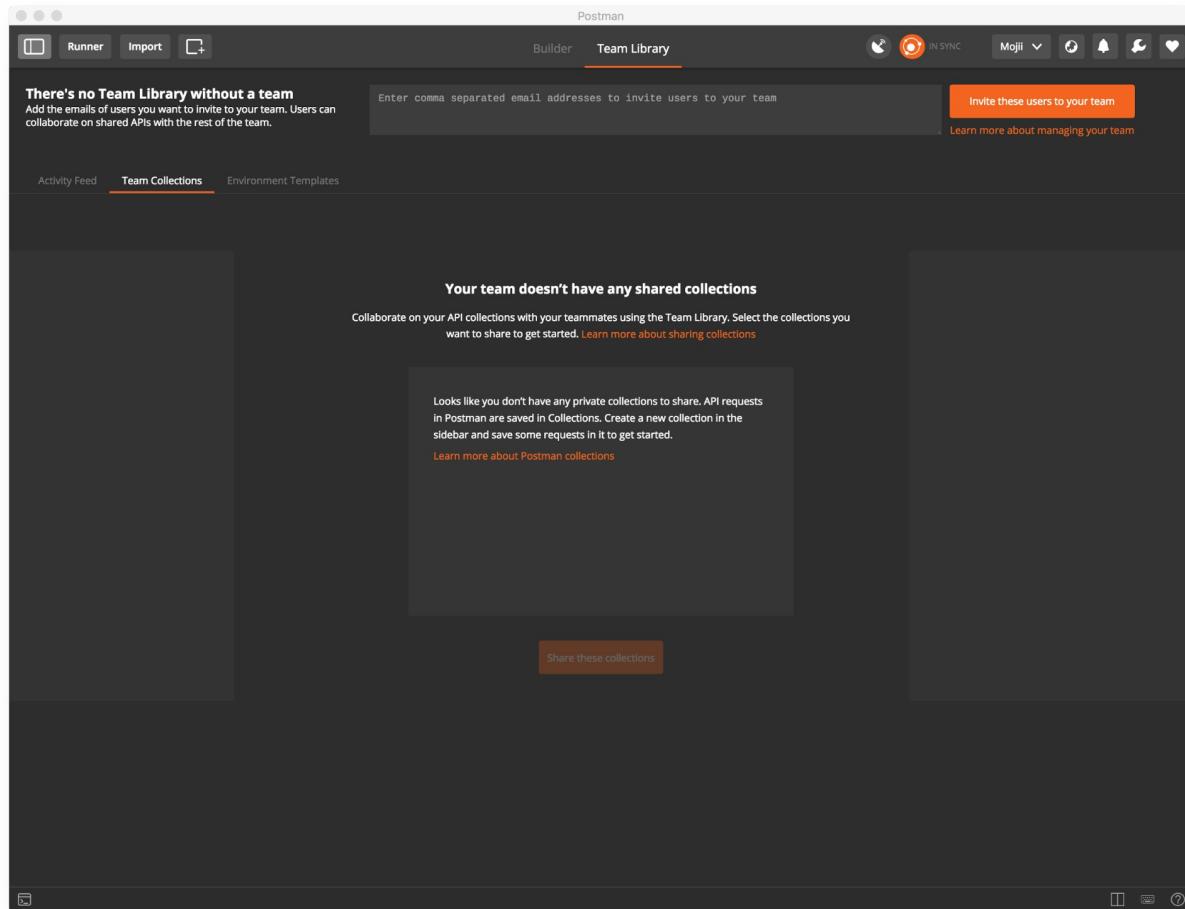
To set up your Team Library:

1. Select the API requests you want to share and save them in collections.
2. Select the collections you want to share.
3. Add the email addresses of the users you want to invite to your Postman Pro team.
4. Click the **Activate your trial and share these collections** button.

When you share your first collection, your 7-day Postman Pro trial is activated.

## Existing Postman Pro or Enterprise users

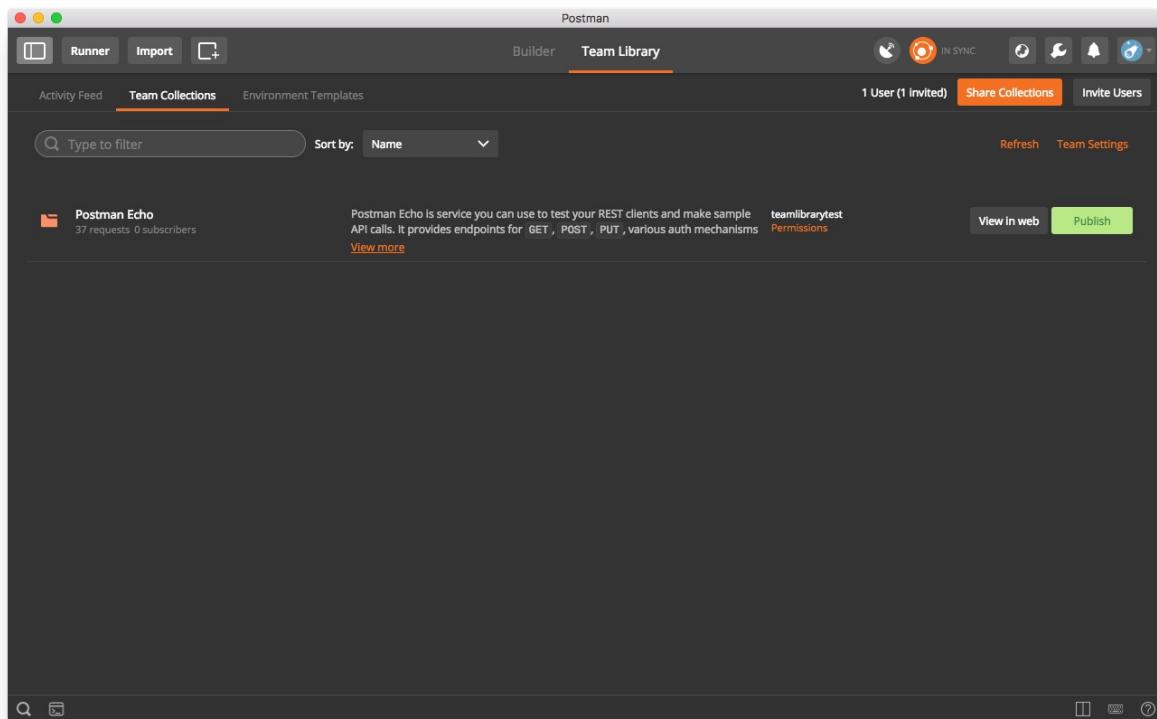
If you are on a Postman [Pro](#) or [Enterprise](#) team, or if you have activated your 7-day Postman Pro trial, you see this screen when you click the **Team Library** tab for the first time.



To set up your Team Library:

1. Select the collections you want to share, and click the **Share these collections** button.
2. Add the email addresses of the users you want to invite in the input field on top of the screen.
3. Click the **Invite these users to your team** button to share collections and collaborate with your team more efficiently.

## Setting up a Team Library

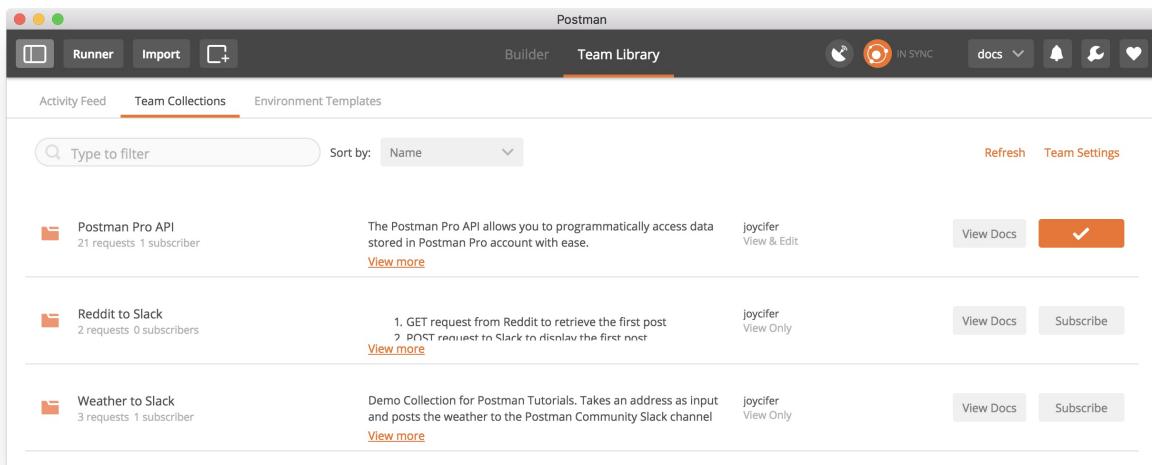


# Sharing

- Pro
  - Enterprise

Postman Pro and Enterprise users have access to a Team Library which lets you collaborate faster with your teammates. Team members can share collections and environment templates and see the activity feed in the Team Library. You can think of the Team Library as a way to organize your API infrastructure and make finding API documentation, workflows, and test suites easy.

Your Team Library should be the single source of truth about your APIs. It will let you see the state of your APIs in real time, or review historical versions and the latest updates.



The Team Library allows team members to subscribe to shared collections. When someone subscribes to a collection, they get a synced copy of this collection in their Postman app. If they have edit permissions for the collection, they can make changes which will be reflected in everyone else's collection copy too. Changes made to shared collections are visible under the [Activity Feed](#).

Environment templates work slightly differently. Through an environment template, you can create and share a snapshot of a local environment. Users may have different environment variable values, so updates to these values are not synced between shared environments.

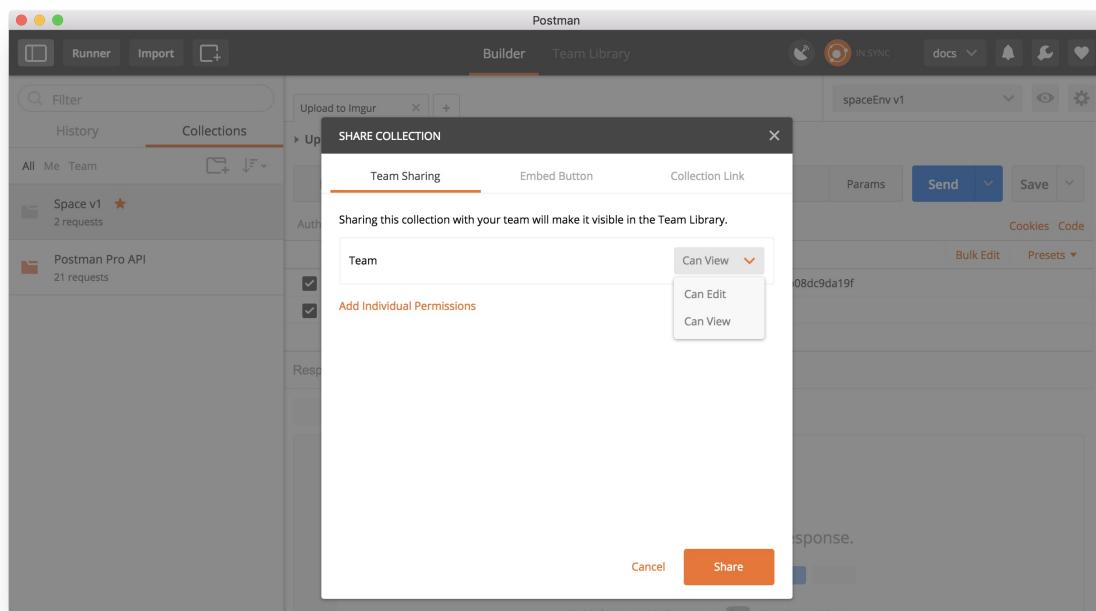
There are a number of enhancements coming up soon on [Postman's product roadmap](#) that will impact workflows for teams wishing to 'sync' updates to their shared environment templates. In the meantime, here are some workarounds that rely on collections as the single source of truth for your APIs:

1. Run a pre-request script to validate the correctness of required environment keys and values. If any are missing, stop the workflow and throw an error.
2. Set environment variables in the first request of the collection, or a pre-request script.

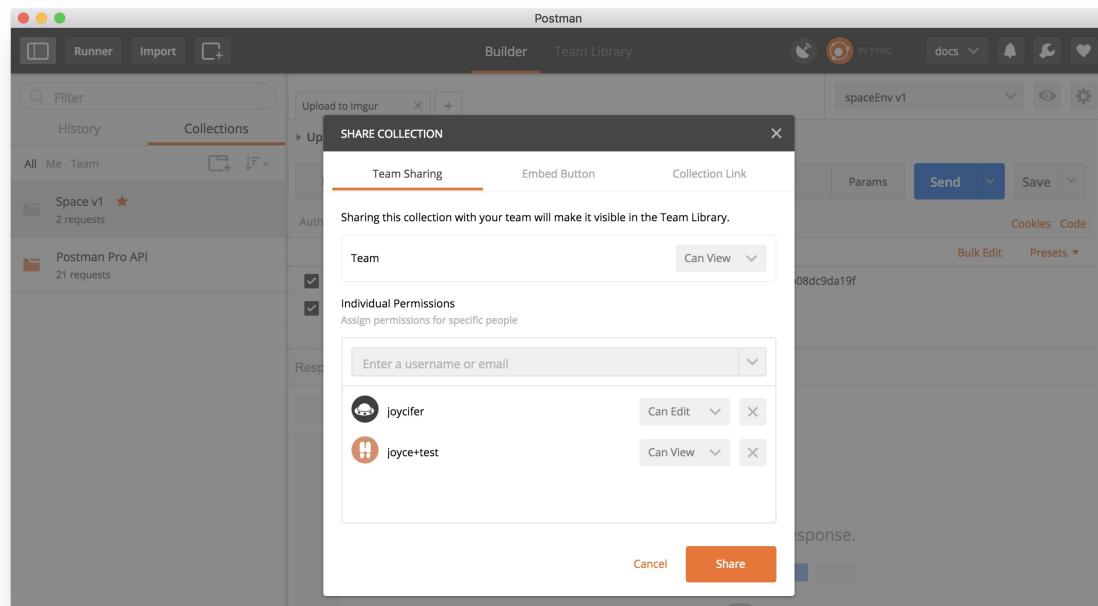
## Sharing Collections

In addition to the [standard ways to share a collection](#), Postman Pro and Enterprise users can also share collections with their team or specific team members.

1. From the Collections tab in the sidebar, click on the ellipses (...) next to the collection you would like to share, and select “Share”.
2. Specify the team permissions by selecting “Can View” or “Can Edit” in the dropdown.



3. Postman also lets you manage access to a collection at an individual level. To do this, click the **Add Individual Permissions** link. Select a team member from the dropdown, or enter the username or email of the person you want to share with. Specify the individual's permissions by selecting “Can View” or “Can Edit” in the dropdown. You can give permissions to multiple people.

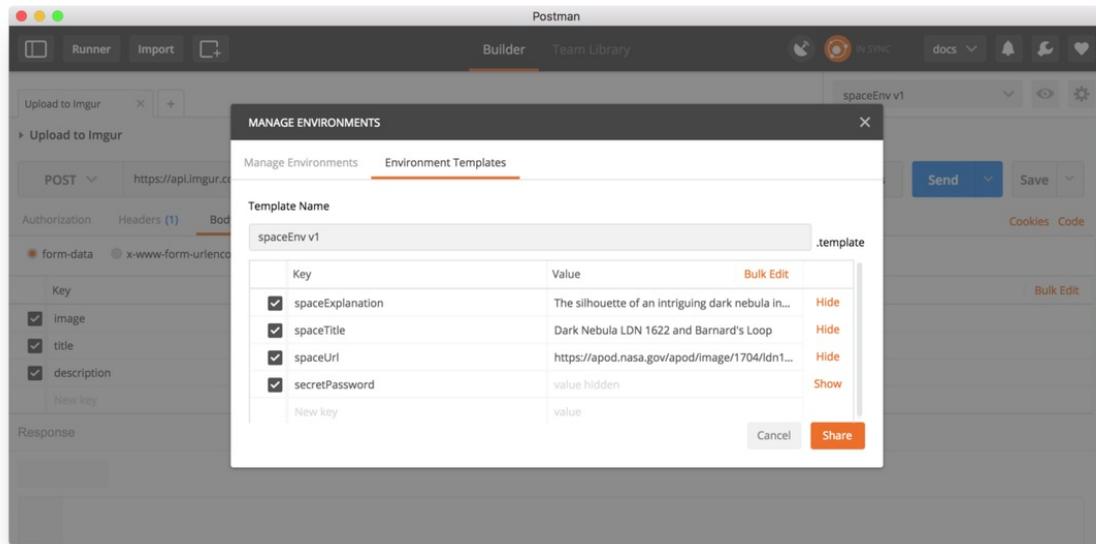


4. Click the **Share** button to complete the process.
5. Go to the Team Library to view the full list of team collections.

## Sharing environments

In addition to the [standard way to share an environment](#), Postman Pro and Enterprise users can also share an environment template with their team.

1. From the gear icon in the upper right corner of the Postman app, select “Manage Environments”, and click the orange **Share** button next to the environment you want to share.
2. You will have one last opportunity to hide any sensitive values like passwords and access tokens before sharing the environment template. When someone else imports the environment, or accesses the shared template, they can input their own personal information within their own version of the template.



3. Click the **Share** button to complete the process.
4. Go to the Team Library to view the full list of team environment templates.

# Activity Feed and restoring collections

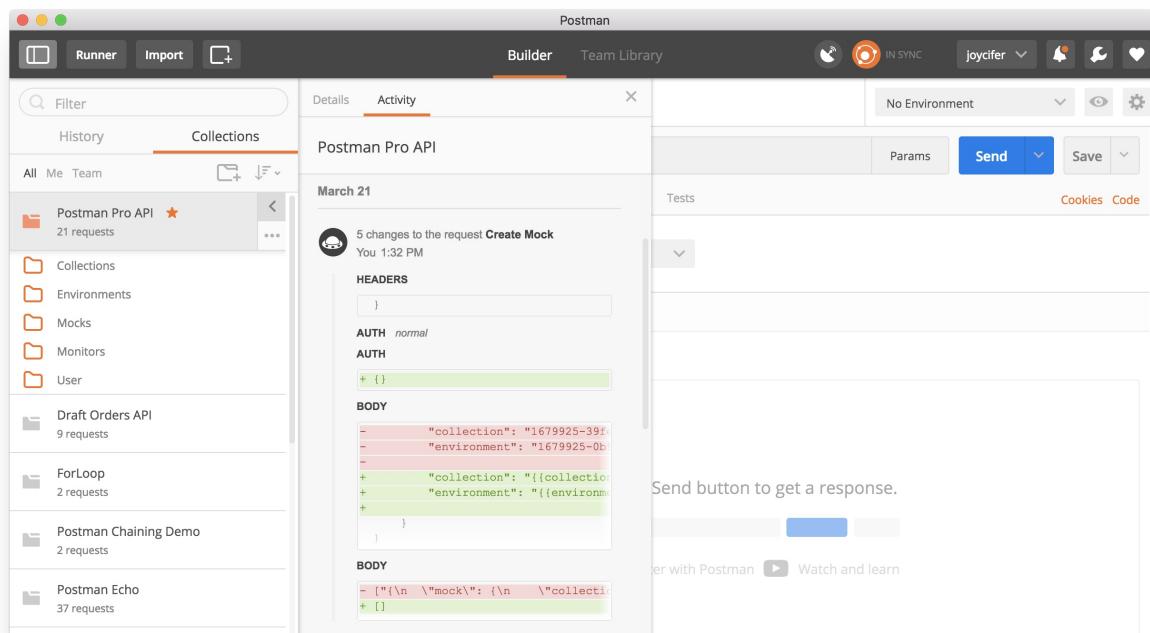
An activity feed is a list of events, displaying to the user in an interactive interface, updates to their Postman data. It is useful to keep track of changes made to your private and team collections by different users across the team. The activity feed also lets you rollback a collection, and restore it to any previous point in time.

## Types of activity feed

Postman tracks changes made at 3 different levels: collection, team and user.

### Collection

To review the activity feed at the collection level, expand the angle bracket (>) next to the collection name to open the collection details view. Under the **Activity** tab, review a chronological listing of activities affecting the collection. The activity feed displays who updated the collection, what the updates were, and when they were completed. Consecutive updates on the same session to the same entity are collated for readability.



### Team

To review the activity feed at the Postman Pro or Enterprise team level, go to the **Activity Feed** tab within the **Team Library** view. Review a chronological listing of activities affecting all collections shared with the team. Like the feed at the collection level, the team activity

feed displays who made changes, to what, and when it was completed.

The screenshot shows the Postman application window with the 'Activity Feed' tab selected. The feed lists activity items:

- 1 change to the request **Upload image to Imgur** in a collection joycifer, 10:57 AM  
HEADERS:

```
- Authorization: Client - ID {  
-  
+ Authorization: Bearer {  
+  
{  
- imgurClientId  
-  
+ imgurRefreshToken  
+  
}  
}
```
- 1 change to the request **Get Imgur auth token** in a collection joycifer, 10:48 AM  
NAME: Retrieve Imgur auth token to Get Imgur auth token
- May 23
- 1 change to the collection **Space v1**

## User

To review the activity feed at the user level, log in and go to the **Dashboard** tab on the [Postman Website](#). The user activity feed includes a list of changes to collections that you own and are subscribed to.

