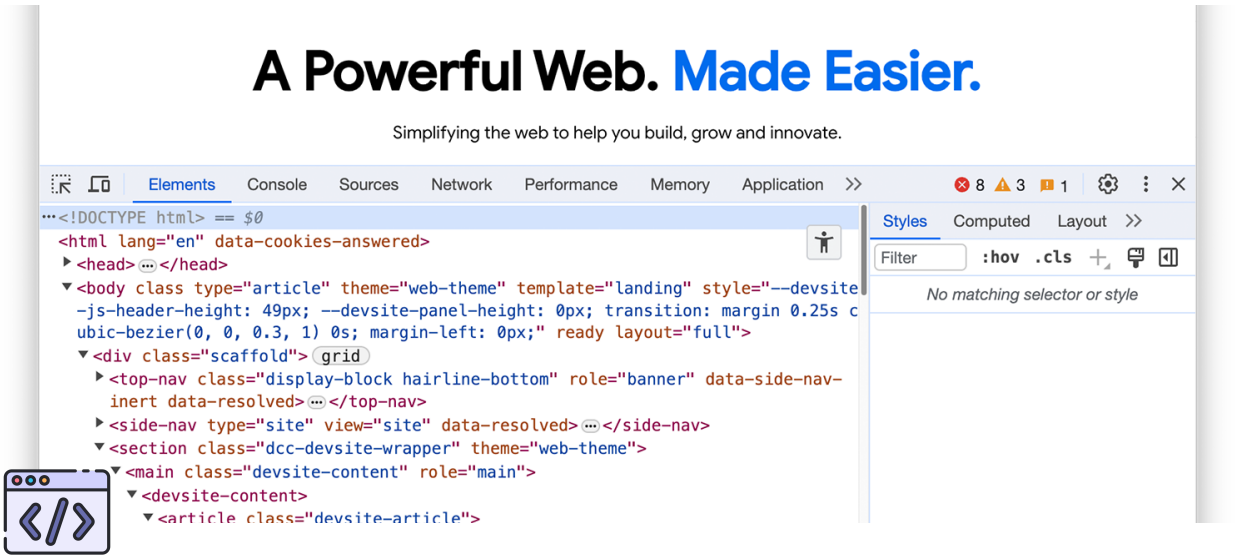


Dev tools

What is DevTools?

DevTools (Developer Tools) are built-in tools in modern web browsers that help developers inspect, debug, and optimize web pages or applications. They provide real-time insights into the structure, behavior, and performance of websites.

Why Do We Use DevTools?

- **Debugging**: Identify and fix issues in HTML, CSS, and JavaScript.
- **Testing**: Simulate different screen sizes, devices, and network conditions.
- **Optimization**: Analyze page performance and reduce load times.
- **Accessibility**: Ensure websites meet accessibility standards.
- **Live Editing**: Make real-time changes to code for rapid prototyping.

What Tools Did Developers Use Before Browser DevTools?

Before browser DevTools became mainstream, developers relied on external tools and techniques like:

1. **Firebug (for Firefox)**: A popular third-party extension for inspecting and debugging web pages.
2. **View Source**: Manually checking the page's source code using the browser's "View Source" feature.
3. **Console Logging**: Using `alert()` or `console.log()` for debugging JavaScript.
4. **Text Editors/IDEs**: Developers would modify files locally and then reload the browser to see changes.
5. **Network Tools**: Standalone tools like Wireshark or Fiddler for analyzing network requests.

Open Dev Tools

There are many ways to open Dev Tools, because different users want quick access to different parts of the Dev Tools UI.

1.Open panels with shortcuts: Elements, Console, or your last panel

If you prefer keyboard, press a shortcut in Chrome depending on your operating system:

OS	Elements	Console	Your last panel
Windows or Linux	Ctrl + Shift + **C**	Ctrl + Shift + **J**	F12Ctrl + Shift + **I**
Mac	Cmd + Option + **C**	Cmd + Option + **J**	Fn + F12Cmd + Option + **I**

Here's an easy way to memorize the shortcuts:

- **C** stands for CSS.
- **J** for JavaScript.
- **I** designates your choice.

The **C** shortcut opens the **Elements** panel in ink_selection inspector mode. This mode shows you helpful tooltips when you hover over elements on a page. You can also click any element to view its CSS

in the
Elements > **Styles** tab.

2.Open Dev Tools from Chrome menus

If you prefer UI, you can access DevTools from drop-down menus in Chrome.

Open the Elements panel to inspect the DOM or CSS

To inspect, right-click an element on a page and select

Inspect.

3.Auto-open Dev Tools on every new tab

Run Chrome from the command line and pass the `--auto-open-devtools-for-tabs` flag:

1. Quit any running Chrome instance.

   Key point: This flag works only for the
   first Chrome instance you open. If it doesn't work for you, for example,
   on Windows, make sure to end any residing Chrome processes from the
   Task Manager.

2. Run your favorite terminal or command line application.
3. Depending on your operating system, run the following command:

- macOS:
- `open -a "Google Chrome" --args --auto-open-devtools-for-tabs`
- Windows:
- `start chrome --auto-open-devtools-for-tabs`
- Linux:
- `google-chrome --auto-open-devtools-for-tabs`

Dev Tools will automatically open for every new tab until you close Chrome.

Customize Dev Tools

Settings

settings **Settings** > **Preferences** contains many options for customizing DevTools.

See Open Settings and Preferences.