The screenshot shows the Postman website dashboard with the 'Activity Feed' section. It displays activity items:

- You are currently on the Pro plan with 60 people  
41 active users, 17 invites, 3 pending invites  
[Manage Pro plan](#)
- Activity Feed Track your collections & subscriptions  
27 Apr, 2017
  - SamvelRaja Sakthivel deleted a environment: samvel.env.template  
3:28 AM
  - SamvelRaja Sakthivel deleted a collection Response type Mocks  
3:19 AM
  - SamvelRaja Sakthivel deleted a collection Hawk test  
3:19 AM
  - Ankit Jaggi shared a collection Dummy Collection  
3:05 AM
- 26 Apr, 2017
  - Kunal Nagpal deleted a environment Bitbucket Pipelines Run Status.template  
2:02 AM
  - Kunal Nagpal created a environment Bitbucket Pipelines Run status.template  
2:02 AM

## Breaking the event down

The activity feed captures different updates that are made to collections. This includes CRUD operations (Create, Read, Update, Delete) on collection, folders, and requests, among other activities.



## 8 changes to the request **Get details**

You just now

**METHOD** *POST* to *PUT*

### URL

```
http://foo.com/get/{{id}}
```

### HEADERS

```
- header: value
+ Content-Type: application / json
```

**BODY** *params to raw*

### BODY

```
+ {
+     "foo": "bar"
+ }
```

### BODY

```
- [ {
-     "key": "key",
-     "value": "alue",
-     "type": "text",
-     "enabled": true
-   }]
+ []
```

### SCRIPTS

```
- postman.clearGlobalVariable("variable"
+ postman.setGlobalVariable("id", "100")
```

### TESTS

```
- postman.clearGlobalVariable("variable"
+ postman.clearEnvironmentVariable("id")
```

A consolidated update to a request, for example, looks like the screenshot above providing you with a diff view to pinpoint the exact changes.

**Note:** Postman Pro and Enterprise users can view diffs. Other users will be able to track the “who” and “when” in the activity feeds, but not the “what” at this level of detail.

## Restoring collections

Postman Pro and Enterprise users also have the ability within the collection-level activity feed to restore their collections to a point in time.

## February 8



1 change to this collection

You 6:39 AM Restore

### DESCRIPTION

```
- target = "_blank" > < img src = "ht  
-  
+ target = "_blank" > < img src = "ht  
+
```

The Spotify playlist generator coll

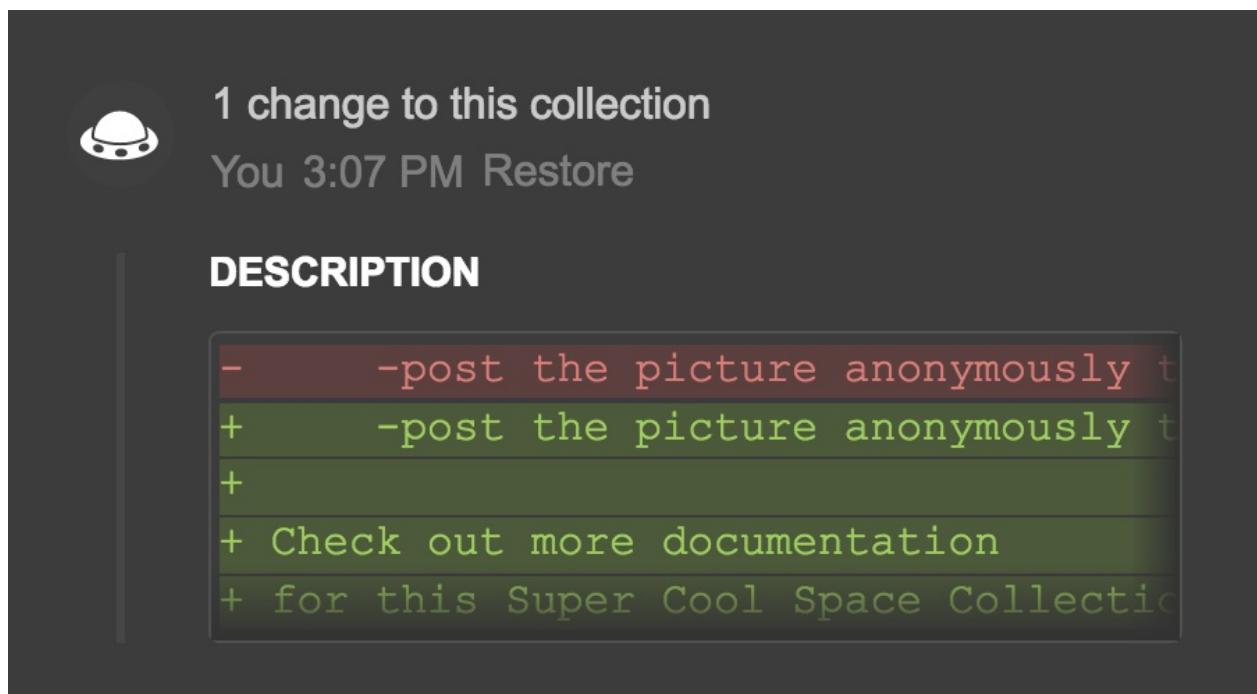
#### ###Workflow

```
< a href = "https://docs.go  
target = "_blank" >  
< img src = "https://docs.g
```

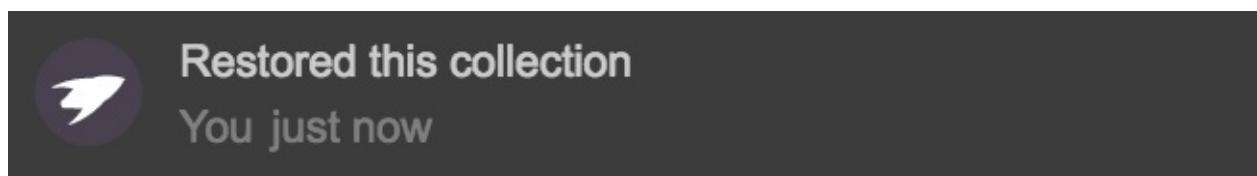
```
1. Take a list of artists a  
s top tracks] (https://deve  
3. Add these related artist
```

#### ### Setup the Environment

Hovering over the activity will display a **Restore** link. Clicking the **Restore** link will restore the collection to the point right *after* this change was applied.



And now the top of the activity feed displays the confirmation of restoring the collection.



## Connecting to Slack, HipChat and other platforms

Postman Pro and Enterprise users can pipe the team's activity feed to a communication channel of your choosing with the following integrations:

- Postman Pro to Slack integration
- Postman Pro to HipChat integration
- Postman Pro to Microsoft Teams integration

## Upcoming additions to the activity feed

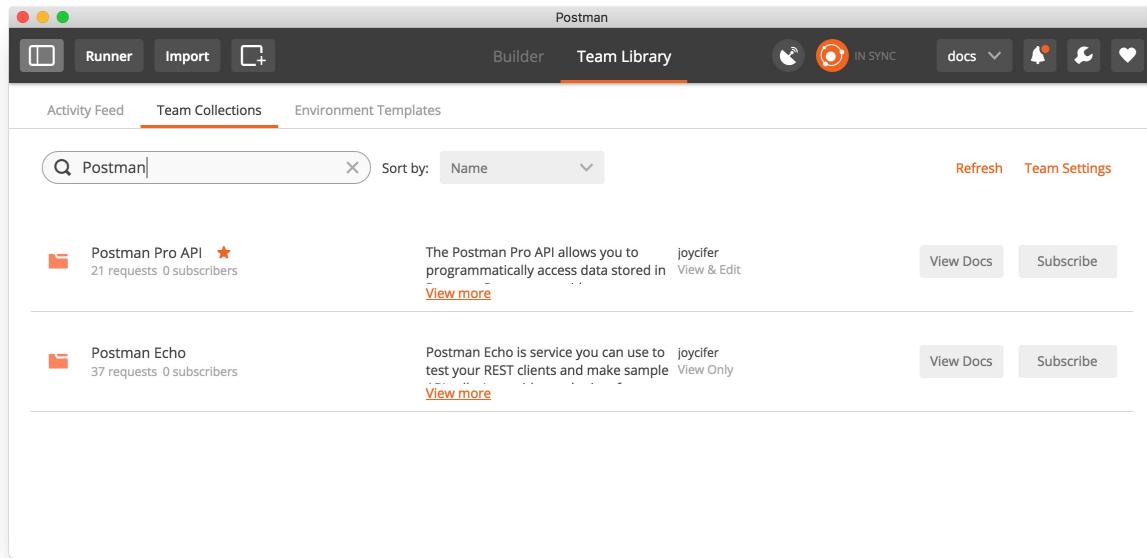
The activity feed will become even more powerful in the near future. It'll integrate with all our Postman Pro and Enterprise products and other features in the pipeline. We're going to soon add **Taggingsupport**. You'll be able to tag collections at a point in time, allowing easier referencing and ability to rollback, while also allowing you to set up Monitors and Documentation on tagged versions of the collection. These version control constructs will allow your team even more flexibility with your API code development within Postman Pro or Enterprise.



# Searching

## Filtering by name or description

To quickly filter the list of collections in your library, begin typing the collection name or description in the search input field.

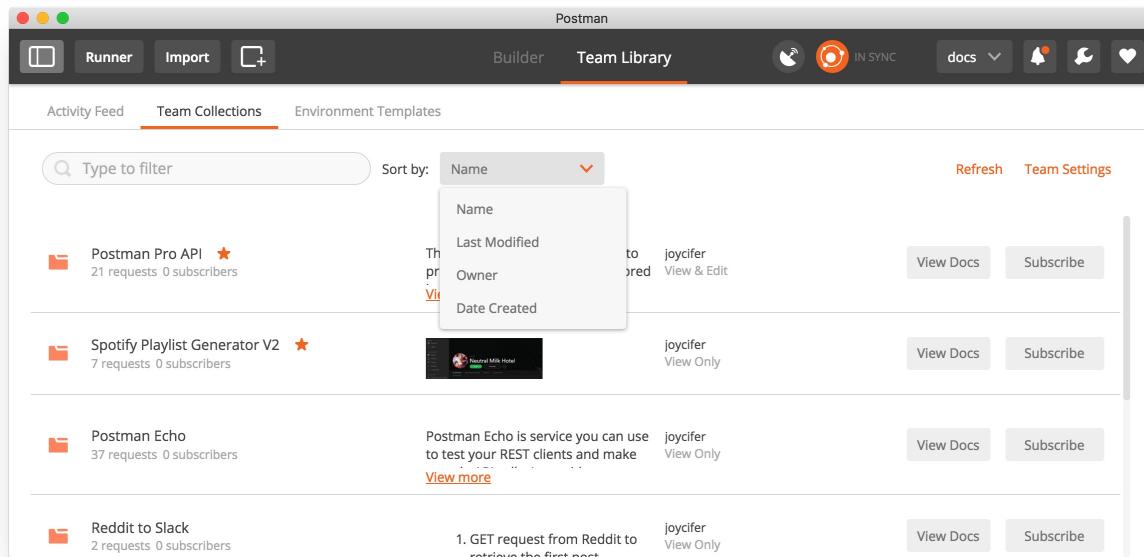


## Favoriting

You can favorite collections in the Team Library like you do in the **Collections** sidebar. Click on the star icon to bring the collection(s) to the top of the list. Favorited collections from both places will always be listed at the top.

## Sorting

You can re-order collections in the Team Library by Name, Last Modified, Owner, or by Date Created. Remember that collections that you have favorited will be at the top but now sorted by the criteria you selected.



## Searching through the Slack Bot

You can integrate the Postman Pro Search Slack bot and get more detailed search results too. This feature will be available in the Team Library soon. Read more about the [Slack Bot](#).

# Conflicts

Postman's conflict management allows you to work collaboratively even while you are offline. Your work will be safely synced when you come online without any accidental changes to your teammates' work. While syncing your changes, if you made a change that might overwrite your teammates' work, the **RESOLVE SYNC CONFLICTS** modal will prompt you to choose which conflicting version to keep.

## Example of managing conflicts

Imagine you share a collection named "My API" with your team and permit your teammate, Bob, to edit the collection. Bob subscribes to this collection from the Team Library and starts working with this collection.

Now, while you are offline, Bob is online and renames the collection to "Team API" and then renames the same collection to "Postman API". Now when you come online and start syncing your changes, Postman will detect that there is a conflicting change made to the same data. Postman will display the **RESOLVE SYNC CONFLICTS** modal. You can review the conflicting versions and choose which version to keep and which to discard.

### RESOLVE SYNC CONFLICTS

Item Type	Item Id / Item Name	Property Name	<input type="checkbox"/> Server Value	<input checked="" type="checkbox"/> Local Value
collection	Postman API	name	<input type="checkbox"/>	Updated to: Team API
			<input type="checkbox"/>	Updated to: Postman API

**Resync**

In the **RESOLVE SYNC CONFLICTS** modal, “Local Value” is the version currently on your local app and “Server Value” shows the version synced to Postman. To keep your changes and overwrite Bob’s changes, check the “Local Value” and click the orange **Resync** button. Or to discard your changes and keep Bob’s change, check “Server Value” and click the orange **Resync** button. Resyncing will update your local data and complete the sync to Postman.

# Intro to API documentation

Postman's API documentation feature allows you to share public or private API documentation, beautifully viewable via a web page.

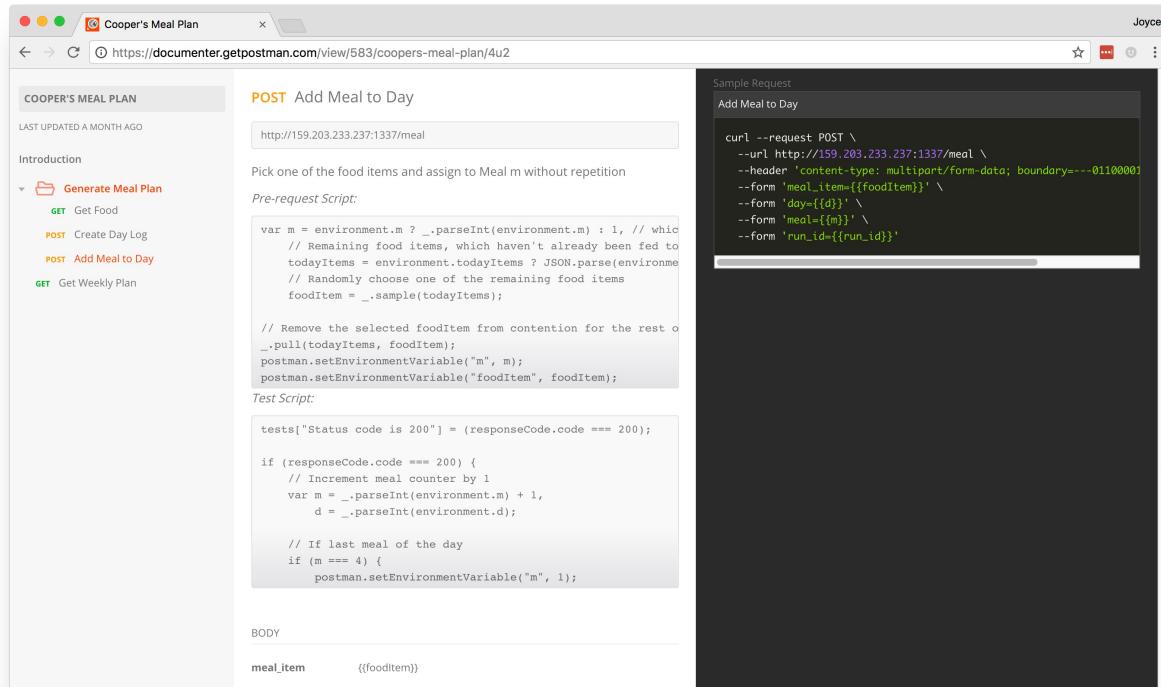
Postman generates and hosts browser-based API documentation for your collections automatically in real-time. Each collection has a private and public documentation view, generated in real-time using the data synced to our servers. In order to [access the private view](#), click “View in web” in the Postman app or in the “Team Library”. The public view is accessible via the public link, generated when you [publish your documentation](#). This link will be displayed right after your documentation is published, and is also accessible via the “Published” dropdown in the private documentation view.

## What gets automatically generated?

Documentation for your API includes the following:

- Sample requests, headers, and other metadata
- Descriptions associated with requests, folders, and collections
- Generated code snippets in some of the most popular programming languages

Documentation is organized into sections that reflect the structure of your collection, by ordered requests and folders. Descriptions can be customized using [Markdown](#) styling with embedded graphics to complement your documentation. GitHub flavored markdown is also supported, so you can even include tables. When including block elements, ensure that you leave an empty line before and after to avoid any rendering issues.



## Free documentation views with your Postman account

Your Postman account gives you a limited number of free documentation views per month. You can check your usage limits through the [Postman Pro API](#) or the [account usage page](#).

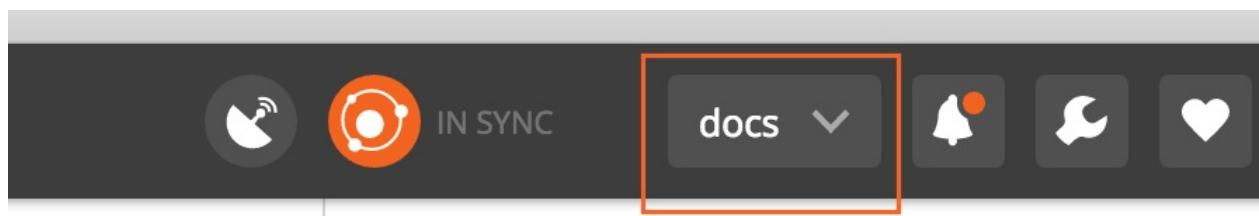
# Viewing documentation

Each collection has a private and public documentation view, generated in real-time using the data synced to our servers.

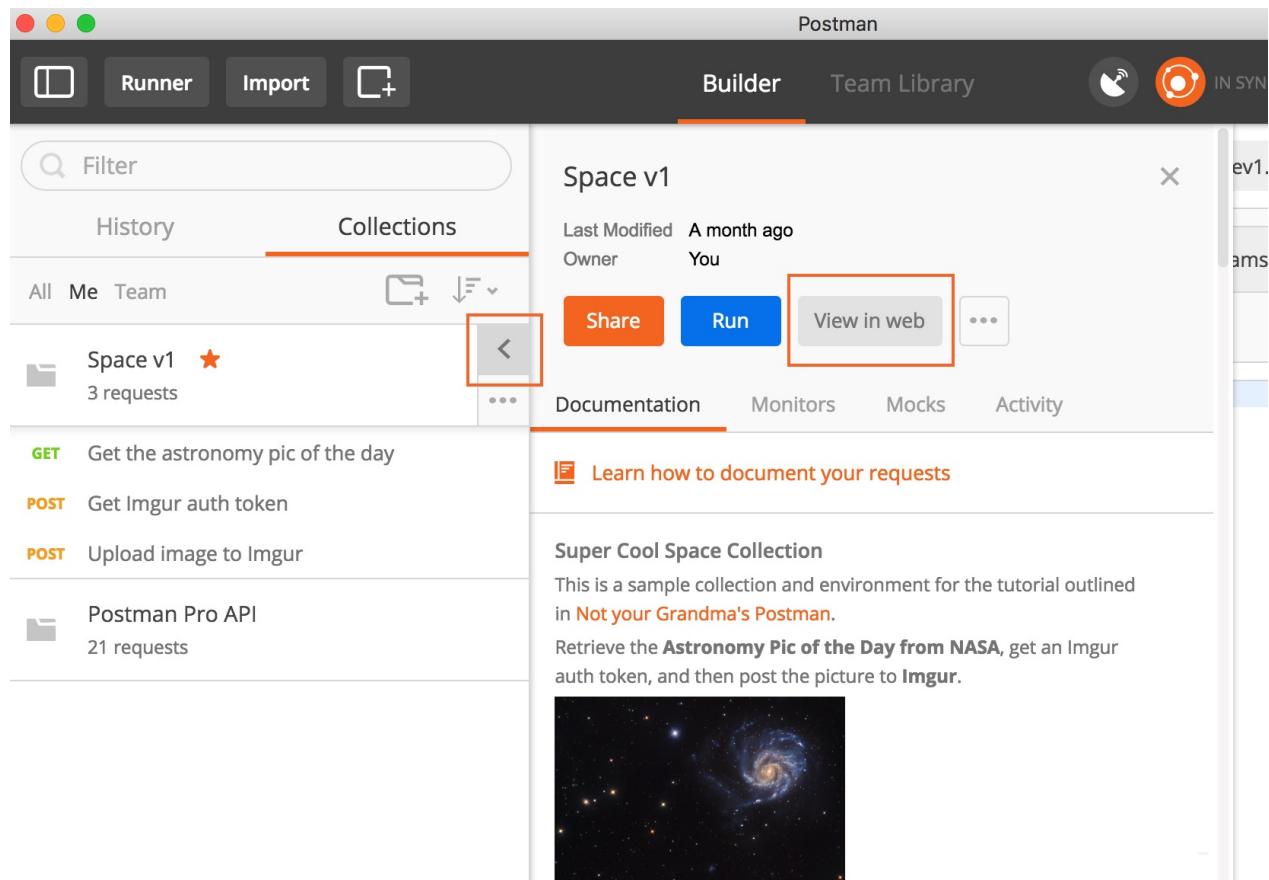
## Viewing Private Documentation

In order to view private documentation, which is only accessible to you and your Postman Pro or Enterprise team, click “View Docs” or “View in web” in the Postman app or in the [Postman web view](#).

Private documentation is available to all Postman users. To view private documentation for a collection, make sure you’re signed in to the Postman app. When signed in, your username will be displayed in the top-right corner.



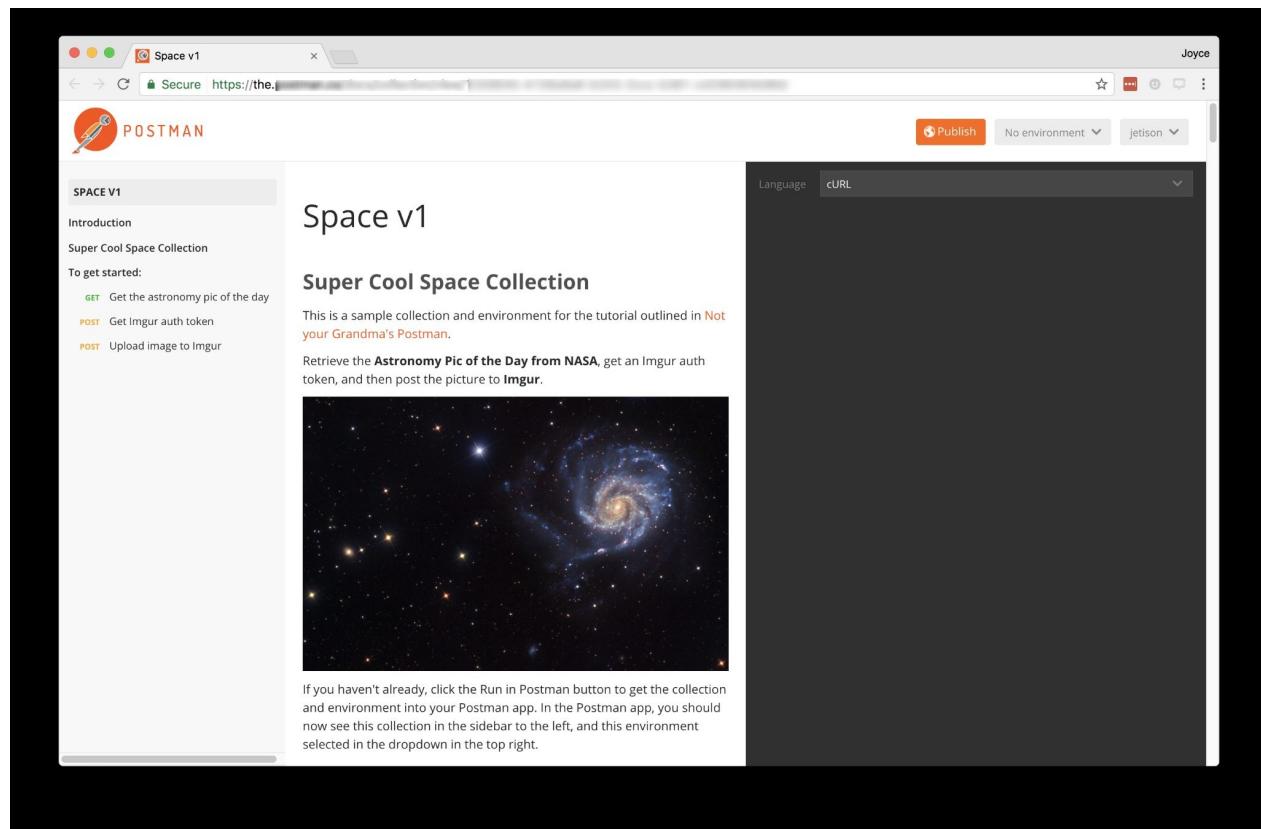
In the Postman app, expand the right angle bracket (>) next to any collection name to open the collections details view, and then click the **View in web** button.



To view documentation for your team's collections in the Postman app, you'll see a [View Docs](#) button in your [Team Library](#).

Space v1 ★ 3 requests 0 subscribers	Super Cool Space Collection This is a sample collection and environment for the tutorial <a href="#">View more</a>	joycifer Permissions	<a href="#">View Docs</a> (highlighted with a red box)	Published
Postman Echo 37 requests 0 subscribers	Postman Echo is service you can use to test your REST clients and make sample API calls. It provides endpoints for <code>GET</code> , <a href="#">View more</a>	joycifer Permissions	<a href="#">View Docs</a>	Published
Postman Pro API ★ 21 requests 0 subscribers	The Postman Pro API allows you to programmatically access data stored in Postman Pro account with ease. <a href="#">View more</a>	joycifer Permissions	<a href="#">View Docs</a>	Publish

Clicking this button will open the documentation for that collection in your browser. You'll be able to browse all folders and requests in the collection, and see what the requests and code snippets look like with different environments selected.



Keep in mind that this view is restricted to users in your Postman Pro and Enterprise team. This link will not work for anyone who does not have access to the collection itself. If the collection is shared with your team, anyone in the team can view the private documentation for this collection. If your collection is not shared, only you can view the private documentation for this collection.

To make this documentation available to other users, perhaps as a link on your website, you'll need to [publish the documentation](#).

## Viewing Public Documentation

The public view is accessible via the public link, generated when you publish your documentation. Once you've published your documentation, anyone can access it via the public URL. This public URL will be displayed right after your documentation is published, and is also accessible via the **Published** dropdown in the private documentation view.

## Publish Collection Public

### Cooper's Meal Plan

Choose Environment Template

No environment

Custom Domain

None

Public URL

<https://documenter.getpostman.com/view/coopers-meal-plan/2SLrU9AA9c082d>

Show Custom Styling Options

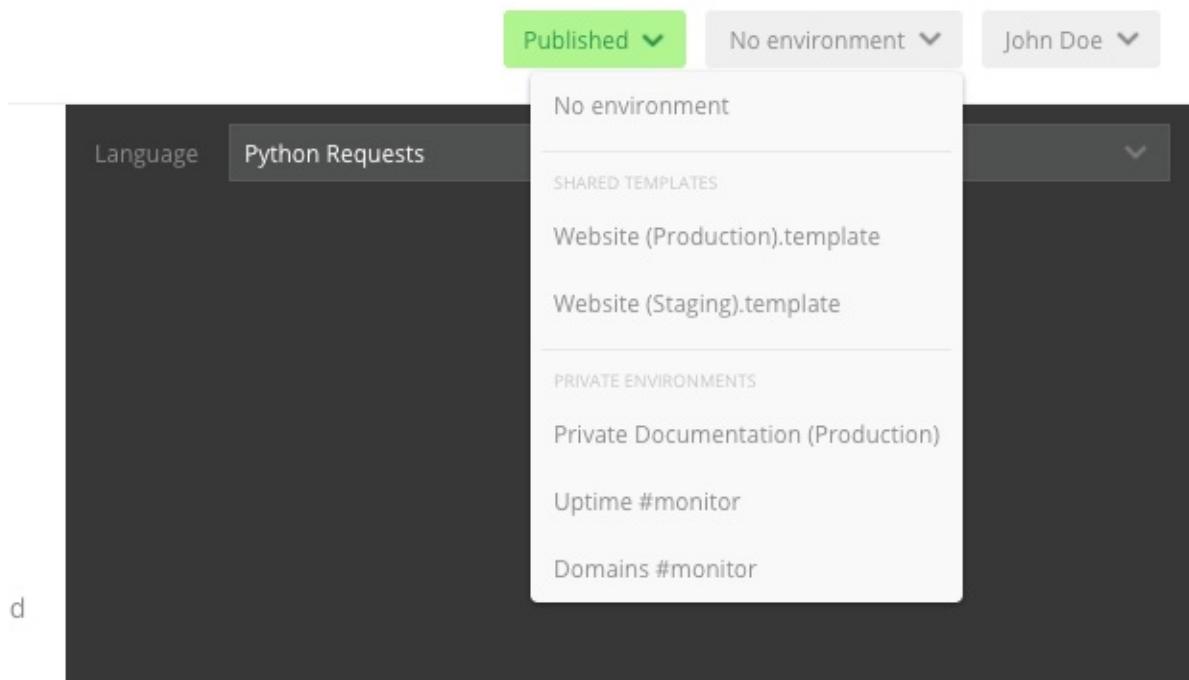
Update

If you choose a verified custom domain, then your public documentation will also be available from your custom domain. For more information about using a custom domain, read about [adding and verifying custom domains](#).

# Environments and environment templates

Your environments and environment templates are automatically synced too and are available through a dropdown in your API documentation. When viewing public or private documentation, selecting an environment will substitute those environment variable values into the relevant parts of the documentation. For example, if the selected environment has a foo variable with the value bar, then all occurrences of in the request will be replaced with bar. All environments are encrypted during storage, for more details, check the Data Security section of our [Security page](#).

## Environments in private documentation



Within the private documentation view, all of your environments and environment templates will be available to you along with environment templates shared in your Team Library.

## Environments in public documentation

# Publish Collection Public

## Cooper's Meal Plan

Choose Environment Template

No environment

No environment

Website (Production).template

Website (Staging).template

Public URL

<https://documenter-beta.getpostman.com/view/coopers-meal-plan/2SLrU9AA9c082d>

Show Custom Styling Options

**Update**

If an environment template was selected while publishing documentation, this will be available to all documentation viewers.

Specifically, if the user is signed into their Postman account, then their synced environments will be available in published documentation too. This allows them to customize your published API documentation to their specific environment.

However, if your public documentation is published on a custom domain, only the environment template will be available in the published page even if the user is signed into their Postman account.

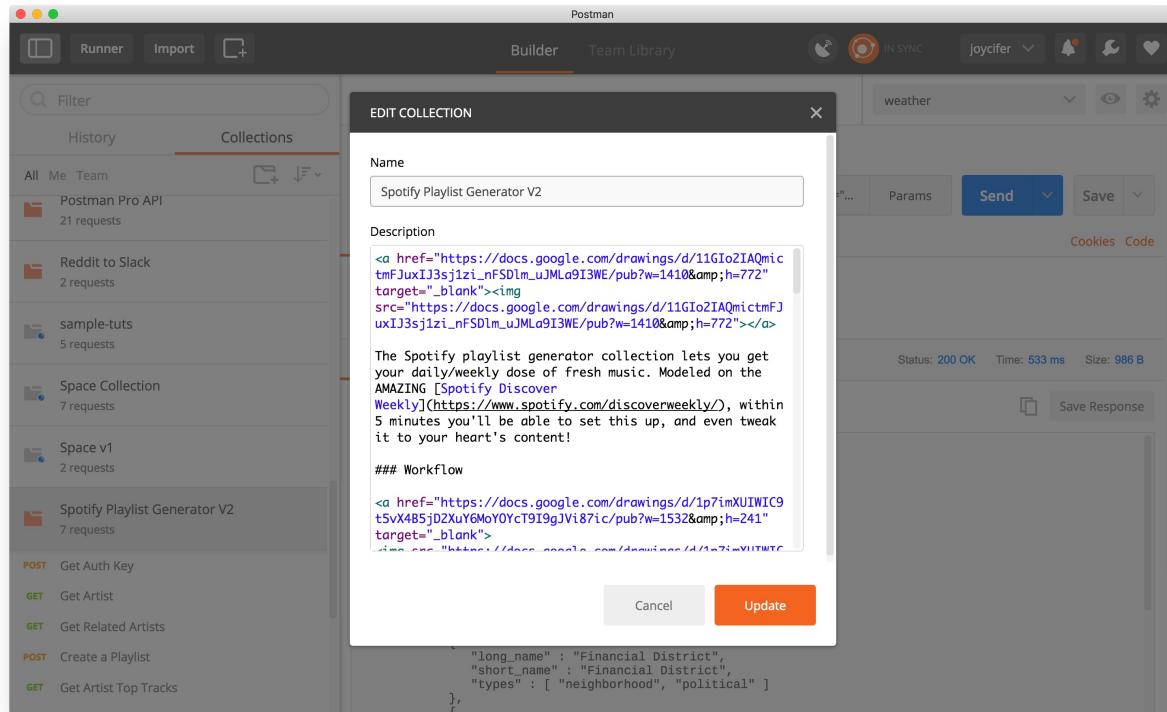
# How to document using Markdown

Review [reference for using Markdown](#) in API documentation. Postman has created a collection intended to test Markdown styling inside Postman or within other services that render Markdown. The descriptions in the collection contain Markdown syntax and some of them have links to HTML pages of their rendered version.

The screenshot shows a Postman collection titled "Markdown in API Documentation" last updated on April 26, 2017. The collection includes an h1 header and sections on basic Markdown elements, Daring Fireball Basic Syntax, and Daring Fireball Extended Syntax. It also contains examples of bolded text, lists, and block quotes. A note at the bottom suggests using <http://markdownlivepreview.com/> to test Markdown. The interface includes a "Run in Postman" button and a "No environment" option.

Postman supports Markdown as a way to style text descriptions for requests, collections, and folders within collections. You can even embed screenshots and other images for more descriptive flair. Learn more about [where Postman renders Markdown](#).

## How to document using Markdown



# Publishing public docs

Published documentation is a way of sharing your API documentation with the world. If you're able to view documentation for one of your collections, publishing docs are a click away! See examples, and read more about [public API documentation](#).

## Generating public documentation

You can only publish documentation for collections that you created or for which you have write permissions. [Open the docs page](#) for one of your collections. Click the **Publish** button at the top of the screen.



If you are signed in to Postman, you can select a corresponding environment with which to publish the collection. Any references to variables like in the collection will be replaced with the correct value from the environment.

A detailed screenshot of the "Publish Collection" dialog. It has a title "Publish Collection" with a "Public" status indicator. Below the title, it says "Postman Echo". There are two dropdown menus: "Choose Environment Template" set to "No environment" and "Custom Domain" set to "None". A large orange "Publish" button is at the bottom. To the right of the dropdowns, there's a yellow callout box with text: "Publishing this collection will make it public to anyone who has the published URL. Make sure any shared environments are ready for consumption before publishing." At the bottom left, there's a link "Show Custom Styling Options".

The public URL field in the screenshot below contains the URL that you can share with the outside world. For example, if you're publishing your primary collection, you might want to select the "Production" environment, so that your documentation is immediately usable for new visitors.

**Publish Collection**  Public

**Postman Echo**

Choose Environment Template

No environment 

Custom Domain

None 

Public URL  
<https://documenter.getpostman.com/view/1559979/postman-echo/6YySu4k>

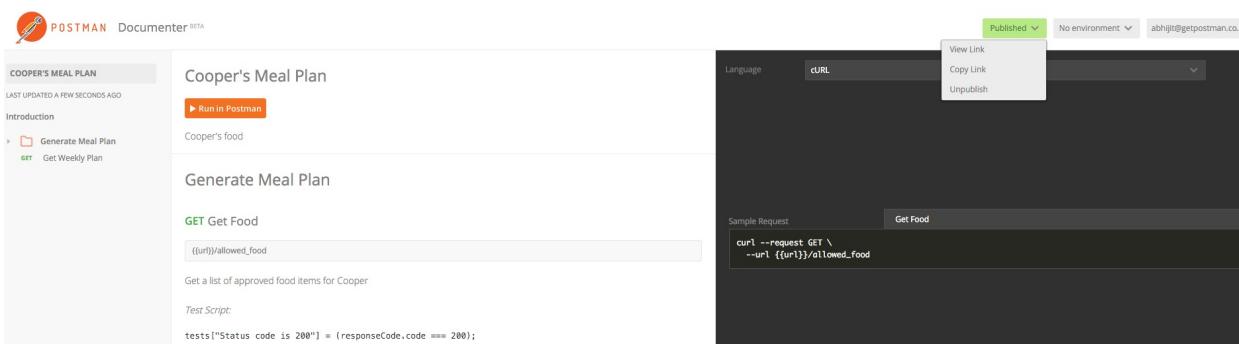
Show Custom Styling Options

**Update**

**PUBLISHING NOTES:** Publishing this collection will make it public to anyone who has the published URL. Make sure any shared environments are ready for consumption before publishing.

**IMPORTANT:** Any confidential info in your environment, such as **passwords and access tokens**, might be visible publicly. Ensure that all such information is removed from the environment before you publish documentation with an environment. The public documentation link will always have the most up-to-date content! You don't need to keep going through the Publish flow each time you want your docs to be updated.

To unpublish, click the View Docs button from your Postman app. Click the **Published** button near the top of the screen. For a collection that's already been published, you'll be able to view the public link or unpublish the collection.



We've published documentation for this collection - check it out [here](#).

## Custom domains

Optionally, you can pick from a list of verified **custom domains** to host your API documentation. You'll be able to view docs on your custom domain, as well as the Postman private documentation page.

### Choose Environment Template

Learn about using shared environments

No environment

### Custom Domain

api.postman.wtf

None

api.postman.wtf

**Publish**

## Custom styling options

Postman allows you to customize the appearance of your published documentation pages.

Add your [team logo](#) and update the color theme to align with your brand.

You can update the custom styling options either before or after you publish your documentation. Click the **Show Custom Styling Options** link to expand the section and update your color palette.

### Publish Collection Public

Postman Echo

#### Choose Environment Template

No environment

Publishing this collection will make it public to anyone who has the published URL.  
Make sure any shared environments are ready for consumption before publishing.

Custom Domain

None

[Show Custom Styling Options](#)

**Publish**

## Style

Top Bar Color

# FFFFFF



Right Sidebar Color

# 303030

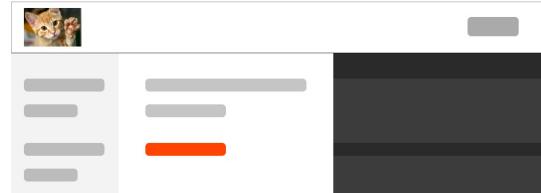


Highlight Color

# EF5B25



## Preview

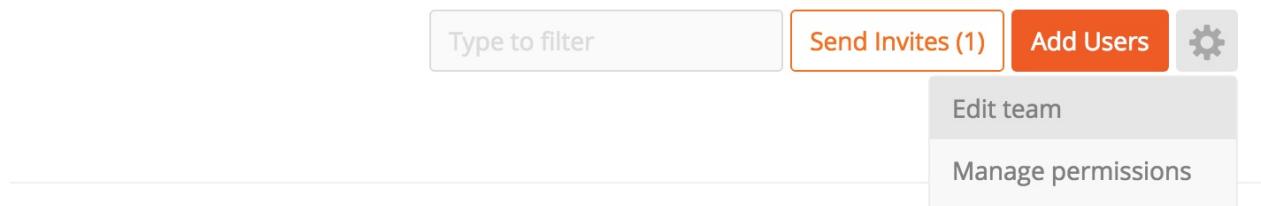


# Adding and verifying custom domains

Postman users with public documentation have the capability to publish documentation on their own custom domain.