Dark theme

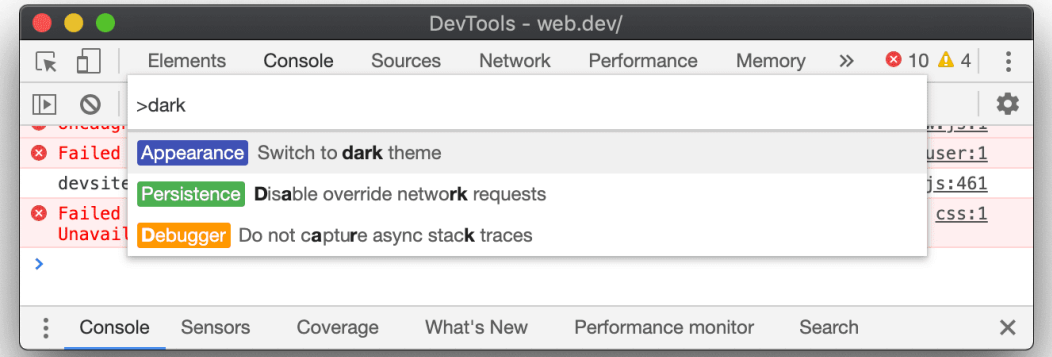
You can enable dark theme in

Settings

or the

Command Menu.

1. Open the Command Menu.
2. Start typing `dark`, select the **Switch to dark theme** command, and then press Enter to run the command.



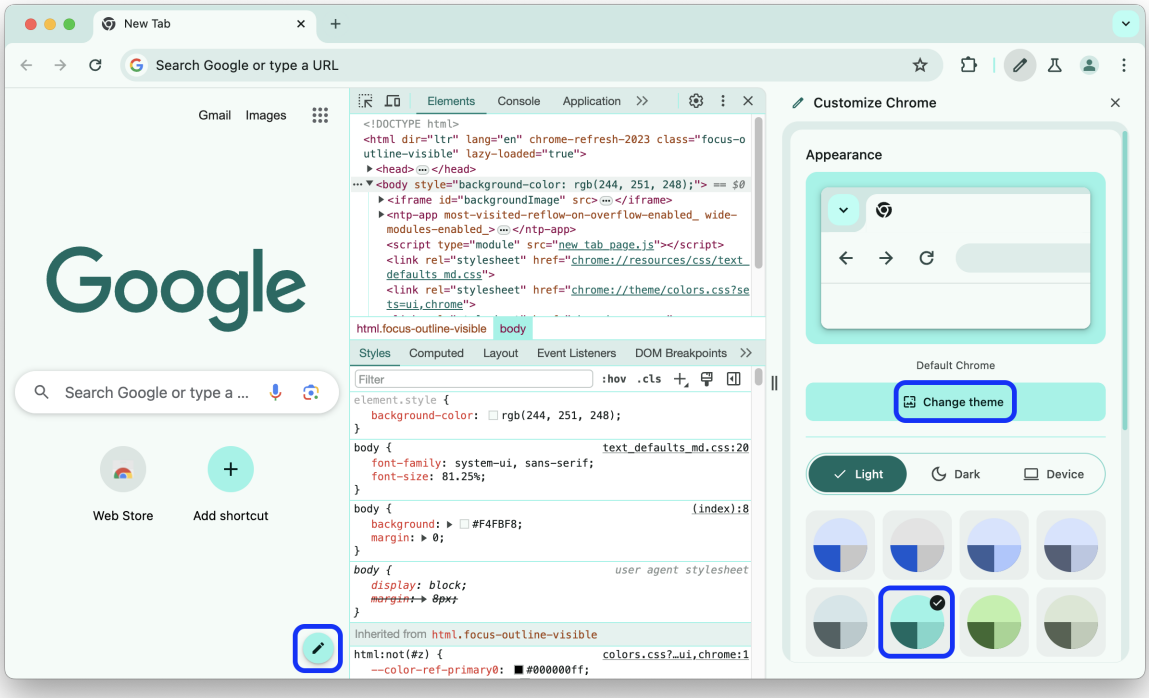
3. Alternatively, set your theme in settings **Settings** > **Preferences** > **Appearance** > **Themes**.

Dynamic theme

DevTools can automatically match Chrome's color theme.

To set a theme:

1. Open a new tab and click edit **Customize Chrome** in the right bottom corner.
2. In **Appearance**, pick a theme through wallpaper **Change themes** or select a color palette.

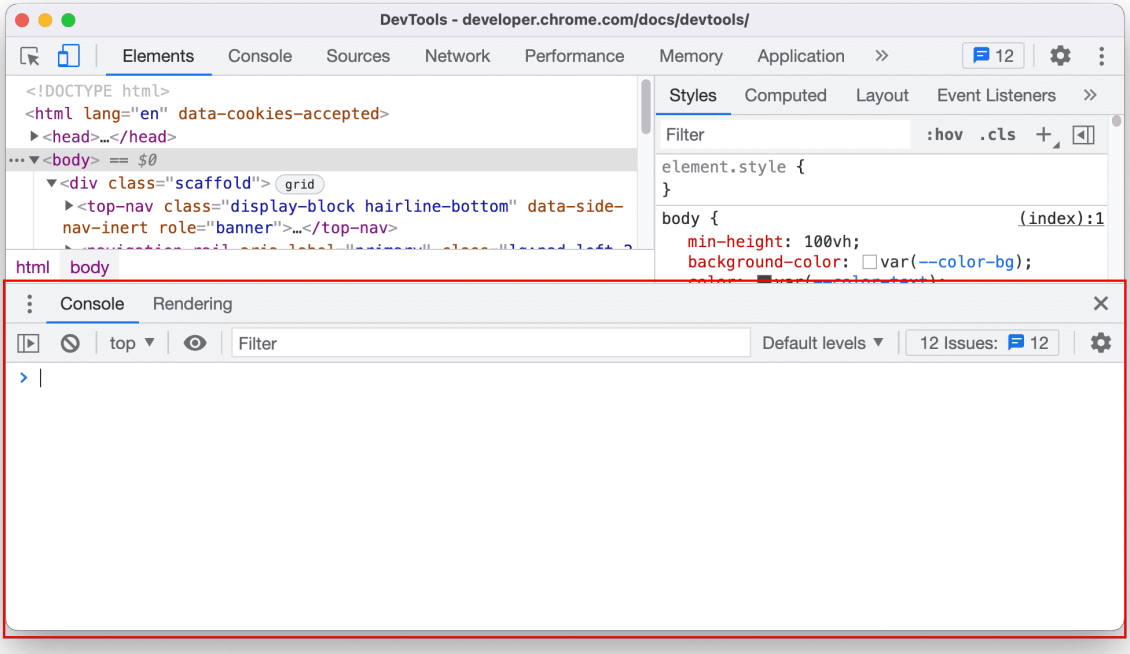


To turn off dynamic theming, clear settings **Settings** > **Preferences** > **Appearance** > check_box_outline_blank **Match Chrome color scheme** and reload DevTools.