## Add a custom domain

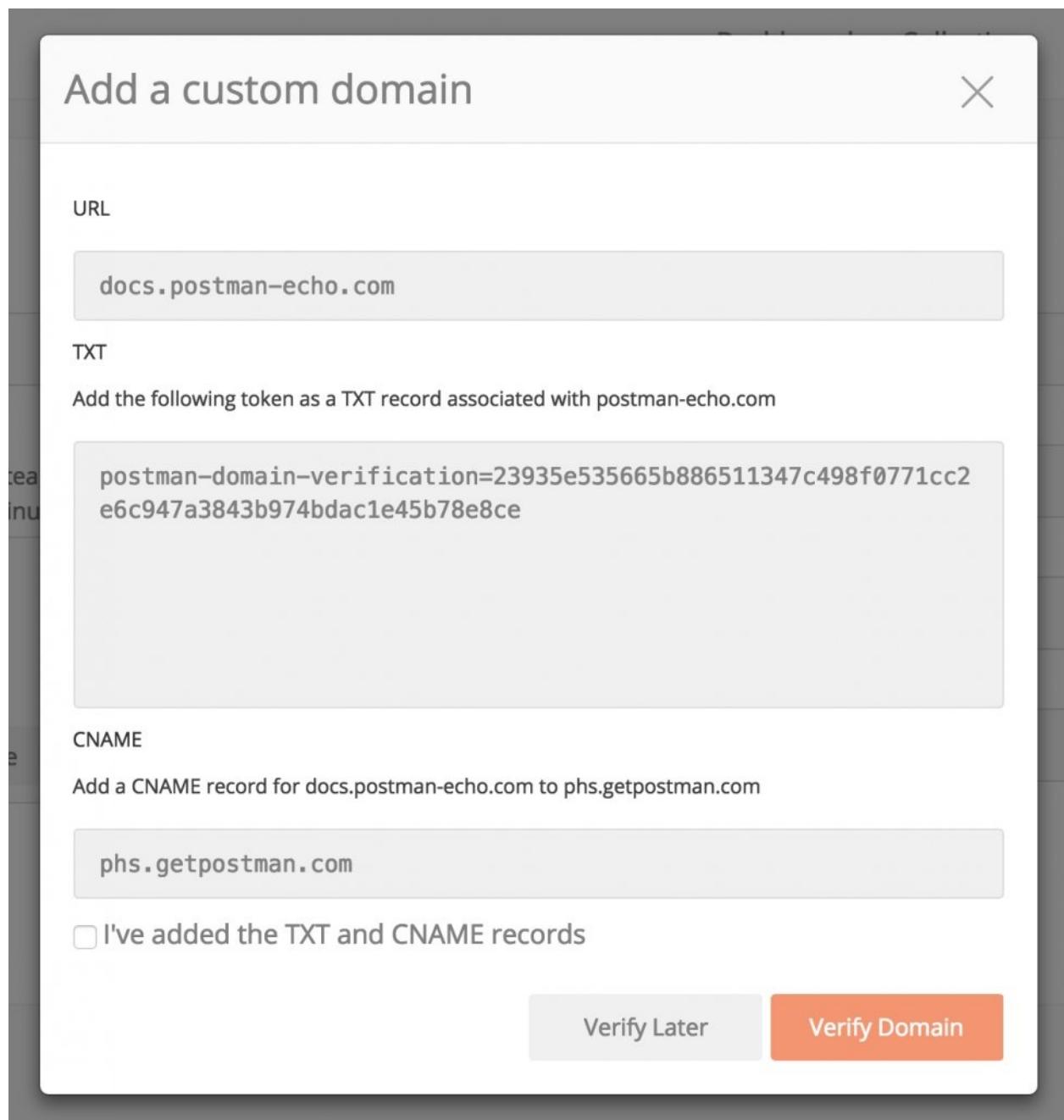
From the Postman web view, go to the [edit view](#) of the Team tab.



There is a section to add custom domains. Add a custom domain by entering the name and hitting the “+” button. Your custom domain can be a full domain or simply a subdomain. For example, you can use either example.com or api.example.com as your custom domain.

## Get verification tokens

Adding a custom domain will open a modal displaying DNS records required to verify domain ownership. The most important pieces of information in the modal below are the TXT and CNAME records. In order to verify that you control the domain you are attempting to add, Postman requires that you add these provided tokens as DNS records to your domain.



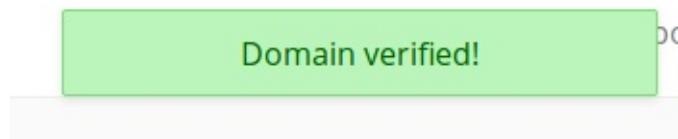
## Add DNS records

In this case, verify ownership with the DNS web service provider. In a separate browser tab, go to the DNS provider's console and add two new records.

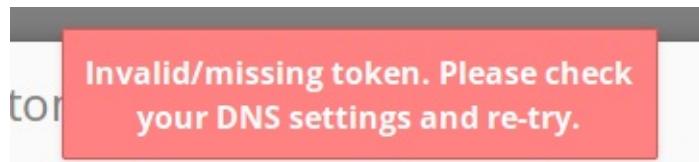
1. A TXT record for postman-echo.com which verifies the ownership of the domain. The value should be the same as the token shown in the modal. The TXT record should be added to the **root domain**. So regardless of whether you're adding example.com or api.example.com, the TXT record should be added to example.com.
2. A CNAME record for docs.postman-echo.com, the value for which should be phs.getpostman.com as shown in the modal. The CNAME record should be added to the URL which will be associated with your public documentation.

## Verify Domain

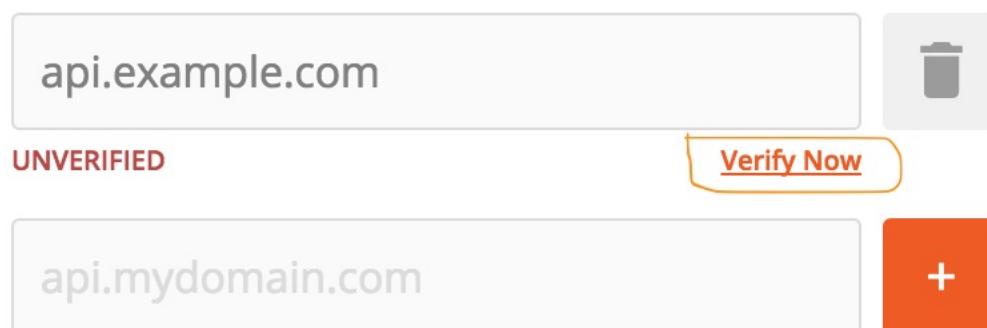
After adding the records in the DNS provider's console, return to the Postman web view. Check the box confirming that the TXT and CNAME records have been added, and then hit **Verify Domain**. A green message will confirm that the custom domain has been verified.



Sometimes, it takes time for your DNS settings to take effect, in which case, you'll see the error message below. This is not a big deal, and you can just try again after some time.



You can also choose to verify your domain control later by clicking the **Verify Later** button in the modal. If you need to access this modal again in order to view the token, or to re-attempt verification, you can click the **Verify Now** link below the custom domain listing.



## Publish a collection on the new custom domain

After the domain is verified, you can use this domain to publicly serve your API documentation. If you have a demo Collection that you would like to publish, head over to the Postman web view and publish your collection. There is now a new option to select a custom domain.

# Publish Collection Public

## Postman Echo

Choose Environment Template

No environment 

Custom Domain

docs.postman-echo.com 

Public URL

<https://docs.postman-echo.com>

[Show Custom Styling Options](#)

**Update**

After publishing, head over to your custom domain, and see the public documentation in all its glory!

# Adding team name and logo

- Pro
  - Enterprise

Postman users with the **Admin** member role can add a team name and logo to their Postman team account directly from the [Edit Team Details page](#).

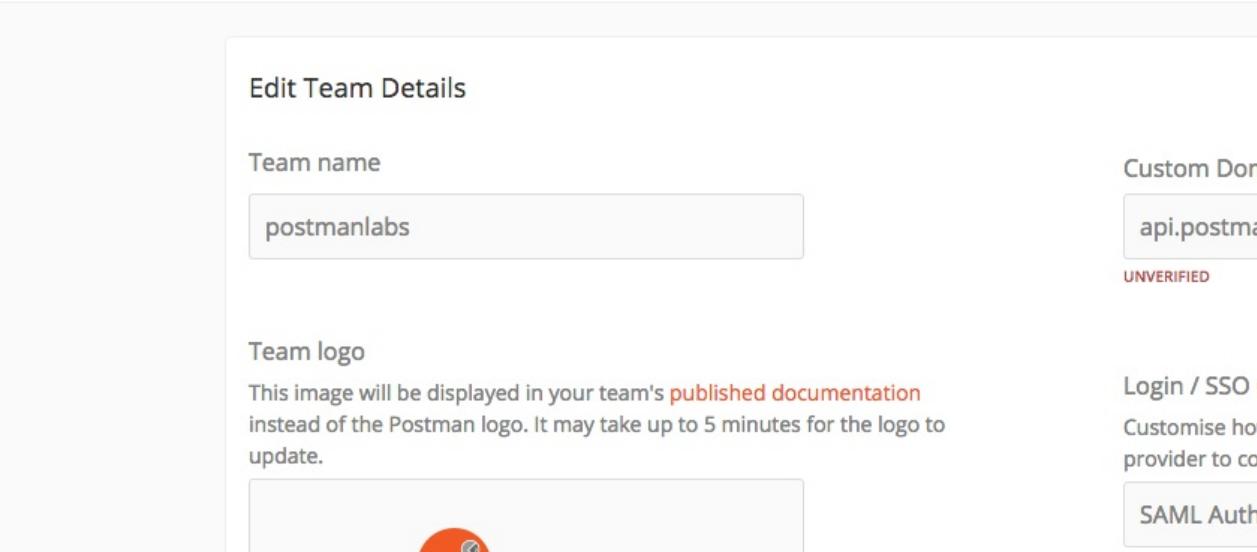
## Update your team name

You can see your current team name on the [team page](#) of the Postman web view.

The screenshot shows the Postman team page. At the top, there's a header with the Postman logo and a search bar labeled "Type to". Below the header, it says "Postman (60 paid slots)" and "Team URL: postmanlabs.postman.co". It also displays "60 active users". A list of four team members is shown, each with a small profile icon, their name, and their email address:

User	Name	Email
	Jelica Festus	jelica@getpostman.com
	Norbu Ngawang	norbu@getpostman.com
	Rashed Fehr	rashed@getpostman.com
	Severianus Edward	severianus@getpostman.com

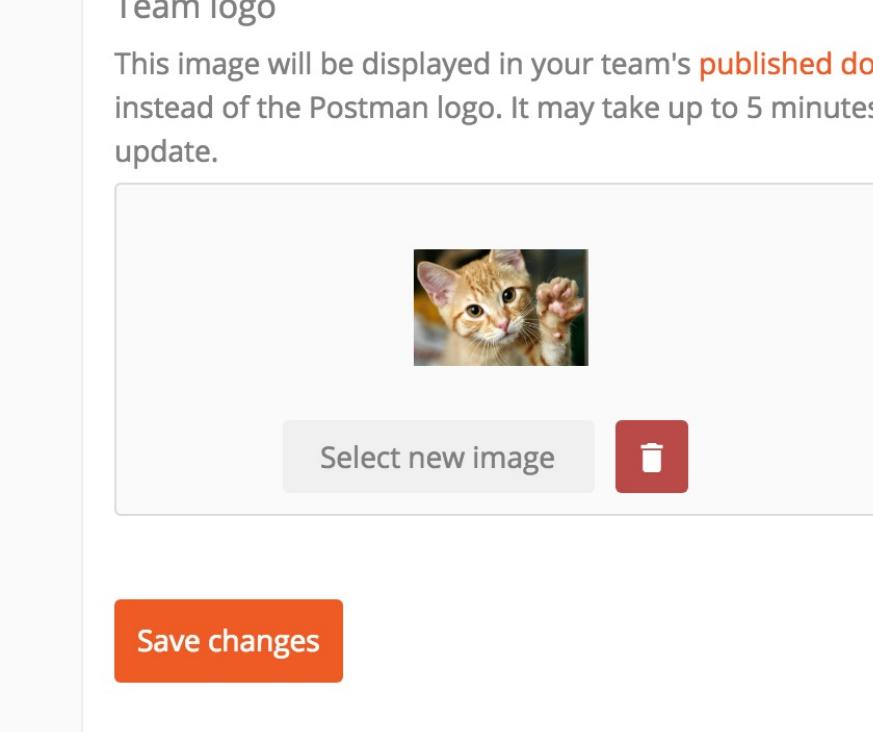
If you didn't enter a team name when you first created your Postman Pro or Enterprise team, you can update that now. From the [team page](#), click the gear icon in the top right, and select "Edit team" to update your team name.



The screenshot shows the 'Edit Team Details' page in Postman. On the left, there's a 'Team name' field containing 'postmanlabs'. To the right, there's a 'Team logo' section with a placeholder image of the Postman logo. A note says: 'This image will be displayed in your team's published documentation instead of the Postman logo. It may take up to 5 minutes for the logo to update.' On the far right, there are several other settings like 'Custom Domains', 'api.postman.com', 'UNVERIFIED', 'Login / SSO', 'Customise how users log in', and 'SAML Auth'.

## Update your team logo

From the [team page](#), click the gear icon in the top right, and select “Edit team” to add or update your team logo.



The screenshot shows the 'Edit team' page. In the 'Team logo' section, there's a placeholder image of a cat's paw. Below it are two buttons: 'Select new image' and a red 'Delete' button. At the bottom, there's a large orange 'Save changes' button.

This uploaded image will be displayed in the header of your team's [published documentation](#) instead of the Postman logo. It may take up to 5 minutes for the logo to update.



# Intro to Monitors

## What is a monitor

A monitor lets you run a [collection](#) periodically to check for its performance and response. You can set up a monitor to run as frequently as 5 minutes to check if all the requests in your collection are up and healthy.

When you set up a monitor, Postman servers will hit the endpoints in your collection according to the specified frequency. You can also select a corresponding [environment](#) to use and store variables. If you have written [tests](#) for your requests , the monitor would run these tests to validate the response and notify you when a test fails. You can configure how to receive the alerts from a wide number of [integrations](#) available.

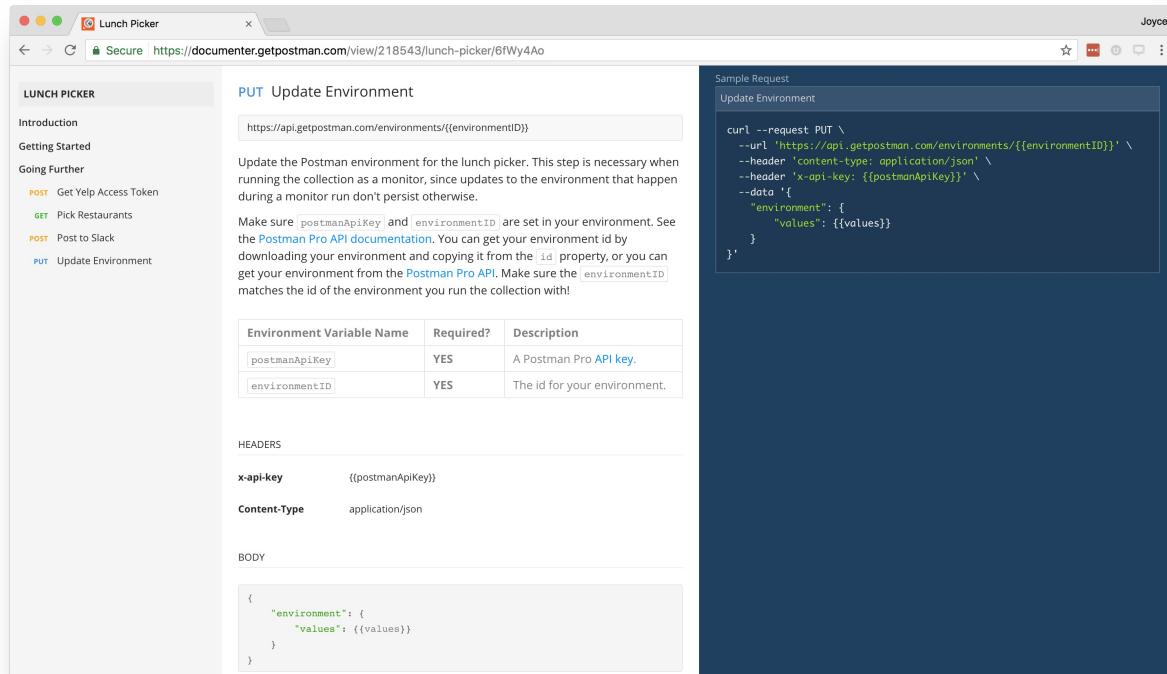
Each Postman user gets 1,000 monitoring calls for free per month. Each Postman Pro and Enterprise team gets 10,000 free monthly requests, and it takes only 2 minutes to set up a monitor. Learn more about [monitor pricing](#) and [getting started with monitors](#).

## Running collections in a Monitor (vs. the Postman app collection runner)

There are a few minor differences between running collections in a Postman monitor as compared to using the Postman app collection runner. If your collection relies on any of these features, then it may not work the same way in Postman monitors that it does in the Postman app.

### Variables

- Can't import existing global variables, but you can create new ones during a monitor run.
- Global and environment variables are not persisted. If you require persisting environment variables, we recommend adding a call to update the environment variable using the [Postman Pro API](#). The following is an [example of how to update the environment variable](#) in this manner.



## Console Output

- Unlike in the Postman app, request & response bodies are not logged to the console by default. This is for security and privacy reasons.
- Same goes for potentially-sensitive headers, such as cookies and auth keys

## Time Limits

- Monitors can currently be scheduled to run as often as every 5 minutes, or as little as once a week. Each run is limited to 2 minutes, including all HTTP requests, responses, and pre-request and test scripts.

## File Uploads

- Can't attach files to requests, like you can in the request builder
- But you CAN upload data as raw request body

## Multiple Iterations

- Monitors only run 1 iteration by default
- But you can use `setNextRequest()` to do multiple iterations

## Multi-region Monitoring

- Monitors allow you to run collections in specified geographic regions
- Can only specify multi-region monitoring from the [monitors page](#)

## Data Files

- Can't attach data files like you can in the runner
- But you can access data files from APIs, such as Google Docs, Google Sheets, Dropbox, etc.

## Accessible APIs

- As with the Postman app, the monitors require all URLs to be publicly-available on the Internet. In the future, you will be able to monitor private APIs as well.
- Monitors can't directly access your localhost and might encounter a firewall because monitors run in the Postman cloud.

## Monitoring resources in multiple regions

Monitoring resources across multiple regions provides useful information about the status and response time for your endpoints. If you've implemented a solution by setting up multiple servers running on multiple continents, then you want to make sure your endpoints are healthy and that none of your users are experiencing unusual delays.

Postman supports monitoring in 6 geographic regions around the world. If you're interested in a region that's not listed in the Postman interface, contact us at [help@getpostman.com](mailto:help@getpostman.com) or through the chat box on the [monitors page](#).

## Pricing for Monitors

Monitors are priced per request made, with some free requests included every month. Learn more about [pricing for Monitors](#).

## Free monitoring calls with your Postman account

Your Postman account gives you a limited number of free monitoring calls per month. You can check your usage limits through the [Postman Pro API](#) or the [account usage page](#).

## Setting up a monitor

If you choose to monitor a shared collection, your team can see the monitor. However, if you create a monitor on an unshared collection, the monitor will be private and only visible to you.

### From the Postman app

Click on the ellipses (...) next to the collection you wish to monitor. Select “Monitor Collection” to open the **MONITOR COLLECTION** modal.

Postman Echo ★

37 requests

Auth: Digest

Auth: Others

Cookies

Headers

Request Methods

Utilities

Utilities / Date

Utilities / Postman

Postman Pro  
21 requests

Share Collection

Rename ⌘E

Edit

Add Folder

Duplicate ⌘D

Export

Monitor Collection

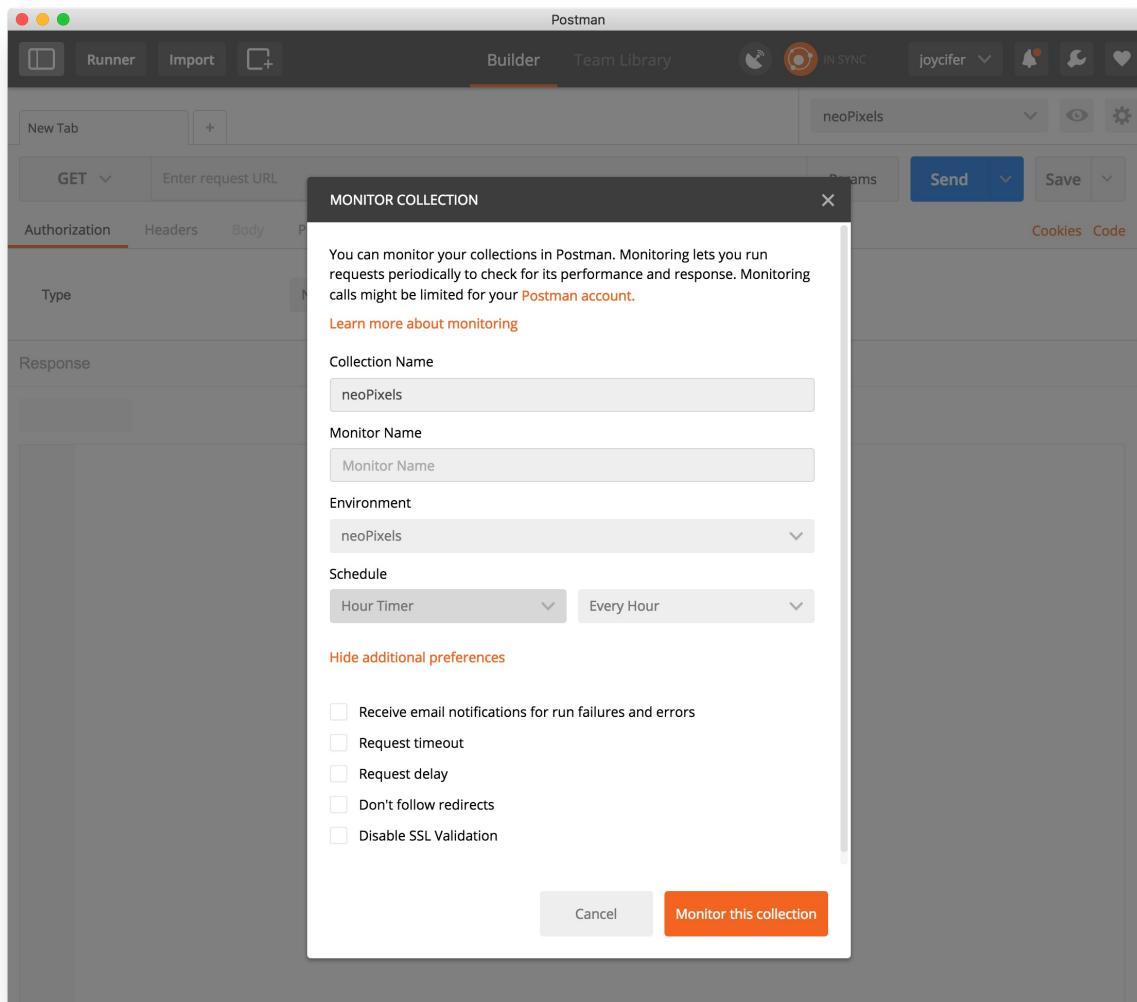
Mock Collection

Publish Docs

Delete

[Reddit to Slack](#)

Enter a name for this monitor and choose a corresponding environment. Add an appropriate [schedule for the monitor](#), and configure [additional preferences](#).



## From the Postman web

Sign in to the Postman web view, and head over to the [monitors page](#). Click the **Add Monitor** button.

The screenshot shows a web-based monitoring tool. At the top, there are two tabs: "Team" (selected) and "Private". To the right are search fields for "Search for monitors" and a red "Add Monitor" button. Below this, a message says "No collections being monitored." with a link to "Track Usage". On the right, there are filters for "TIMEFRAME Last 24 Hours" and "GROUP BY None". The main area has columns for "NAME", "STATUS", "AVG. SUCCESS RATE", and "AVG. RESPONSE TIME". A large orange icon of a computer monitor with a heart rate graph on its screen is centered. Below it, a message says "You don't have any monitors set up." and a red "Add Your First Monitor" button.

Enter a name for this monitor. Choose the collection that you want to monitor and a corresponding environment. Add an appropriate [schedule for the monitor](#), and configure [additional preferences](#).

## Monitoring schedule and region

Select a frequency to run your monitor. Monitors can run as frequently as every 5 minutes. For example, you can run a monitor every 5 minutes or every Monday at 9:00 AM.

## Create Monitor



### Collection

Type to filter



### Schedule

Hour Timer



Every Hour



### Monitor Name

### Environment

No Environment



### Regions



United States (East)

To [monitor resources in multiple regions](#), select the regions to run your monitor from the [monitors page](#). When you specify a monitor to run in multiple regions, the monitor will run multiple times. This means that if there is a side effect from running the monitor, it will also happen multiple times.

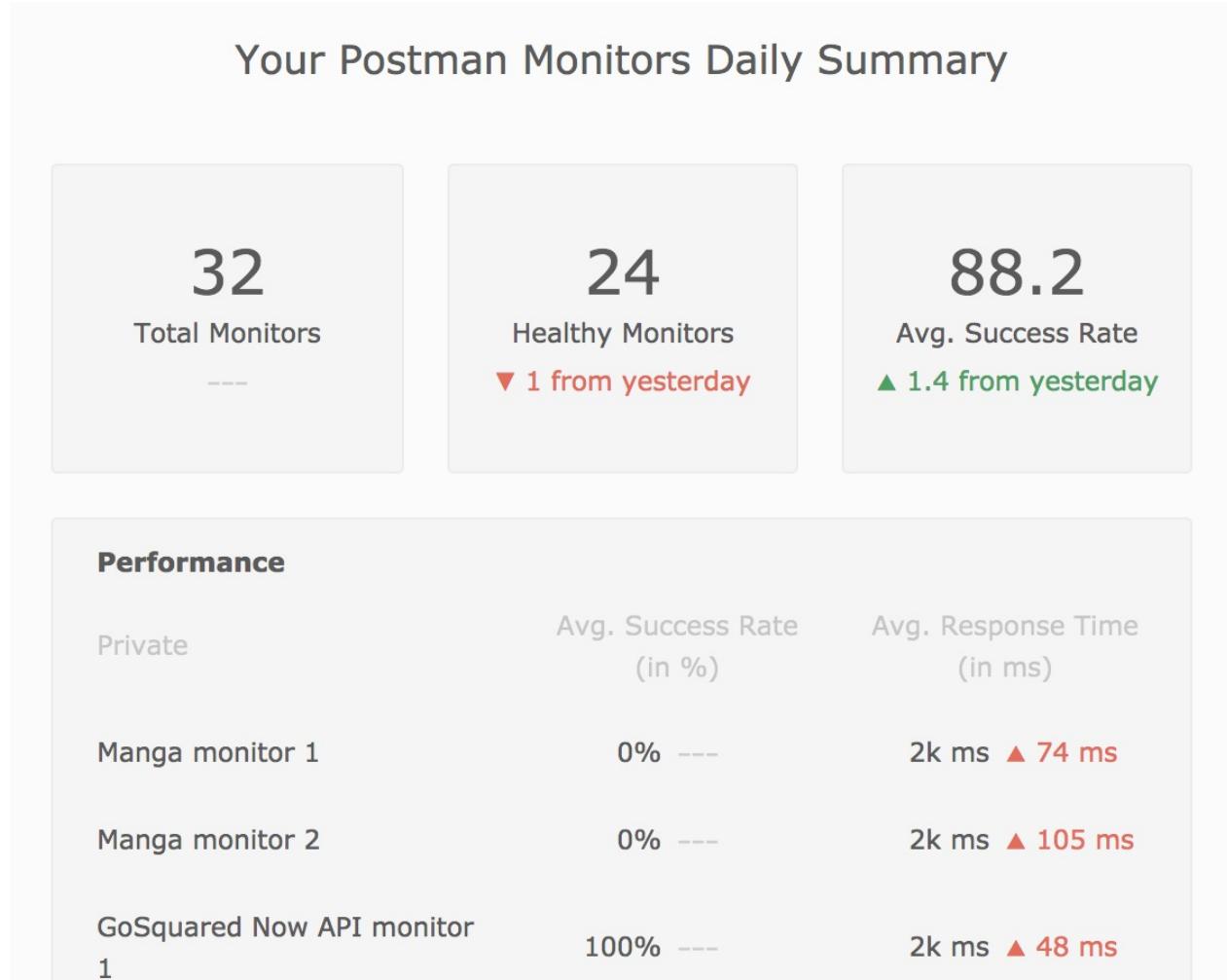
## Additional preferences

**Additional preferences**  
\*\*Description\*\*  
Email notifications  
Get notifications about failures on up to 5 email addresses  
Request timeout  
Specify a time interval after which your request is considered to time-out  
Delay between requests  
Add a time lag between subsequent

requestsDon't follow redirectsDisable following all redirectsDisable SSL validationThis disables validations performed on SSL certificates. Check this if you use self-signed certificates. Use with caution!

# Viewing monitor results

Once you set up monitors, you can receive daily and weekly emails with a summary for all your monitors. These email notifications can be turned off in the settings. In addition, you will receive important notifications (both [in-app](#) and email) in case a monitor fails.



You can also view more detailed results from your dashboard.

## Monitors page

Sign in to the Postman web view, and head over to the [monitors page](#) which lists out all your monitors (both team and private). Select a timeframe for which you want to view the results of the monitor runs.

## Viewing monitor results

Team		Private	Search for monitors			Add Monitor
	36 Collections with 44 Monitors are up and running. Admins can view all monitors set up in a team.			TIMEFRAME	Last 24 Hours	GROUP BY
NAME	STATUS	Avg. Success Rate	Avg. Response Time			
Production Abhinav	24 FAILURES	0%	- 0%	1,534 ms	+ 1.7%	
BuildKite PR Gatekeeper Kamalakannan N	7 FAILURES	87%	+ 31%	200 ms	+ 12%	
this is testing Ankit Jaggi	2 FAILURES	50%		170 ms		
Postman_IDOR_Test#Beta Suhas	2 FAILURES	33%	+ 8%	8,673 ms	+ 5.2%	
ElasticBeanstalk Audit #aws #ops #POC monitor Kunal Nagpal	1 FAILURE	0%	- 0%	67,190 ms	+ 12%	
RDS Audit #aws #ops monitor Kunal Nagpal	1 FAILURE	0%	- 0%	11,617 ms	+ 60%	

You can view various stats for each of the monitors.

**Status** Number of failed runs in the selected timeframe. A ‘Healthy’ status indicates that there were no failures in any of the runs. **Average success rate** The percent of successful runs out of the total runs in the timeframe. A run is said to be successful only when all the tests passed during that run. You can also see a change in this value compared to the previous time period. **Average response time** The average response time of all the requests over all the runs in milliseconds. You can also see the percentage change from the previous time period. Click on any of the monitors to view more details about its performance and troubleshooting in case of any failures.

## Monitor details page

The main timeline shows all past runs of the monitor. Each bar signifies one run of the monitor - with red indicating failing tests. The graph in blue shows the total response time (of all the requests) over time. This is a great way of measuring performance improvements when you’ve made changes to your infrastructure.



You can click on any of the runs from this graph to see its results in more details.

The screenshot shows the Postman Monitor interface. At the top, there are tabs for 'Results' (selected) and 'Console Log'. Below that, it says 'Run on 19 Jul, 2017 11:30 PM'. On the right, it shows 'REGION United States (East)'. The main area displays a table of test results:

55 passes		63 failures	3 seconds	Need help debugging?		
<span style="color: red;">✖</span>	<span style="color: green;">✖</span>	GET Describe RDS Instances https://rds.us-east-1.amazonaws.com/?Operation=DescribeDBInstances&V... 200 OK 928 ms 84403 B				
<span style="color: green;">✖</span>	<span style="color: green;">✖</span>	PASS Status code is 200				
<span style="color: red;">✖</span>	<span style="color: green;">✖</span>	GET Get Metric Statistics: FreeStorageSpace https://monitoring.us-east-1.amazonaws.com/?Dimensions... 200 OK 107 ms 514 B				
		PASS Status code is 200				
<span style="color: red;">✖</span>	<span style="color: green;">✖</span>	GET Check RDS Instance tags https://rds.us-east-1.amazonaws.com/?Operation=ListTagsForResource&V... 200 OK 92 ms 683 B				
		PASS Status code is 200				
	<span style="color: red;">✖</span>	FAIL production-postman-monitor-db-a uses encrypted storage (false)				

The results section shows request-level details: test results, response code, response time, and the response size. Additionally, you can filter by region if you set up [monitors in multiple regions](#).

You can also view the log by clicking on the **Console Log** tab.

```
1 16:00:46 Production started.
2 16:00:46 Documenter - JSON Blob
3 16:00:46 GET https://documenter.getpostman.com/view/55577/jsonblob-core-api-published/JCqd
4 16:00:46 Homepage
5 16:00:46 GET https://www.getpostman.com
6 16:00:46 Pro API docs
7 16:00:46 GET https://api.getpostman.com
8 16:00:46 unable to verify the first certificate
9 16:00:47 unable to verify the first certificate
10 16:00:47 Postman Echo
11 16:00:47 GET https://echo.getpostman.com
12 16:00:47 unable to verify the first certificate
13 16:00:47 unable to verify the first certificate
14 16:00:47 Blog
15 16:00:47 GET https://blog.getpostman.com/
```

The console log will print a detailed log of run events, along with any `console.log` statements that ran as part of your pre-request and test scripts and can be used to [diagnose failures](#).

# Monitoring APIs and websites

Some teams use Postman monitors to ensure their APIs and websites remain operational. Monitors can be run as frequently as five minutes.

## Monitoring APIs

### Monitoring a specific endpoint

To monitor a specific endpoint, create a collection with different variants of the same endpoint in different requests. The idea here is to test responses for each variant, so as to cover the endpoint completely. Review a complete [reference of testing various aspects of a request](#).

### Monitoring an entire API

This is similar in approach to monitoring a specific endpoint, with the subtle difference of storing the common API host in an environment variable, such that the requests across different API endpoints differ in their path, among other request parameters. Such a sequence also makes it possible to chain data across requests, which allows testing an entire API as a whole.

### Running an API test suite

In an API where various endpoints are interlinked, precise knowledge about their functioning is crucial. In cases where data is passed from one request to another, the entire response, or a part of it, can be saved as an environment variable. Additional care should be taken while setting non-atomic values (objects, arrays, etc), as the original value will be lost. Instead, such complex objects and arrays can be handled via:

```
// set the value
postman.setEnvironmentVariable('complexObj', JSON.stringify(myComplexObjOrArray, null, 2));

// Fetch the value
var foo;
try {
    foo = JSON.parse(postman.getEnvironmentVariable('complexObj'));
}
catch (e) {
    console.error(e);
    foo = { __parseError: true };
}
if (foo.__parseError) {
    // handle parse errors here
}
```

With the stringified nested value in place, it can be passed to subsequent requests, for instance, as a request body.

## Monitoring Websites

### Monitoring HTTP response codes

Response code tests can be done by checking the value of `responseCode.code` within test scripts.

```
tests['Request resulted in 200 OK'] = responseCode.code === 200;
```

### Monitoring latency

As an alternative to request timeouts, website response latency can be monitored by comparing values of the `responseTime` variable within test scripts.

```
tests['Response latency is acceptable'] = responseTime < 1000;
// responseTime is in milliseconds
```

# Set up integrations to receive alerts

- Pro
  - Enterprise

## Webhooks (Coming soon)

Postman Monitors enable you to set up recurring runs of your Postman Collections at scheduled intervals. But sometimes, you may have a use case where you need to run a monitor at a particular time. That's where the monitoring webhooks come in. Monitoring webhooks are a way to trigger a collection at a specific time with your own custom payload which can then be accessed in the collection. In this way, your collections can run independently of any environment and can solely rely on the incoming data in the request.

So how do they work? Webhooks will POST data to a URL when certain events are triggered. That data will then be accessible inside your collection in the [globals object](#). You can then parse that data and use it in any way possible. Essentially, webhooks are the same as monitors but without a schedule. So, you can debug your webhooks in the same way as you [debug a monitor or a collection](#).

Currently, webhooks on a particular collection can only be created using the Postman Pro API. In order to create a webhook, you can refer to the [Postman Pro API](#).

## Accessing the request body in scripts

The request body of the webhook is available inside the `globals.previousRequest` object. In order to use it, first parse the `globals.previousRequest` object. The data sent to the webhook is available in the `data` parameter inside the parsed object.

The following snippet shows the same:

```
var previousRequest = JSON.parse(globals.previousRequest),
    webhookrequestData = previousRequest.data;

// webhookrequestData contains the data sent to your webhook.
console.log(JSON.stringify(webhookrequestData));
```

Note: only JSON data is currently supported as the request body in the webhook.

## Sending output to another API

The data that is sent to the webhook can be used to trigger another API and define a logic based on the incoming data. For example, you could set up a monitoring webhook on your Github repository, so that based on the updates happening in your repository, you can trigger custom build pipelines and perform CI tests.

# Pricing for monitors

- Pro
  - Enterprise

Usage of Postman Monitors is billed on a per-request basis. A request is any HTTP request needed to run your collection. If your collection has 5 requests, but you've used `postman.setNextRequest()` to skip some requests, or run requests multiple times, you'll be billed according to the number of requests actually made, not the number of requests in your collection. Any requests needed for the auth helpers (Digest Auth, OAuth, etc.) will also be included in your usage count.

Each Postman user gets 1,000 monitoring calls for free per month. Each Postman Pro or Enterprise team gets 10,000 free requests per month. The first month starts the day you send your first monitoring request, or when you set up a monthly block for your team.

Teams on the free Pro trial cannot go beyond this limit. If you are on the free trial, you will have to wait for the next monitoring billing cycle to get another 10,000 requests.

## For paid teams

- If you're in a paid team, you can go over the limit, but you will be billed at \$0.75 for every 1,000 extra requests you make (over the limit of 10,000).
- For a more predictable billing pattern, you can configure "blocks" of requests for your team.
- You'll be charged (for configured blocks + any overage requests) at the end of the monitoring billing cycle. We'll attempt to charge your card if one is saved under your account. If there's none, or we're unable to charge your card, we'll send you an invoice at your registered billing email, payable within 30 days.

## Request blocks for paid Pro teams

On the [billing page](#), paid teams can configure "blocks" of requests to save on monitoring charges. Each request block gives you 500,000 requests at \$200, and is auto-renewed each month. If you set up monitoring blocks before sending your first monitoring request, your monitoring billing cycle will start from the day you set up the blocks. You may increase or decrease your block count at any time during the billing cycle, but the cost of extra blocks will not be prorated.

At the end of each billing cycle, you'll be charged \$200 for each block configured, plus \$0.75 per 1000 requests made over your block limit.

**Example:**

If a paid team has configured 2 blocks, and makes 1,200,000 requests in a billing cycle, they will be charged: (2 blocks \$200/block) + (\$0.75 \* 190) = \$542.50 for that cycle.

Prepaid requests: 10,000 (free) + 2 \* 500,000 (2 blocks) = 1,010,000

Overage requests: 1,200,000 - 1,010,000 = 190,000

Note that unused requests don't "roll-over" to the next month.

## Tracking Usage

Team admins can use the "Track Usage" link on the monitor dashboard to check how many requests the team has made that month.

# Troubleshooting monitors

Monitors are kept in sync with your collections in the Postman app at all times. This means that you can debug in the app locally, while your monitors are updated on our servers, seamlessly.

The best way to debug monitors is via the Postman console available on the monitors web view. Click on the failed monitor, and review the relevant logs under the **Console Log** tab. Check out some tips and tricks for [debugging in the console](#).

The screenshot shows the Postman monitors interface. At the top, there are tabs for 'Results' and 'Console Log', with 'Console Log' being the active tab. Below the tabs, it says 'Run on 28 Mar, 2017 4:00 PM'. The main area displays a log of requests:

```

1 16:00:50 Weather to Slack Demo Test copy monitor 1 started.
2 16:00:50 Get lat/long
3 16:00:50 GET https://maps.googleapis.com/maps/api/geocode/json?address="415 Jackson Street"
4 16:00:50 { type: 'Error',
5 16:00:50   name: 'TypeError',
6 16:00:50   message: 'Cannot read property \'geometry\' of undefined' }
7 16:00:50 Get Weather
8 16:00:50 GET https://api.darksky.net/forecast/{{weatherapikey}}/{{latlong}}
9 16:00:50 Post to Slack
10 16:00:50 POST https://hooks.slack.com/services/T0WB3TACU/B2117R12A/SWaMqcYgo0D4QoGwvJFJ
11 16:00:51 Weather to Slack Demo Test copy monitor 1 finished.

```

A 'Need help debugging?' modal is overlaid on the right side of the screen. It contains the following content:

**Tip:** Using `console.log` in your scripts makes the process of debugging much easier.

- Test your collection in the [Postman Runner](#) or [Newman](#). If the results differ, try debugging your tests in the [Request Builder](#).
- Contact us at [help@getpostman.com](mailto:help@getpostman.com).

When a monitor fails, a “*Need help debugging?*” popup will display on the Postman monitors web view. Whenever possible, these suggestions will include debugging tips that are specifically related to the error that occurred. For example, if the monitor failed because we were unable to contact the server, then the debugging tip will suggest ensuring that the server’s IP address is correct, and that it’s publicly available. Or, if the monitor failed because of a missing variable, then the debugging tip will suggest verifying the monitor is using the correct environment, and that the variable exists in the environment.

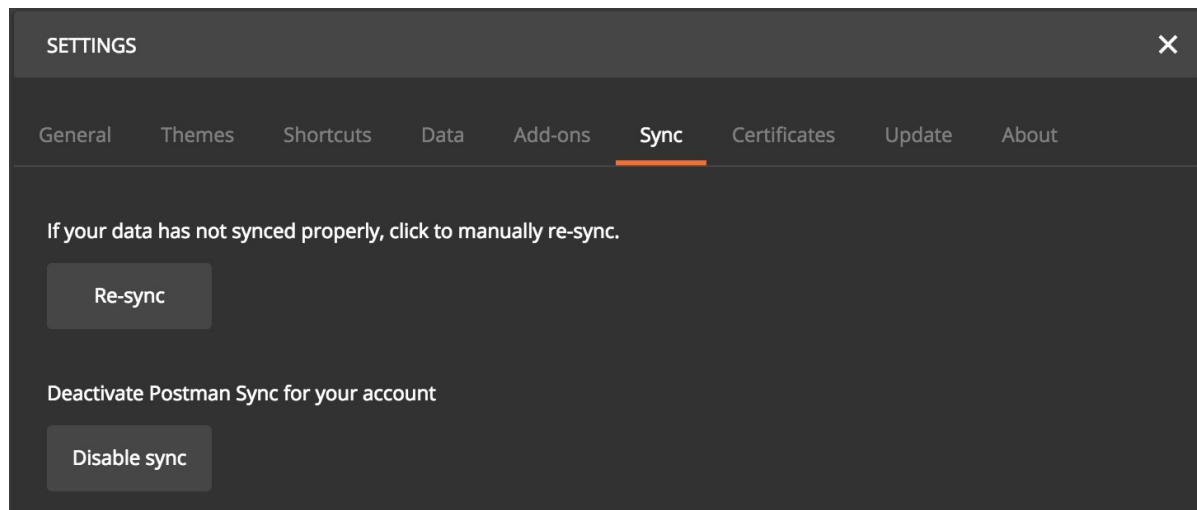
In addition to the above, here are some debugging tricks:

## Attempt local runs

- Try running the failing monitor’s collection with its environment within the Postman app or Newman, and see if it works correctly.
- If a local run passes, [ensure that sync is enabled](#) within the toolbar header of the Postman app, so that any local changes are persisted.