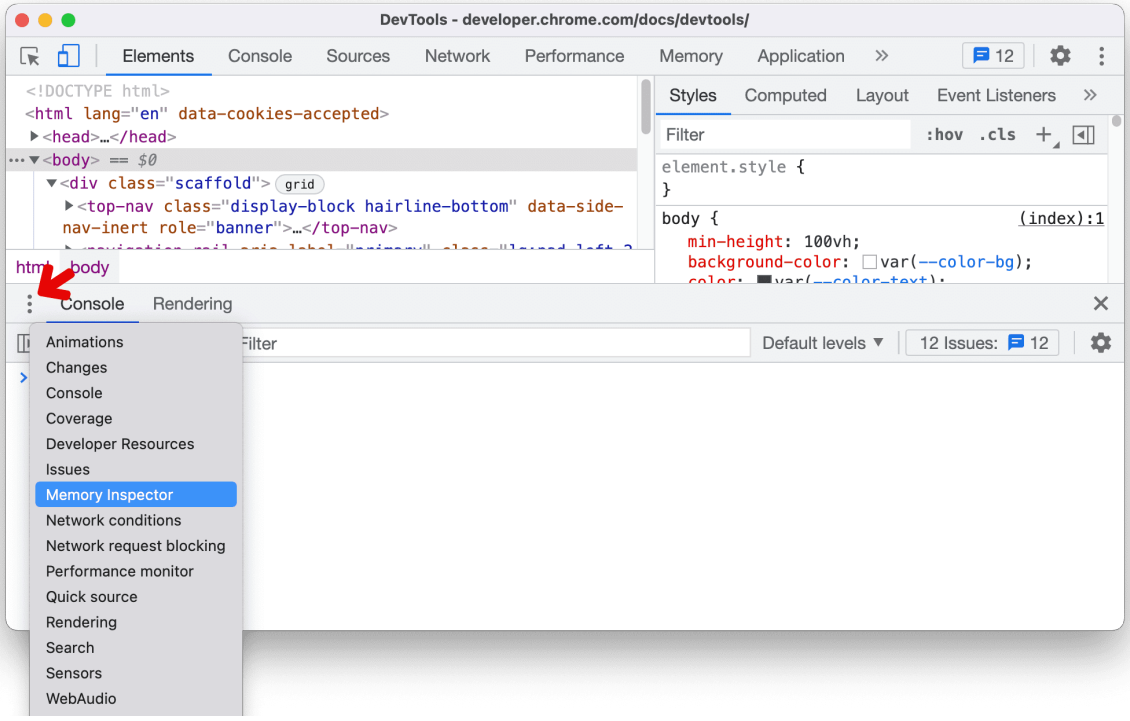
Drawer

The **Drawer** contains many hidden features.

Press Escape to open or close the Drawer.



Click ⋮ **More Tools** to open other **Drawer** tabs.

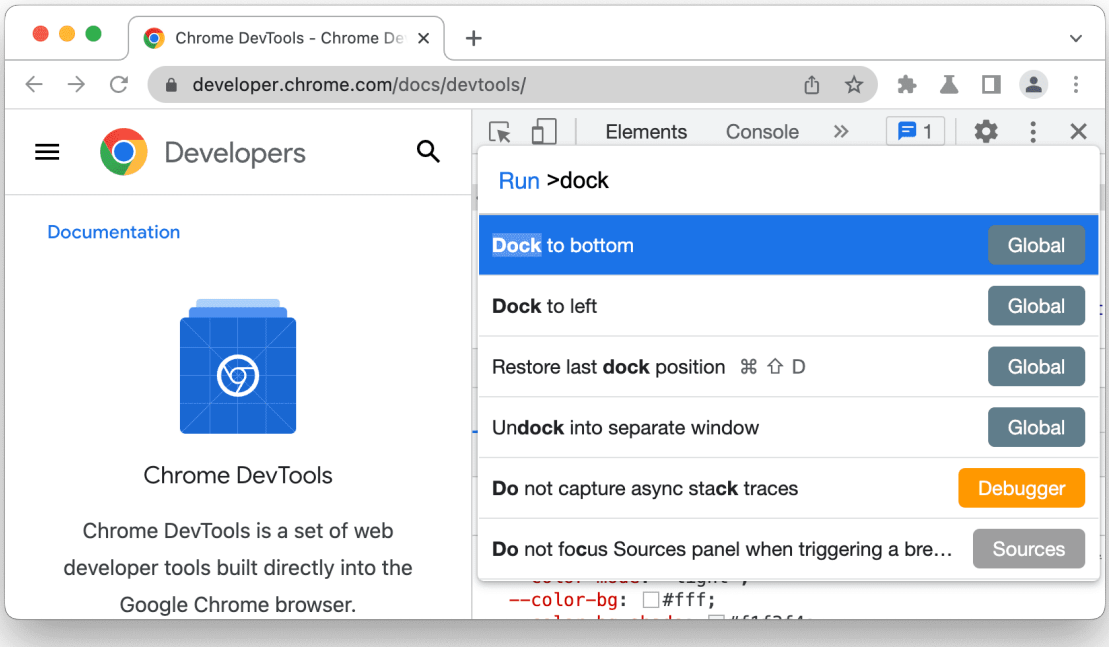


Change Dev Tools placement

By default, Dev Tools is docked to the right of your viewport. You can also dock to the bottom or left sides or undock Dev Tools into a separate window.

You can change the placement of Dev Tools in two ways:

- **Main Menu**: Open more_vert **Customize And Control DevTools** and click:
  - ad_group **Undock into separate window**
  - dock_to_right **Dock to left**
  - dock_to_bottom **Dock to bottom**
  - dock_to_left **Dock to right**
- **Command Menu**:
  1. Open the Command Menu.
  2. Start typing `dock` and select one of the suggested options: dock to bottom, left, right, undock, or restore last dock position.



To toggle **Restore last dock position** with a keyboard shortcut, press:

- On Linux or Windows: `Ctrl+Shift+D`
- On macOS: `Cmd+Shift+D`

Reorder panels, tabs, and panes

To change ordering, click and drag left or right any of the following:

- Panels at the top of DevTools.
- Panes in the **Elements** panel such as **Styles**, **Computed**, **Layout**, and others.
- Panes in the **Sources** panel such as **Page**, **Workspace**, **Overrides**, and others.
- **Drawer** tabs at the bottom of DevTools.

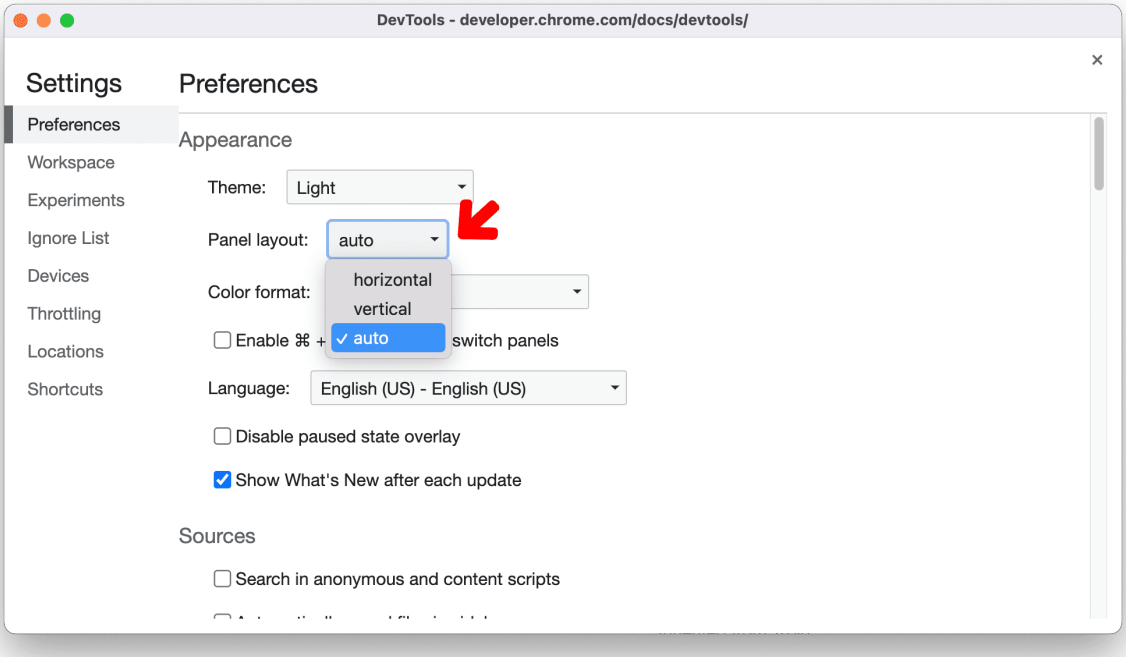
Additionally, you can move panels and tabs up and down to and from the **Drawer**. To do this, right-click the panel or tab and select **Move to top** or **Move to bottom** from the drop-down menu.

Your custom tab order persists across DevTools sessions.

Panel layout

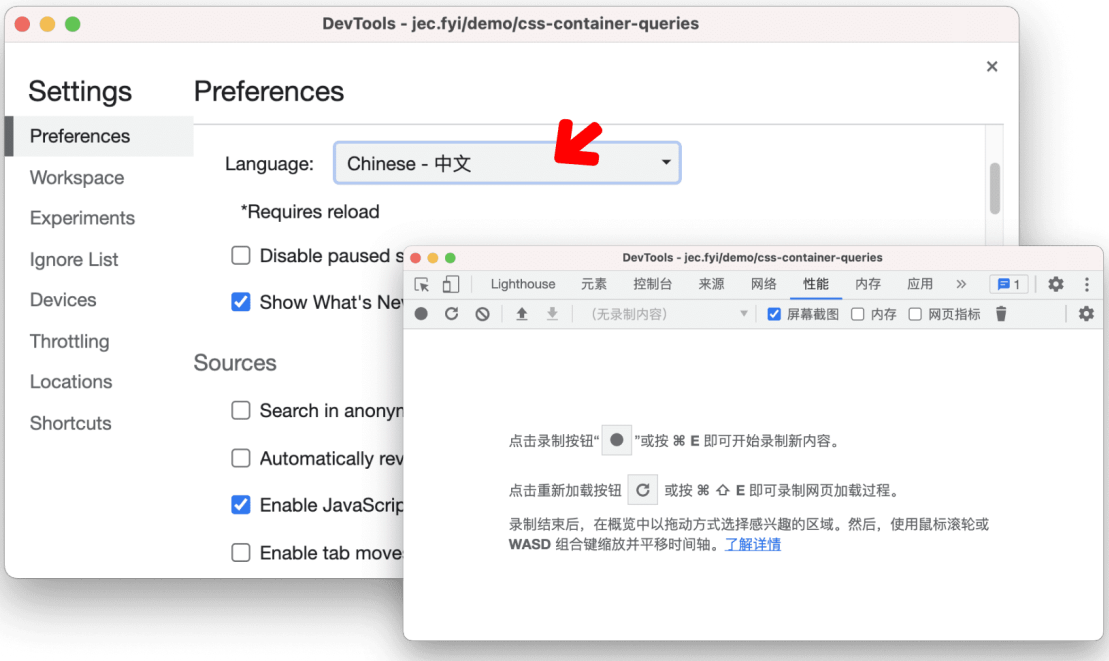
By
default, DevTools will auto-rearrange your panel layout depending on window size. You can disable the auto-rearrangement. Go to settings **Settings > Preferences > Appearance** and update the **panel layout** based on your preference.