- You can also force a sync from the **Sync** tab within the **SETTINGS** modal.



### Variable issues

- Ensure that the same environment is used across local runs and monitor runs. You can confirm this by adding `console.log(environment);` to your request scripts and comparing the results across monitoring and local runs.
- If your collection run depends on a global variable, change it to an environment variable. Global variables are not supported in monitors at this time.

### Log relevant information

- Often, issues come from unexpected response bodies or header values. You can log these with the following:

```
console.log(JSON.stringify(responseBody, null, 2));
console.log(JSON.stringify(responseHeaders, null, 2));
```

### Uncaught errors

- Wrapping suspicious code in a try - catch block will also let the test and pre-request scripts in your collection run to completion, allowing you to see the entire picture.

# FAQs for monitors

## General questions

### What can I test with Monitors?

You can use Postman Monitors for simple uptime monitoring (to make sure your servers are online) or performance monitoring (to make sure your servers are responding promptly), or you can write detailed [test suites](#) to check for proper behavior, business logic, error handling, etc.

### What restrictions apply?

Review [differences between monitors and the Postman app](#).

### How many monitors can I create?

There is no limit to the number of monitors you can create. You can have any number of collections, each with any number of monitors. And each monitor can run on a different schedule.

### How long can a monitor run?

Monitors are currently limited to 5 minutes for each run. This includes all HTTP requests, responses, and test scripts.

### How many HTTP requests can a monitor send?

There is no limit to the number of requests, although the total run-time cannot exceed 5 minutes.

### How much data can a monitor send/receive?

There is no limit to the amount of data that can be sent or received per request. However, large requests or responses will take longer to send and receive, so be sure that you can do everything within the 5-minute time limit.

### How do I troubleshoot problems?

You can view the full console output for every monitor run, including any errors that occurred. You can also use methods such as `console.log()`, `console.warn()`, etc. to output your own debugging information. Learn more about [troubleshooting monitors](#).

## Security

### Who can see my Monitors?

Monitors have the same permissions as Postman Collections. By default, your collections are private, so only you can see the collection and its monitors. If you share a collection, then other members of your Postman Pro or Enterprise team will be able to see the collection and its monitors. If you grant View & Edit permissions, then your team members will also be able to add monitors to your collection.

Each collection can have different permissions, so you can choose to have some private, some shared but view-only, and some shared and editable.

### Can I delete a Monitor?

Yes. You can delete a monitor at any time. Once deleted, all run history for the monitor is deleted as well. If you want *keep* the history, then you should pause the monitor instead of deleting it.

### Where do Monitors run?

Monitors run on our cloud infrastructure, which is hosted by Amazon Web Services (AWS). More information about our cloud infrastructure is available at [our Security page](#).

### Can Monitors access private networks?

No. Monitors can only connect to URLs that are publicly-available on the Internet. You cannot monitor APIs running on private networks, VPNs, or corporate intranets.

### Will Monitors impact my API performance?

You have full control over the behavior of your monitors. You determine which of your API endpoints are called, how many, and how often. In addition, we restrict each monitor's total run-time to 5 minutes, to limit the number of requests it can perform.

# Setting up a mock server

## Simulate a back end with Postman's mock servers

Throughout the development process, delays on the front end or back end can hold up dependent teams from completing their work efficiently.

Using Postman's mock servers, front-end developers can simulate each endpoint in a Postman Collection (and corresponding environment) to view the potential responses, without actually spinning up a back end.

Front-end, back-end and API teams can now work in parallel, freeing up developers who were previously delayed as a result of these dependencies.

## Setting up the mock

Developers can mock a request and response in Postman before sending the actual request or setting up a single endpoint to return the response. Establishing an [example](#) during the earliest phase of API development requires clear communication between team members, aligns their expectations, and means developers and testers can get started more quickly.

There are two ways to create a mock:

1. [using the Postman app](#)
2. [using the Postman Pro API](#)

Once the mock has been created, Postman Pro and Enterprise users can share the mock with their team for review and collaboration. This is accomplished by [sharing the underlying collection](#) with the team or specific team members, providing permissions to edit or view.

## HTTP access control (CORS)

Not only can you make requests to mock endpoints using the Postman app, you can also rely on a mock using a browser. A web browser makes a cross-origin HTTP request when requesting a resource from a domain, protocol, or port that's different from its own. For security reasons, [cross-origin resource sharing \(CORS\)](#) is a standard that defines a way in which a browser and server can interact securely. In this case, we are referring to how a web browser interacts with the mock endpoints hosted on the Postman server.

CORS is enabled for Postman mock servers which means you can stub your web apps with mocked data using the mock endpoints. In other words, development or production web apps can make requests to the Postman mock endpoint you just created and receive an

example response.

## Free mock server calls with your Postman account

Your Postman account gives you a limited number of free mock server calls per month. You can check your usage limits through the [Postman Pro API](#) or the [account usage page](#).

# Mocking with examples

Let's take a deep dive into how [mock servers](#) and [examples](#) work together, and how you can integrate them into your workflow for a more enhanced API experience with Postman.

1. Sending a request (R1)
2. Saving the request (R1) to a collection (C1)
3. Saving the request R1's response as an example (P1)
4. Creating a mock (M1) for the collection (C1)
5. Sending a request using the mock server (M1)

## Setting up some basics

Before we get into the details of mocking, let's start with setting up some basics required for mocks to work:

### Step 1: Sending a request (R1)

From the Postman app, send a GET request to the URL <https://postman-echo.com/get?test=123>. This request hits the [Postman Echo](#) service which you can use to test out your REST clients and make sample API calls.

The resulting response can be seen on the right, and a record of this request will now be visible in your [history](#) on the left.

The screenshot shows the Postman interface. On the left, the 'History' tab in the sidebar is highlighted with a pink box, showing a recent request: 'GET https://postman-echo.com/get?test=123'. An arrow points from this entry to the main workspace. The main workspace displays a successful response for the same URL, with a status of 200 OK, time 1984 ms, and size 866 B. The response body is shown in JSON format:

```

1 - {
2 -   "args": {
3 -     "test": "123"
4 -   },
5 -   "headers": {
6 -     "host": "postman-echo.com",
7 -     "accept": "*/*",
8 -     "accept-encoding": "gzip, deflate",
9 -     "cache-control": "no-cache",
10 -    "cookie": "sails.sid=s%3A0nf40MoZG0rqKNSU28rgP5d01SDXuK7s.1C70mN7dq7471fRL2n8mPho08mu9%2F26GNm5FVKAYgsM",
11 -    "postman-token": "0fedaf6d-2900-4a29-af03-256302a7e2ed",
12 -    "user-agent": "PostmanRuntime/6.1.6",
13 -    "x-forwarded-port": "443",
14 -    "x-forwarded-proto": "https"
15 -  },
16 -  "url": "https://postman-echo.com/get?test=123"
17 -

```

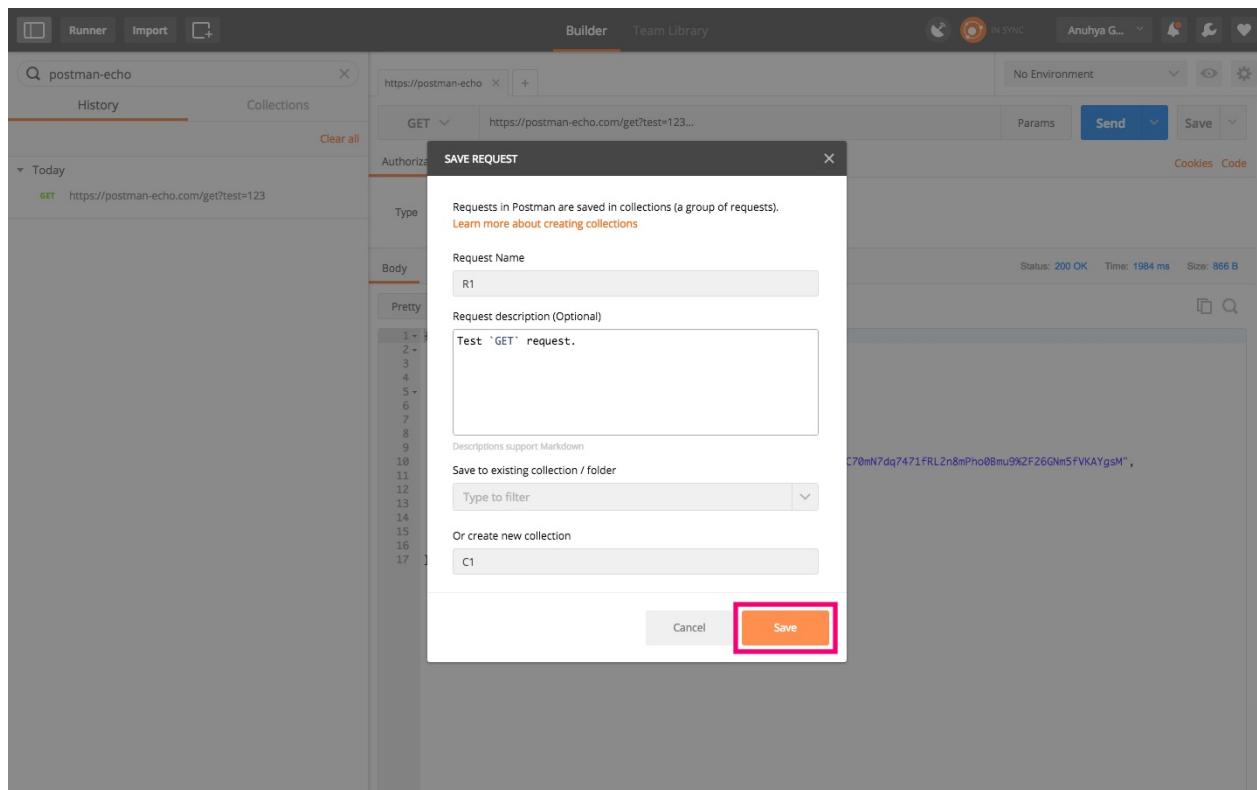
### Step 2: Saving the request (R1) to a collection (C1)

Hit the **Save** button to open the **SAVE REQUEST** modal. **Collections** are simply groups of requests that can be connected together to create APIs and workflows.

The screenshot shows the Postman interface again. The 'Save' button in the top right corner is highlighted with a pink box and an arrow points to it from the text 'save the request to a collection' at the bottom right. The main workspace shows the same successful response as before. The response body is identical to the one in the previous screenshot.

You can save a request to an existing collection, or save it to a new collection. Let's create our new collection called C1.

## Mocking with examples



Collection C1 will now be accessible in the **Collections** tab in the application. We can do all sorts of things within the collection details view: [viewing API documentation](#), [mocking a collection](#), [monitoring a collection](#), or [running the collection](#).

Newly created Collection, C1 with request, R1

### Step 3: Saving the request R1's response as an example (P1)

Now, let's save an example response from the request we just sent by hitting the **Save Response** button.

The screenshot shows the Postman Builder interface. On the left, there's a sidebar with 'C1' selected. In the main area, a request labeled 'R1' is shown with a 'GET' method and URL 'https://postman-echo.com/get?test=123...'. The 'Headers' tab is selected. On the right, the response body is displayed in JSON format, showing various headers and parameters. A red box highlights the 'Save Response' button at the top right of the response preview area.

```

1 [
2   "args": {
3     "test": "123"
4   },
5   "headers": {
6     "host": "postman-echo.com",
7     "accept": "*/*",
8     "accept-encoding": "gzip, deflate",
9     "cache-control": "no-cache",
10    "cookie": "sails.sid=s%3Afddp7ZqBJWZd_AQgj54VIL162p30JLQ.2FTnndgE9KZK2BzTxMlLwvNljSYB&qPFGN=Dl0Cg4pc8Y",
11    "postman-token": "6ba03584-c82c-4bde-b0f5-95e83d2edf7b",
12    "user-agent": "PostmanRuntime/6.1.6",
13    "x-forwarded-port": "443",
14    "x-forwarded-proto": "https"
15  },
16  "url": "https://postman-echo.com/get?test=123"
17 }

```

This takes us to the **Examples** screen which can be used to save the request response as an example. Let's call this example P1.

The screenshot shows the Postman Examples screen. It displays a saved example named 'P1'. The example details are as follows:

- Saved Example Name:** P1
- Request Method:** GET
- Saved Example URL:** https://postman-echo.com/get?test=123...
- Status Code of Example:** 200 OK

The example response body is identical to the one shown in the previous screenshot.

Enter a name for this example. The request method, URL, and status code are crucial in determining which responses will be returned by the mock we will create. Verify these elements are all as desired, and hit the **Save Example** button. Hit the back arrow in the top left to return to the request builder, and we can now see the example we created in the top right, added to the request.

The screenshot shows the Postman application's Builder interface. On the left, there's a sidebar with 'History' and 'Collections'. Under 'Collections', 'C1' is selected, showing '1 request'. The main area displays a request named 'R1' with a GET method and URL 'https://postman-echo.com/get?test=123...'. The 'Headers' tab is active. In the top right corner, there's a 'Params' section with a table and a 'Cookies' tab. To the right of the table, there's a 'Presets' dropdown. A red box highlights the 'Examples (1)' dropdown, and an arrow points from it to another red box containing the text 'Example we just saved'.

## Mocking with examples

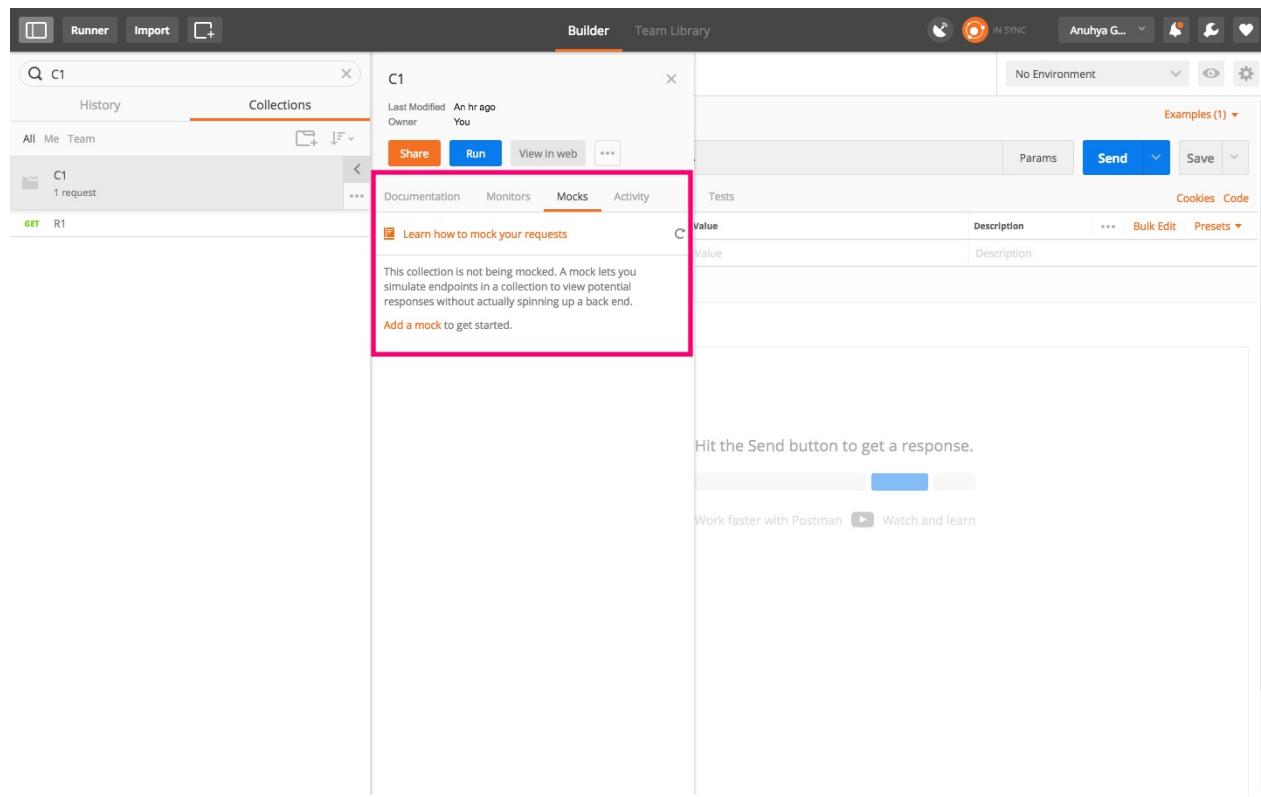
In the previous steps, we prepared the collection, request, and example response necessary for us to get started with mocking. So let's continue with the next steps.

### Step 4: Creating a mock (M1) for the collection (C1)

There are two ways to create a mock for our collection: 1) using the Postman app and 2) [using the Postman Pro API](#). In this example, we will mock a collection using the Postman app.

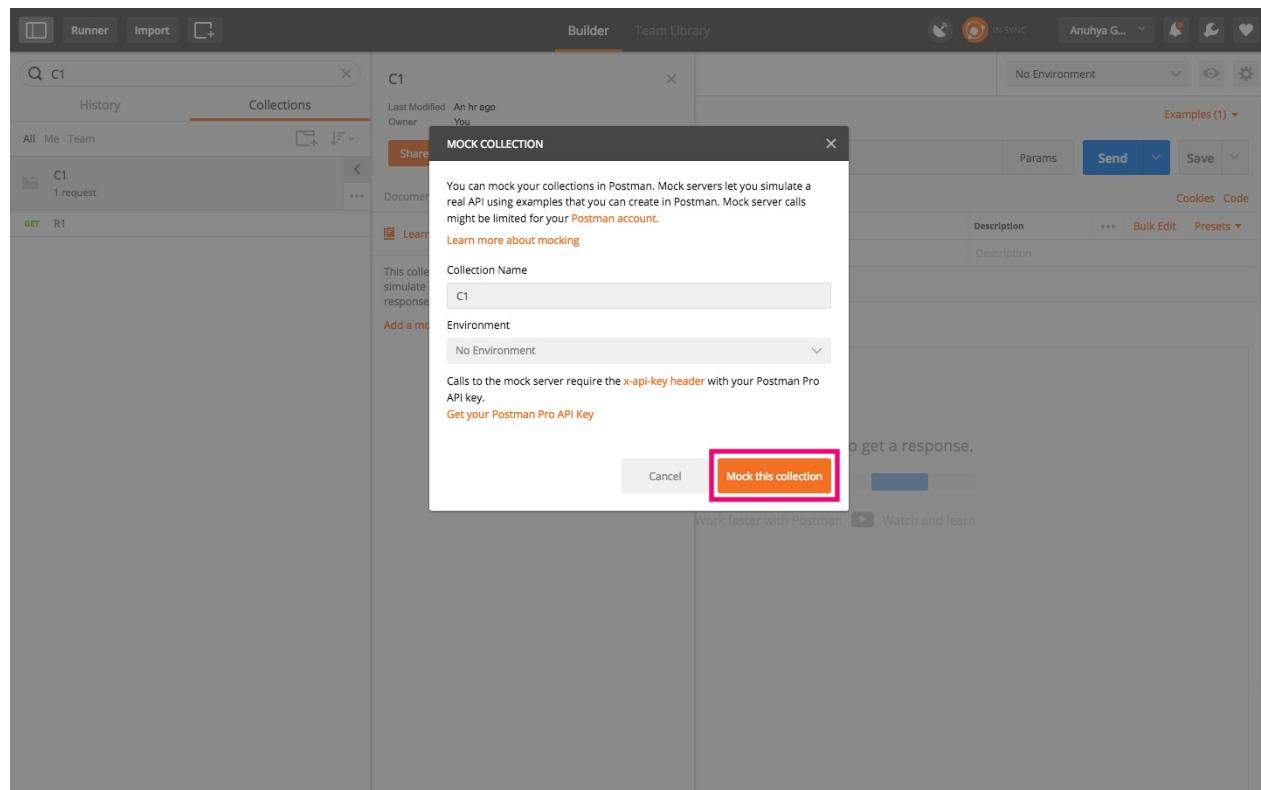
From the Postman app, click on the right angle bracket (>) next to the collection you wish to mock to expand the collection details view.

## Mocking with examples



Under the **Mocks** tab, click the **Add a mock** link to open the **MOCK COLLECTION** modal. Here, you can choose a corresponding environment to include in your mock.