For example, the **Styles pane** in the **Elements** panel will move from the side to the bottom when screen size is small. If you want the **Style pane** to always stay on the side, change the **panel layout** to **vertical**.



Change DevTools UI language

See settings **Settings > Preferences > Appearance** > **Language**.



Panels in Dev Tools

1. **Elements Panel**:
   - **Purpose**: View and edit HTML and CSS of a page.
   - **Use**: Change styles, inspect elements, and modify the page's structure.
2. **Console Panel**:
   - **Purpose**: View log messages, errors, and run JavaScript.
   - **Use**: Debug code, check logs, and test JavaScript code interactively.
3. **Network Panel**:
   - **Purpose**: Monitor all network requests (e.g., images, scripts).
   - **Use**: Check how resources are loading and troubleshoot slow performance.
4. **Sources Panel**:
   - **Purpose**: Inspect and debug JavaScript code.

- **Use**: Set breakpoints and step through code to find bugs.

5. **Performance Panel**:
   - **Purpose**: Analyze website performance.
   - **Use**: Track CPU, memory usage, and rendering performance.
6. **Application Panel**:
   - **Purpose**: Manage storage, service workers, and web app features.
   - **Use**: View cookies, local storage, and cache data.
7. **Security Panel**:
   - **Purpose**: Check the security of the page (e.g., HTTPS status).
   - **Use**: Ensure the page is secure and has a valid SSL certificate.
8. **Lighthouse Panel**:
   - **Purpose**: Run audits for performance, accessibility, and SEO.
   - **Use**: Get suggestions to improve the website's quality.
9. **Memory Panel**:
   - **Purpose**: Track memory usage and identify leaks.
   - **Use**: Optimize memory and prevent issues like memory leaks.
10. **Accessibility Panel**:

- **Purpose**: Check accessibility features (e.g., for screen readers).
- **Use**: Ensure the page is usable for people with disabilities.

Elements Panel

1. Inspecting the Webpage Structure

- **HTML Viewer**:
  - On the left side, see the webpage’s structure (HTML).
  - Hover over elements in the HTML tree to highlight them on the page.

- **CSS Viewer**:
  - On the right side, see the styles (CSS) applied to the selected element.

---

2. Editing HTML

- **Modify Text**:
  - Right-click an element → Select "Edit as HTML" → Change the text.
- **Add or Remove Elements**:
  - Right-click → Add or delete elements to change the structure.

---

3. Editing CSS

- **Change Styles**:
  - Modify styles like `color`, `font-size`, or `background` in the **Styles** section on the right.
- **Toggle Styles**:
  - Use checkboxes to turn CSS rules on/off and see the effect instantly.
- **Add New Rules**:
  - Add new properties by clicking on empty lines in the **Styles** section.

---

4. Box Model (Layout)

- **What It Shows**:
  - The **Box Model** displays how margins, borders, padding, and content are spaced for the selected element.
- **How to Change**:
  - Adjust values directly in the Box Model to see real-time changes.

---

5. Real-Time Debugging

- **Experiment with Design**:

    - Show how changing styles (like font size or color) updates the webpage instantly.
- **Fix Issues**:
    - Example: If text is hard to read, increase the font size or change the color live.

---

6. Hands-On Practice

- Inspect a sample webpage.
- Try:
    - Changing the text of a heading.
    - Modifying the background color.
    - Adjusting the padding and margin of an element.

Quick Tips & Tricks

- Editing CSS
- Adding CSS Rules
- Editing HTML
- Scroll into View
- Console Shortcuts
- Hide and Show Elements
- Simulate States
- Computed Styles
- HTML Breakpoints
- Find Event Listeners
- Changing Color Formats
- Import Custom Theme

- CSS Specificity
- Accessible Color Picker
- Accessibility Panel

Exercise 1 - Quick Edits

1. Change this list from an ordered list `ol` to an unordered list `ul`.
2. Can you find the JavaScript code that calls the `alert`?

There is a button below. When clicked, it will reveal a `blockquote`. The `blockquote` has the following HTML:

```
<blockquote class="quote" id="quote" style="background-color:
blue;">
  Did you get it right?
</blockquote>
```

And the following CSS:

```
blockquote {
  background-color: red;
}

.quote {
  background-color: orange;
}

#quote {
  background-color: yellow !important;
}
```

1. Can you guess what background color it will have?
2. Can you "cheat" using the Dev Tools to be 100% sure before you click?

There are a bunch of cards! Can you figure out the border color of the one with id **12345678**? Can you do it in a way that would work if there were 1000 cards on the page?

Apple

Bird

Car

Dove

Elephant

Fire

Goat

Hello

Igloo

Jon

Kuperman 😂

Lion

Mars

Nope

Console Panel

Basic Console Commands:

a. Log a Message:

- The most basic thing you can do in the Console is log messages using `console.log()`.
- Type the following in the Console and press **Enter**:

```
console.log('Hello, Console!');
```

- This will print the message "Hello, Console!" to the Console. This is helpful for debugging and tracking code execution.

b. Logging Different Data Types:

- You can log different data types, such as strings, numbers, objects, and arrays.
- Try the following in the Console:

```
console.log(123);  // Logs a number
console.log({ name: 'Michal', age: 20 });  // Logs an obje
ct
console.log([1, 2, 3, 4]);  // Logs an array
```

c. Logging with Multiple Arguments:

- You can also log multiple values at once. Type the following in the Console:

```
console.log('The result is:', 42);
console.log('User:', { name: 'Michal' }, 'Age:', 20);
```

- This will print a string followed by the number and object/variables in a more readable format.

Using `console.error()`, `console.warn()`, and `console.info()`:

- `console.error()`: Logs errors with a red color.
  - Try typing:

```
console.error('This is an error message.');
```

- `console.warn()`: Logs warnings with a yellow color.
  - Try typing:

```
console.warn('This is a warning message.');
```

- `console.info()`: Logs information with a blue color.
  - Try typing:

```
console.info('This is an informational message.');
```

Using `console.table()` to Log Data in a Table:

The `console.table()` function is useful for logging arrays or objects in a table format. Try this:

```
javascript
Copy code
let users = [
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 30 },
  { name: 'Charlie', age: 35 }
];
console.table(users);
```

- This will display the `users` array in a table format, making it easier to view the properties of each object.

Clear the Console:

Sometimes, you might want to clear the Console to get rid of all the logs. You can do this by typing:

```
console.clear();
```

- This clears the console window.

Exercise - 2

DOM Interaction

In this HTML structure, we have:

1. A heading `<h1>` with the ID `header`.

2. A button with the class `button`.
3. A container `<div>` with the class `container` that will hold some dynamic content.

Now, let's write JavaScript code that interacts with this page via the **Console**.

Interacting with the DOM

1. **Selecting Elements**:
   - `let heading = document.getElementById('header');` selects the header element using its ID.
   - `let clickButton = document.querySelector('.button');` selects the first element with the class `button`.
2. **Manipulating Text Content**:
   - `header.textContent = 'New Header Text';` changes the text content of the `header` element to the new value.
3. **Changing Styles**:
   - We modify the button's background color and font size using inline styles:

```
button.style.backgroundColor = 'lightgreen';
button.style.fontSize = '18px';
```

4. **Adding New Elements**:
   - We create a new paragraph using `document.createElement('p')`, add text to it with `textContent`, and then append it to an existing container:

```
let container = document.querySelector('.container');
let newParagraph = document.createElement('p');
newParagraph.textContent = 'New paragraph added after b
utton click!';
container.appendChild(newParagraph);
```

5. **Using `setTimeout` to Add Elements Dynamically**:

- We use `setTimeout` to simulate delayed execution, adding a paragraph after 3 seconds.

Source Panel

1. Source Panel

The **Source Panel** is where you can view and debug the code running on a webpage.

- **View Files**: It shows all the files (like JavaScript, CSS, etc.) used by the webpage.
- **Set Breakpoints**: You can click on line numbers in the code to pause the script and inspect what's happening.
- **Step Through Code**: Once paused at a breakpoint, you can move through the code line-by-line to see how it works.
- **Watch Variables**: You can track the values of specific variables as you step through the code.

2. Workspace

The **Workspace** feature lets you **edit** your local project files directly in the browser.

- **Link Local Files**: You can link your project files (on your computer) to the browser's DevTools.
- **Edit Files in DevTools**: After linking, you can edit JavaScript, CSS, or HTML directly in the browser.
- **Save Changes**: Changes you make in DevTools can be saved to your actual files on your computer.

How It Works Together:

- **Source Panel** helps you **debug** your code (pause, inspect, step through).

- **Workspace** allows you to **edit** and **save** your code live in the browser, speeding up development.

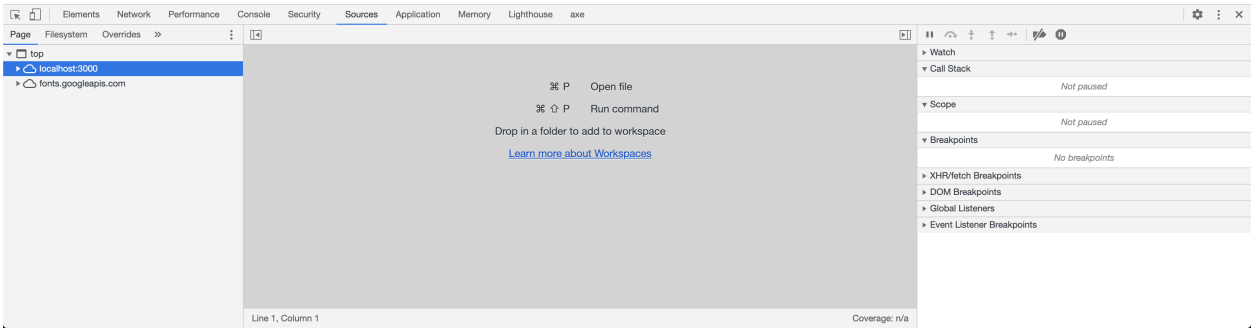
Example Workflow:

1. **Debug in the Source Panel**: Set breakpoints in your code, step through it, and watch variables change.
2. **Edit in the Workspace**: Modify your code directly in the browser, then save the changes to your local files.

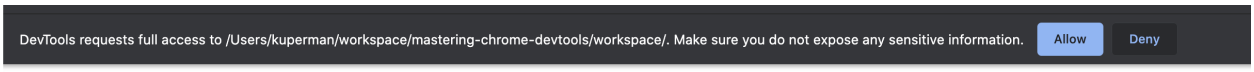
Exercise - 3

Instructions

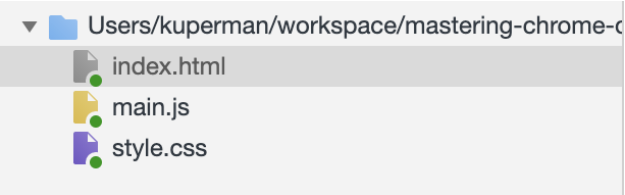
1. Open the DevTools
2. Click on the Sources panel
3. Open Windows Explorer / Finder and navigate to this project
4. Grab the `workspace` folder and drag it into the workspace section.