We are not using any environment variables in our single saved example (P1), therefore we are going to go ahead and create a mock with No Environment chosen. It's important to note that if your saved example has an environment variable in the URL, for example, /my/path and you do not provide the corresponding environment when creating the mock, trying to mock that particular request will not work.



Once you mock the collection, it will be visible under the Mocks tab of the collection details view. You can also see the mock URL we will need for the next step.

### Step 5: Sending a request using the mock server (M1)

Now that we have created our mock M1, let's try sending a request to this mock endpoint. Copy the mock URL from the mock we created in the previous step, and paste it into a new request, with an undefined path in this case <https://b75a340e-4268-4b20-8f5f-3fcf8f37cec6.mock.pstmn.io>. Under the **Headers** tab of this new request, add the x-api-key header, with the value of your [Postman Pro API key](#).

## Mocking with examples

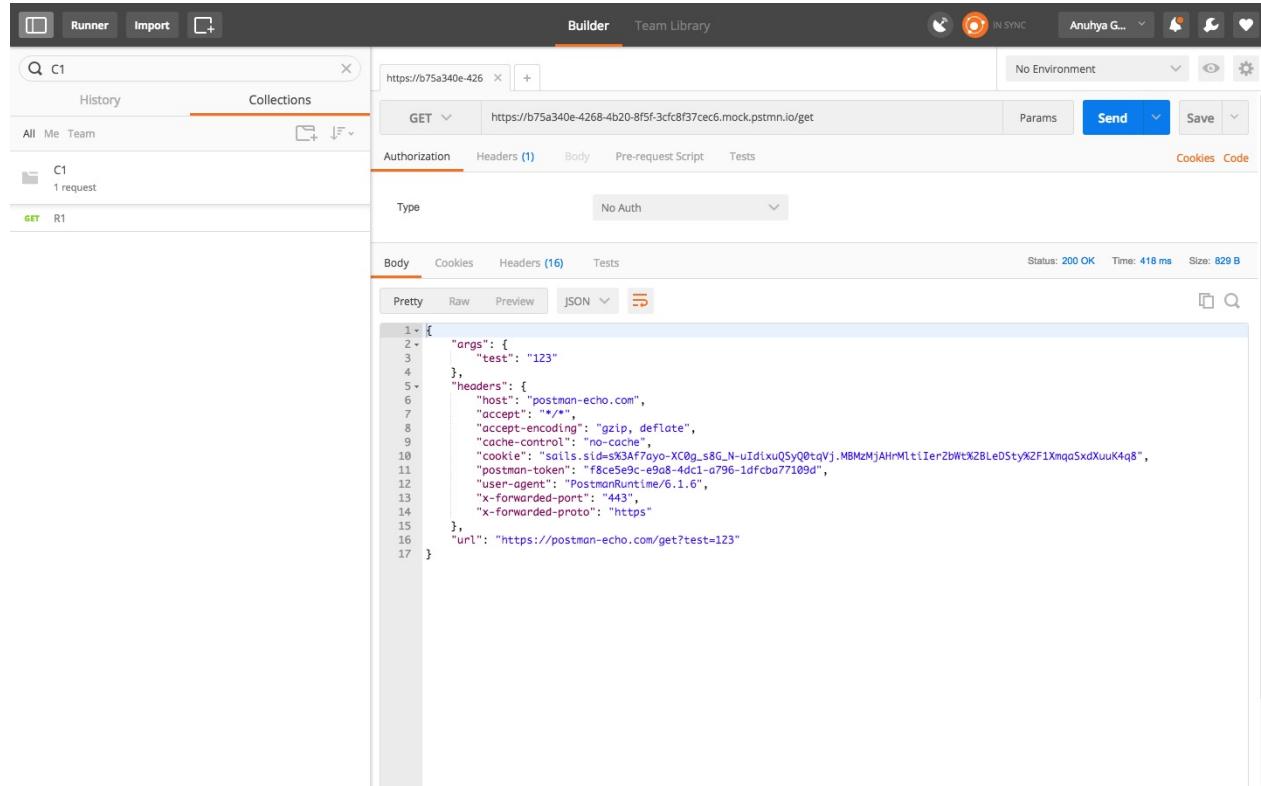
The screenshot shows the Postman Builder interface. A collection named 'C1' is selected. In the center, there's a 'Mocks' tab with a URL listed: <https://b75a340e-4268-4b20-8f5f-3fcf8f37cec6.mock.pstmn.io>. Below the URL is a button labeled 'Copy Mock URL'. To the right, a message says 'Hit the Send button to get a response.' At the bottom, there's a note: 'Work faster with Postman' with a 'Watch and learn' link.

Sending a request to this mock endpoint with an undefined path returns an error. As you can see, there is no matching saved example with the path " and the request method GET. Responses returned by the mock service are entirely dependent on your saved examples and the included URL and request method type.

The screenshot shows the Postman Runner interface. A request is being made to the URL: <https://b75a340e-4268-4b20-8f5f-3fcf8f37cec6.mock.pstmn.io> using the GET method. The response status is 404 Not Found, with a message: 'We were unable to find any matching requests for the mock path (i.e. undefined) in your collection.' The response body is a JSON object:

```
1. {  
2.   "error": {  
3.     "name": "mockRequestNotFoundError",  
4.     "message": "We were unable to find any matching requests for the mock path (i.e. undefined) in your collection."  
5.   }  
6. }
```

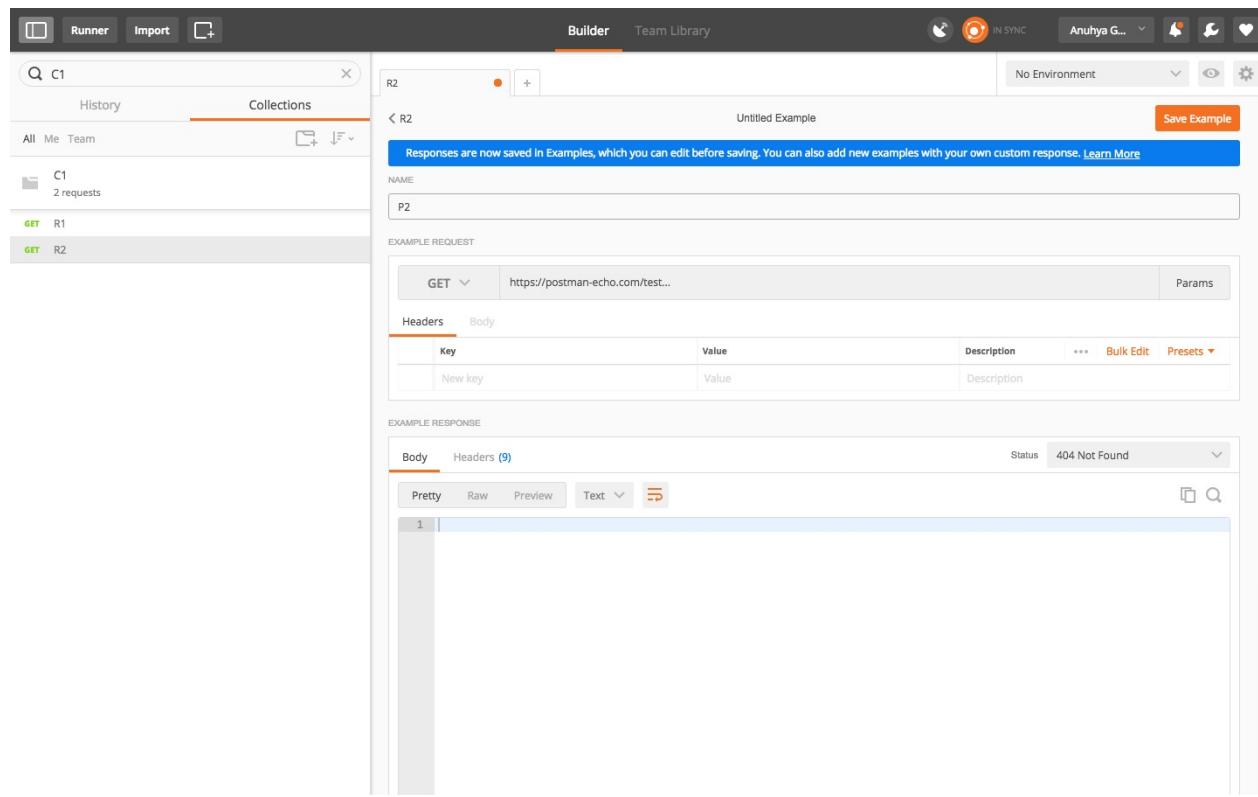
We do, however, have a saved example with the path /get and the request method GET. So sending a GET request to the URL <https://b75a340e-4268-4b20-8f5f-3fcf8f37cec6.mock.pstmn.io/get> will return the proper response we are looking for.



```
1 {
2   "args": {
3     "test": "123"
4   },
5   "headers": {
6     "host": "postman-echo.com",
7     "accept": "*/*",
8     "accept-encoding": "gzip, deflate",
9     "cache-control": "no-cache",
10    "cookie": "sails.sid=s%3AF7yo-XC0g_s8G_N-uIdixuQ5yQ0taVj.MBmzMjAHrMltIer2bWtK2LeDStyx2F1XmqoSxdXuuK4q8",
11    "postman-token": "f8ce5e9c-e908-4dc1-a796-1dfcba77109d",
12    "user-agent": "PostmanRuntime/6.1.6",
13    "x-forwarded-port": "443",
14    "x-forwarded-proto": "https"
15  },
16  "url": "https://postman-echo.com/get?test=123"
17 }
```

## Adding more examples

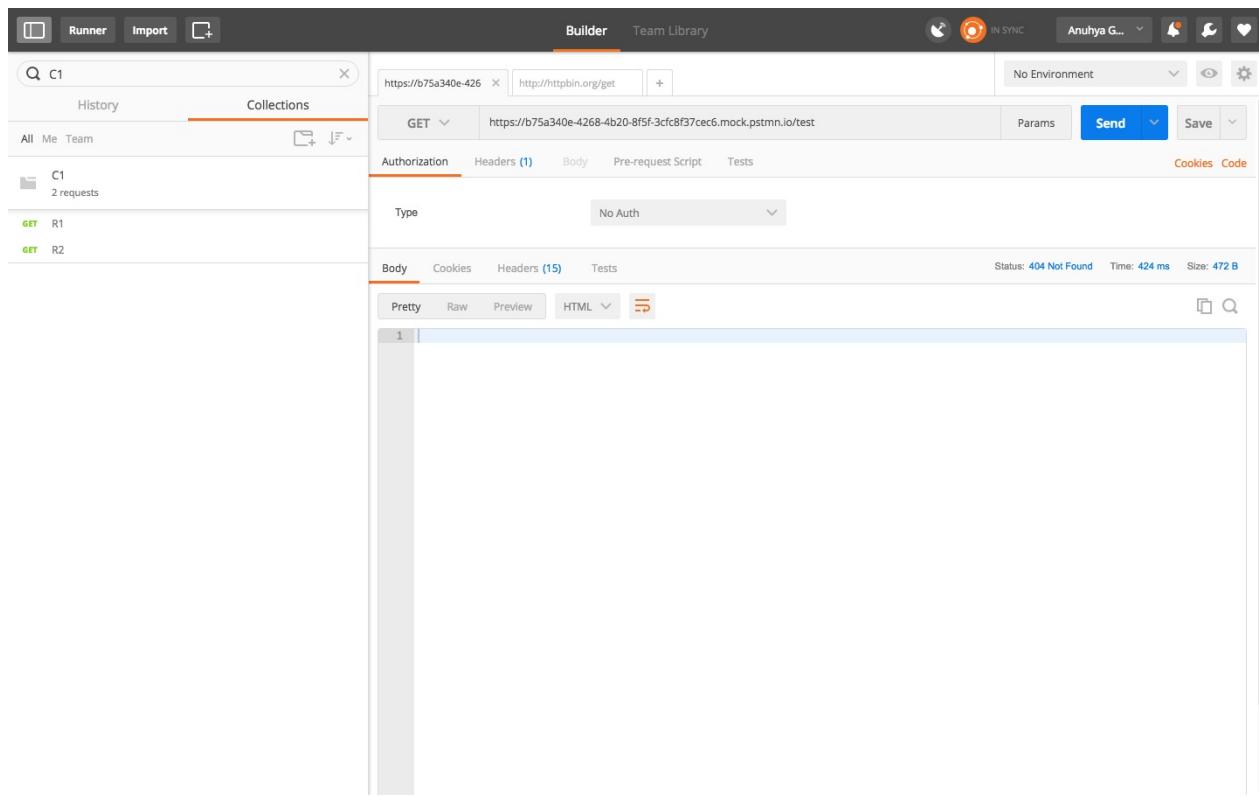
To further illustrate how responses from the mock service are entirely dependent on your saved examples, let's try adding another example to this collection. We'll repeat steps 1 to 3 of saving the request to a collection and saving the response as an example, with a new URL <https://postman-echo.com/test>.



Sending a GET request to <https://postman-echo.com/test> returns a 404 error which we will then save as another example. Our collection C1 now has two requests and two saved examples:

- GET > /get
- GET > /test/

Mocking the /test mock path also gives us our expected 404 response.



Your examples might vary depending on the URL endpoint, request method type, or status code. If you have multiple examples saved to the same mock, you can choose to save each example under a unique URL endpoint like we saw in this example with /get and /test. Alternatively, if you have saved examples with different response status codes, you can send an authenticated request to the mock endpoint along with the x-mock-response-code header specifying which integer response code your returned response should match.

Learn more about the [matching algorithm](#) for mocks.

And we're done! We have walked through how to create a collection, save requests, save examples, create a mock, and use a mock.

# Mocking with the Postman Pro API

You can [mock a collection](#) directly from the Postman app. Additionally, you can create a mock using the Postman Pro API. Let's walk through this step by step.

## Set up a collection for mocking

In this example, we have a Collection testAPI with corresponding environment testAPIEnv. Let's set up a mock service to enable your front-end team to simulate each endpoint in testAPI and view the various responses.

Navigate to every request in the Collection testAPI that you would like to include in this simulation, and [save responses](#) with details about the response body, header or status codes that you would like to see returned by that endpoint. In this example, we will save 2 responses with status codes of 200 and 401 for this particular request. Once you save the desired responses, the Collection is ready for mocking.

**Note:** In addition to mocking a collection with a saved response, you can also [mock a request and response using examples](#).

The screenshot shows the Postman Pro application interface. The left sidebar displays a collection named 'testAPI' containing two requests: 'testAPI Request 1' and 'testAPI Request 2'. The main workspace shows 'testAPI Request 1' being edited. The 'Saved Responses' section, which contains entries for status codes 200 and 401, is highlighted with a red box. The 'Send' button is visible at the bottom right of the request editor.

## Retrieve information needed for mock creation

Let's retrieve the collectionId of testAPI using the [Postman Pro API](#). Get a list of all your Collections using the [GET All Collections endpoint](#). Search for the name of your Collection and retrieve the uid from the results, which will be used as the collectionId in the next step.

```

75  {
76    "id": "b1abb7ac-ce97-ac09-428f-89323b4913eb",
77    "name": "testAPI",
78    "owner": "463161",
79    "uid": "463161-b1abb7ac-ce97-ac09-428f-89323b4913eb"
80  },
  
```

You can also use the Postman app to retrieve the collectionId. Find the Collection in your app and hit View Docs. The collectionId is visible in the documentation url:

```
https://documenter.getpostman.com/collection/view/{{collectionId}}
```

As an optional step, you can include an environment template as a part of your simulation by retrieving the environmentId of testAPIEnv using the [Postman Pro API](#). Get a list of all your environments using the [GET All Environments endpoint](#). Search for the name of your environment and retrieve the uid from the results, which will be used as the environmentId in the next step.

```

33  {
34    "id": "7cd2f47b-2f9f-210f-4c9f-ce61fe15b779",
35    "name": "testAPIEnv",
36    "owner": "463161",
37    "uid": "463161-7cd2f47b-2f9f-210f-4c9f-ce61fe15b779"
38  },
  
```

## Create a mock using the Postman Pro API

Create a mock using the [POST Create Mock endpoint](#) with the collectionId and environmentId you retrieved previously.

The screenshot shows the Postman Pro application window. In the top navigation bar, 'Builder' is selected. The main area shows a POST request to 'https://api.getpostman.com/mocks'. The 'Body' tab is active, displaying a JSON payload:

```

1 - [
2 -   "mock": {
3 -     "collection": "[{collectionId}]",
4 -     "environment": "[{environmentId}]"
5 -   }
6 - ]

```

To the right of the request, the 'testAPIEnv' environment is selected, showing its details:

- collectionId: 463161-b1abb7ac-ce97-ac09-428f-89323b4913eb
- environmentId: 463161-7cd2f47b-2f9f-210f-4c9f-ce61fe15b779
- Globals: No variables

At the bottom of the interface, the status bar indicates: Status: 200 OK, Time: 487 ms, Size: 617 B.

Verify that the mock has been created using the [GET All Mocks endpoint](#), and your Collection is now ready to be simulated.

## Run the mock service

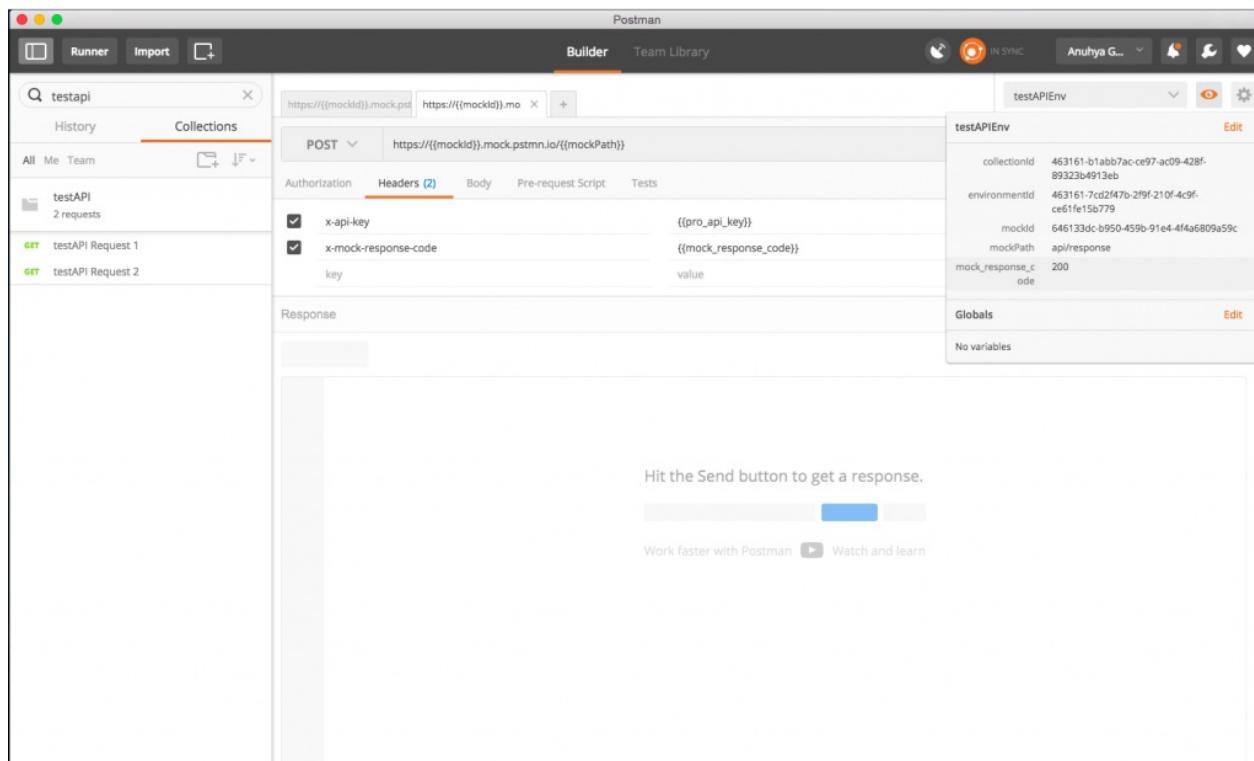
**Mock your Collection using the following url:**

```
https://{{mockId}}.mock.pstmn.io/{{mockPath}}
```

- mockId is the id that you received upon creating the mock and can be retrieved using the [GET All Mocks endpoint](#).
- mockPath is the path of your request that you'd like to mock, for example api/response.

**Add the request header(s):**

- Mock requests require one mandatory header, x-api-key, which is your Postman Pro API key for authentication. Don't have a Postman Pro API key? [Create one here](#).
- Mock requests also accept another optional header, x-mock-response-code, which specifies which integer response code your returned response should match. For example, 500 will return only a 500 response. If this header is not provided, the closest match of any response code will be returned.



## Mock requests and responses with examples

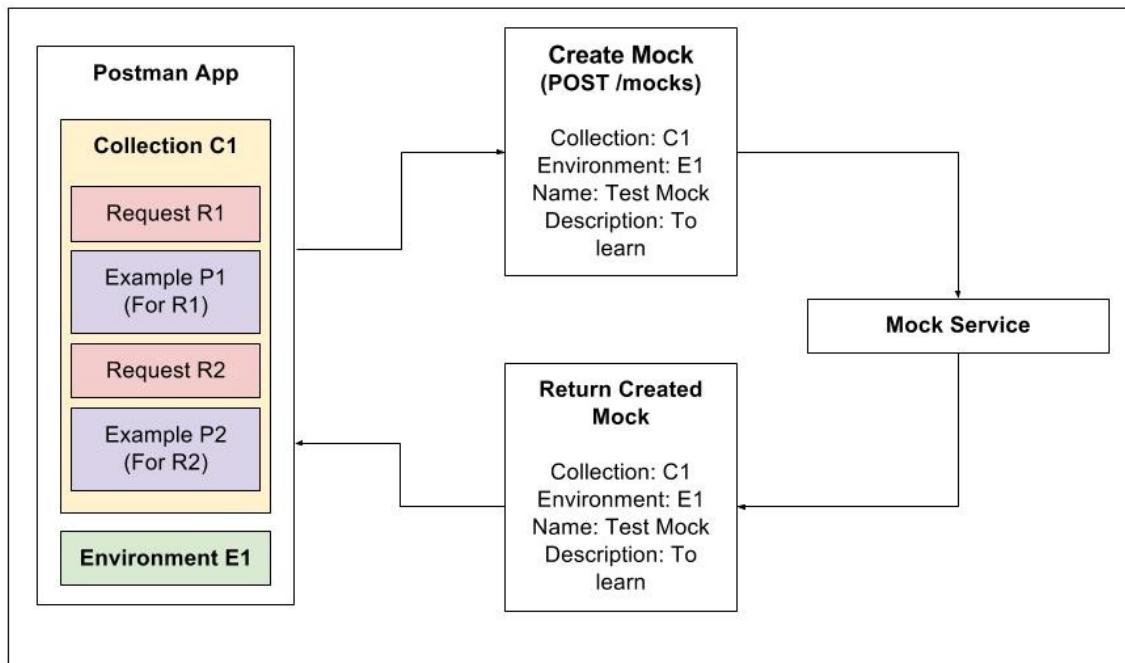
In the previous example, we used a saved response to mock our collection. You can also [mock a request and response using examples](#) in Postman before sending the actual request or setting up a single endpoint to return the response. With examples, you can mock raw responses and save them. Then, you'll be able to generate a mock endpoint for each of them using Postman's mock service.

# Matching algorithm

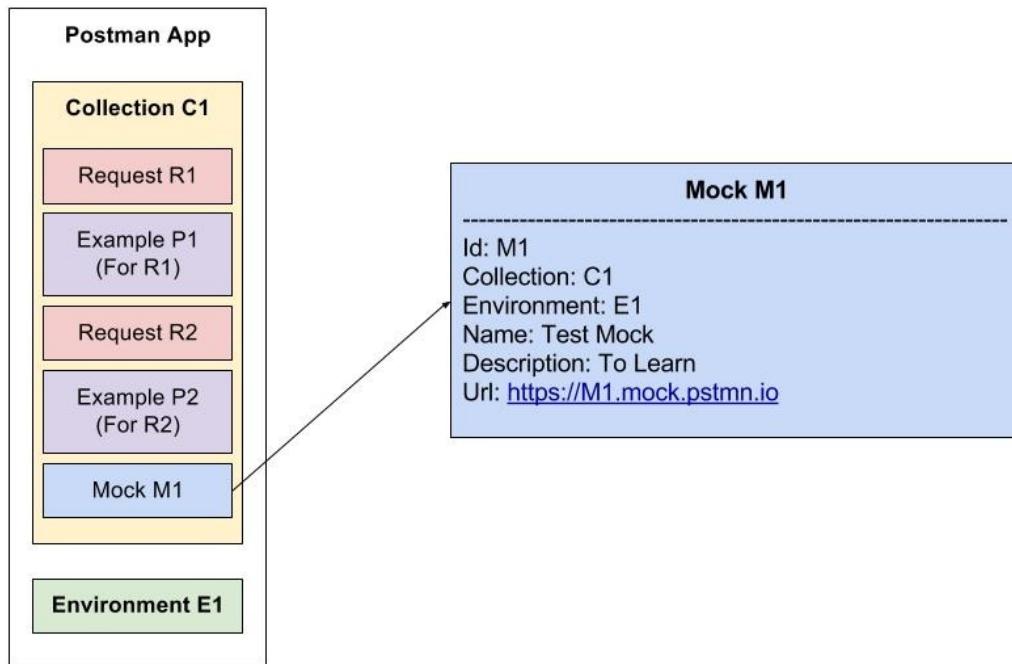
Using the Postman mock service requires the following: a collection with requests, a mock server, and saved request examples. You can save as many examples to a collection as you please, and the mock server will return these examples predictably. But how exactly does the mock decide which example to return?

## Matching algorithm for mocks

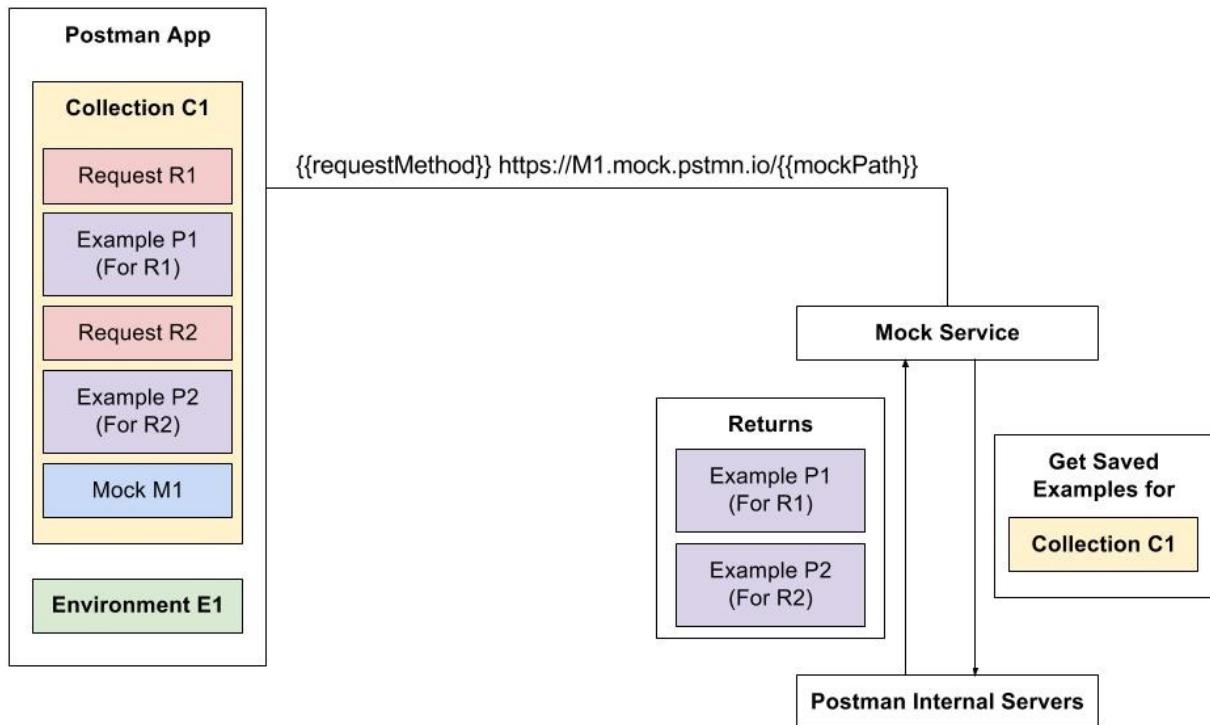
To begin, let's start with an example.



When a mock is created using either the Postman Pro API or the Postman app, a call is made to the Postman servers that associates a particular collection (and environment if you choose one) with a newly created mock. The collection C1 that we just mocked is now associated with the new mock M1.



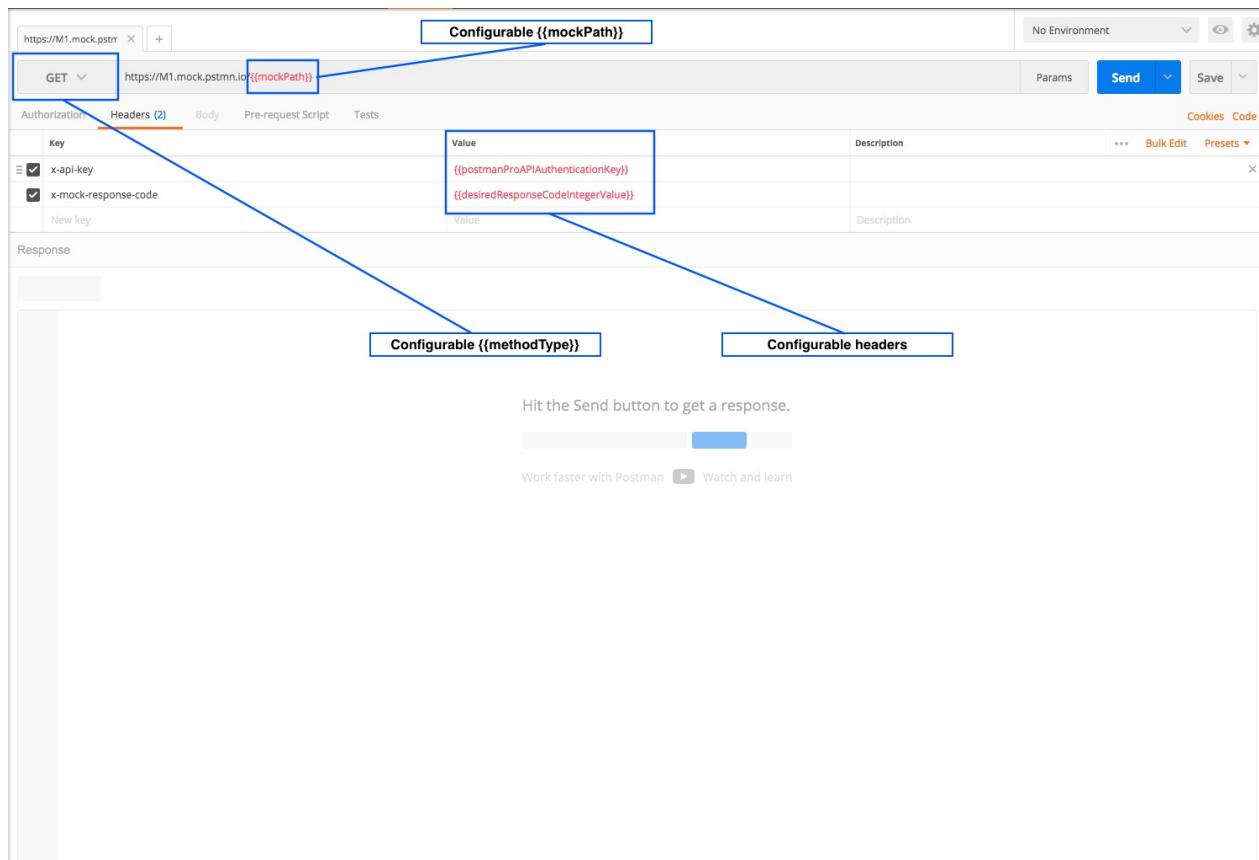
When we use the mock M1 via the mock URL <https://M1.mock.pstmn.io> in the Postman app, the mock service will retrieve all saved examples from the Postman servers for that particular collection before it begins the matching process.



Now that the mock service has all the saved examples for the current collection, it will now iteratively pair the incoming request with the closest matching example.

The incoming request can have several configurable variables, such as `RequestMethod` and `mockPath`. The `RequestMethod` variable corresponds to any valid HTTP request method (e.g. GET, POST, PUT, PATCH, DELETE, etc.), and the `mockPath` refers to any valid string path (e.g. `/`, `/test`, `/test/path`, `/test/path/1`).

The request requires an authentication header `x-api-key` corresponding to the [Postman Pro API Key](#), and optionally accepts a header `x-mock-response-code` corresponding to the desired response code of the mock request. For example, you could request a 200, 400, 404, or 500 response for a particular endpoint.



Keeping these various configurable elements in mind, let's take a look at the matching algorithm logic.

- Properly formatted responses** Any responses that are not in the expected format are removed from the matching process.
- HTTP method** Any responses that are not the same HTTP method type are removed from the matching process. For example: if the mock request you sent was POST to <https://M1.mock.pstmn.io/test>, all saved examples whose method type is not POST will be disregarded.
- Filter by URL** The matching process will now examine each saved example, and iterate over every possibility. Compare the mockPath of the input URL with that of the saved example. If the input URL was <https://M1.mock.pstmn.io/test> and the example currently being examined had a URL of <https://google.com/help>, the mock service would compare /test with /help. While comparing URLs, a step-by-step matching is conducted. Each consecutive step that the matching algorithm traverses reduces the matching threshold of the current example response. For example:
  - Try to match the input path with the example path exactly as it is. The max value is set as the matching threshold.
  - Try to strip trailing slashes and match the input path with the example path. The threshold is reduced by a certain value, n.
  - Try to additionally lowercase the input path and the example path. The threshold is reduced by a greater value, n + m.

- Try to additionally strip out alphanumeric ids from the input path and the example path. The threshold is reduced further,  $n + 2m$ .
  - If all steps fail, this saved example is not an eligible response.
4. **Response code** If the `x-mock-response-code` header is explicitly provided, filter out all examples that do not have a matching response code.
  5. **Highest threshold value** Sort the remaining filtered responses in descending order and return the response with the highest threshold value.

And there we have it! This is how the mock service finds and returns the appropriate response to a mock request.

# Intro to the Postman API

The [Postman API](#) has several endpoints to help you integrate Postman even more deeply with your development toolchain. You can add new collections, update existing collections, update environments, add and run monitors directly through the API. This API allows you to programmatically access data stored in your Postman account with ease.

The easiest way to get started with the API is to click the **Run in Postman** button at the top of the [Postman API documentation page](#) and use the Postman app to send requests.

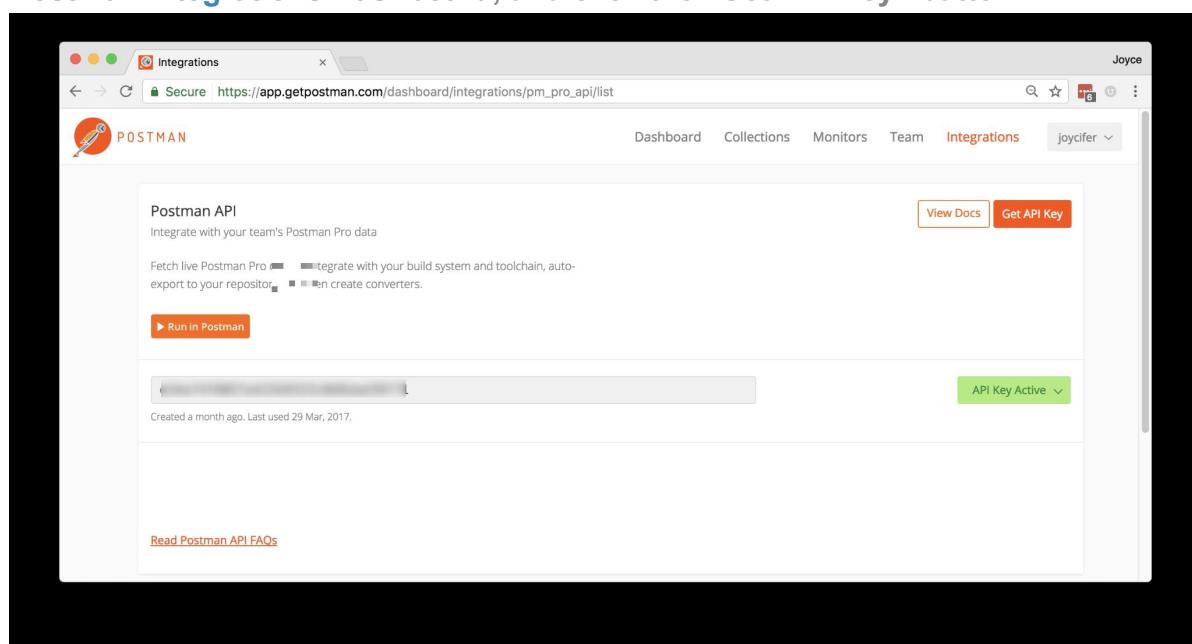
## Postman API overview

1. You need a valid API Key to send requests to the API endpoints. Postman users can get a key from the [integrations dashboard](#).
2. The API is rate limited.
3. Using the API, you can add and update collections, environments, and users. Run monitors, create a mock server, and so much more.
4. Review more [documentation for the Postman API](#).

## Authentication

An API Key is required to be sent as part of every request to the Postman API, in the form of an X-Api-Key request header.

If you do not have an API Key, you can easily generate one by heading over to the [Postman Integrations Dashboard](#), and click the “Get API Key” button.



An API Key tells our API server that the request it received came from you. Everything that you have access to in Postman is accessible with an API Key that is generated by you.

For ease of use inside Postman, you could store your API key in an [environment variable](#) called `postman_api_key` and this collection will automatically use it to make API calls.

## Rate Limits

API access rate limits are applied at a per-key basis in unit time. Access to the API using a key is limited to **60 requests per minute**. In addition, every API response is accompanied by the following set of headers to identify the status of your consumption.

HeaderDescriptionX-RateLimit-LimitThe maximum number of requests that the consumer is permitted to make per minute.X-RateLimit-RemainingThe number of requests remaining in the current rate limit window.X-RateLimit-ResetThe time at which the current rate limit window resets in UTC epoch seconds.

## Free API calls with your Postman account

Your Postman account gives you a limited number of free Postman API calls per month. You can check your usage limits through the [Postman API](#) itself or the [account usage page](#).

# Continuous Integration

Continuous Integration (CI) is a development practice that requires developers to regularly merge code updates into a shared repository. It involves the process of automating the build and testing of code every time a developer commits code updates.

Let's access collections using the Postman API to run inside your Continuous Integration / Continuous Deployment (CI/CD) environments.

Before we get started:

- Ensure you have a CI system setup which can run shell commands and that you have access to modify the same.
- If you don't already have a [Postman API key, get one now](#).
- Make sure you have a Postman Collection that tests your localhost server, and note the UID of the collection.

## Step 1: Install Node

You may skip this step if your CI already has Node installed.

Follow the [steps to download Node](#) which is specific to your CI's platform. Otherwise, some CI has configuration which simply pre-installs Node. Ensure you are using NodeJS v4 or above.

## Step 2: Install Newman

[Newman](#) is a command-line tool that allows you to run a collection in your local development environment or on your own server. The following command installs Newman in your CI.

```
npm i newman -g;
```

## Step 3: Run Newman

Run the following Newman command with the appropriate parameters:

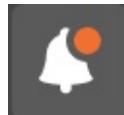
```
newman run https://api.getpostman.com/collections/{{collection_uid}}?apikey={{postman-api-key-here}}
```

If you need to provide an environment to the collection, change the above command to the following:

```
newman run https://api.getpostman.com/collections/{{collection_uid}}?apikey={{postman-api-key-here}}
```

# Notifications

The Postman app notifies you in case of some important events. These notifications can be



accessed by clicking on the icon in header bar.

A screenshot of the Postman mobile application's notifications screen. The header bar includes icons for signal strength, sync status (IN SYNC), user profile (sankalp0o), and notifications. A large, semi-transparent 'NOTIFICATIONS' card is overlaid on the main content. The card lists five notifications, each with a circular profile picture, the sender's name, the collection shared, the date, and a 'Subscribe' button. The notifications are:

- Ankit Jaggi shared the collection **Dummy Collection** Yesterday, 3:35 pm **Subscribe**
- kamalakannan@getpostman.com shared the collection **BuildKite** 25 Apr, 2017 **Subscribe**
- kamalakannan@getpostman.com shared the collection **BuildKite Agent** 25 Apr, 2017 **Subscribe**
- Suhas<sup>BIGB</sup> shared the collection **Security Validations** 19 Apr, 2017 **Subscribe**
- "><img src=x onerror=alert(document.domain)>" shared the collection **Documenter API** 19 Apr, 2017 **Subscribe**

You will get notified in case of the following events:

- **Team Library notifications**

- A user in your team shares a collection
- A user in your team unshares a collection
- A user in your team deletes a shared collection
- A user in your team subscribes to a collection that you own
- A new user joins your team
- A user in your team publishes a shared collection
- A user in your team unpublishes a shared collection

- **Monitor notifications**

- A new team monitor is created on a collection you are subscribed to
- A monitor that you're subscribed to fails a test
- A monitor that you're subscribed to fails because of an error
- A monitor that you're subscribed to fails a test multiple times in a row
- A monitor that you're subscribed to fails because of an error multiple times in a row
- A monitor that you're subscribed to recovers from a test failure
- A monitor that you're subscribed to recovers from an error

- **Monitoring notifications (for admins and monitor owners)**

- The monthly monitoring quota for the team is about to be reached
- Monthly monitoring quota is exceeded and overage starts