6. Click `allow` at the top of your browser window.



7. You should now see little green circles next to all of the filenames in the sidebar.



8. Then complete the three tasks outlined in the Workspace!

▼ Nothing happens

First, we need to check the console for any logged output. Then, we'll examine the JavaScript in the source panel.

Inside the `form.addEventListener()` function:

1. Remove the line `console.log(form.value);`
2. Add the line `items.push(form.value);`

Debugging

For demonstration purposes, we're just logging in the background with an interval. Click on the `Debugging.js` file.

If you click on a function, it will turn blue, indicating that a Breakpoints is set. Once the Breakpoints is created, you won't see any more console logs here.

- Set and activate breakpoints
- Use the `debugger` keyword
- Walk the call stack
- Set conditional breakpoints
- Set XHR breakpoints

Exercise - 4

Debugging

The area below is supposed to be a list of interesting facts about cats! If you look at the HTML you'll see the following:

```
<ul id="catfacts"></ul>
```

And a file in this repository `/exercises/js/Debugging.js` is hitting an API and supposed to populate that list! Can you figure out what's going wrong?

Hint: Checking the console is a great place to start. Once you've identified the area, use the debugger in the Sources tab to get a lay of the land.

Here is a list of FACTS about CATS

▼ May be...

1. Open the Debugger.js file
2. Inspect the DevTools interface
3. Go to the console and fix the error:

   `const li = document.createElement("li");`
4. Navigate to Sources and resolve the undefined value
5. Set breakpoints at:

   `li.innerText = inner.fact`
6. Open the Watch panel and enter "li"

7. Add "item" to the watch list

Network Performance

Open the Network panel

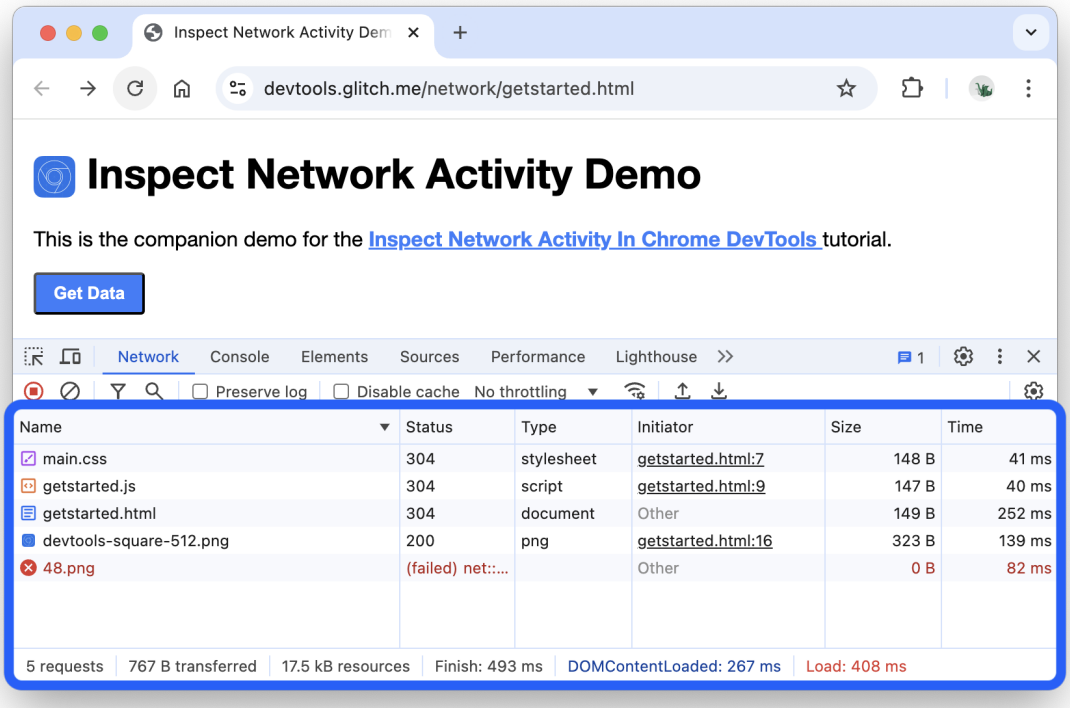
To get the most out of this tutorial, open up the demo and try out the features on the demo page.

1. Open the Get Started Demo.

Log network activity

To view the network activity that a page causes:

1. Reload the page. The **Network** panel logs all network activity in the **Network Log**.



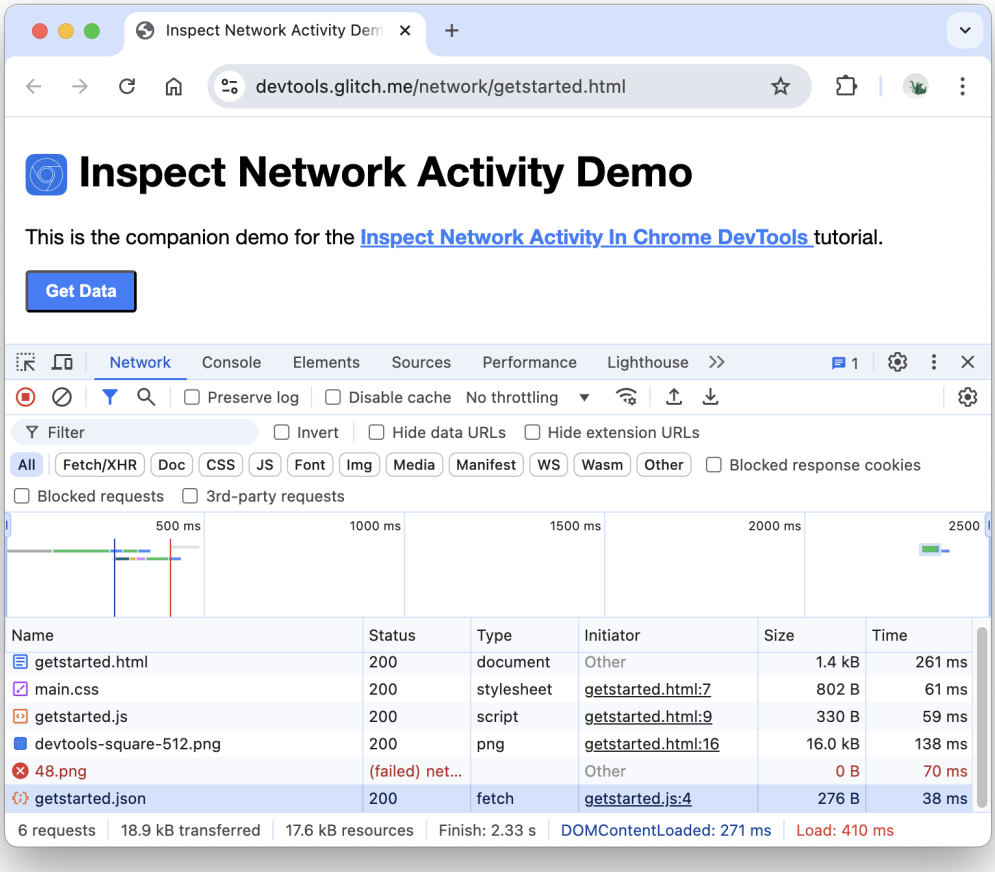
Each row of the **Network Log** represents a resource. By default the resources are listed
chronologically. The top resource is usually the main HTML document. The bottom resource is
whatever was requested last.

Each column represents information about a resource. The default columns are:

- **Status**: The HTTP response code.
- **Type**: The resource type.
- **Initiator**: What caused a resource to be requested. Clicking a link in the Initiator column
  takes you to the source code that caused the request.
- **Size**: Resource amount transferred over the network.
- **Time**: How long the request took.

1. So long as you've got DevTools open, it will record network activity in the **Network Log**. To
   demonstrate this, first look at the bottom of the
   Network Log and take note of the
   last activity.
2. Now, click the **Get Data** button in the demo.
3. Look at the bottom of the **Network Log** again. There's a new resource called

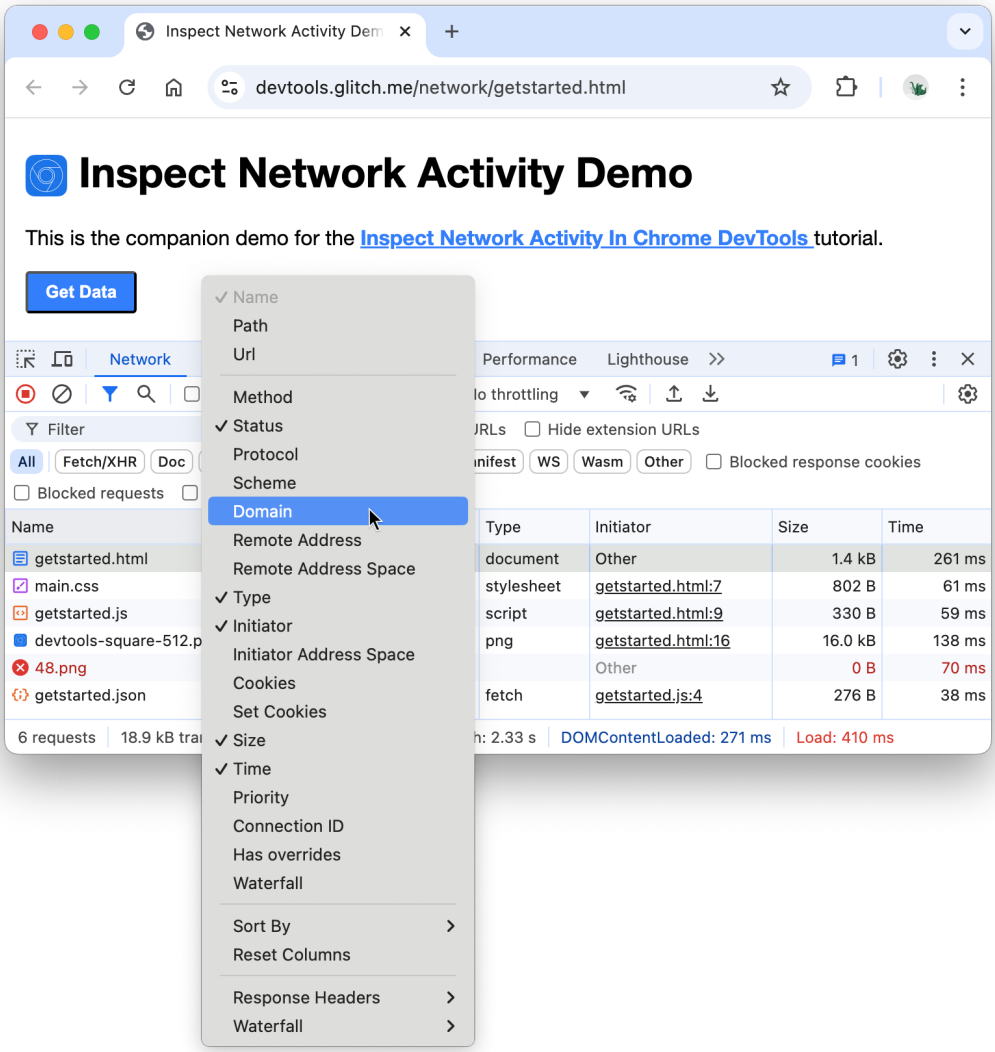
   `getstarted.json`. Clicking the **Get Data** button caused the page to request this file.



Show more information

The columns of the **Network Log** are configurable. You can hide columns that you're not using. There
are also many columns that are hidden by default which you may find useful.

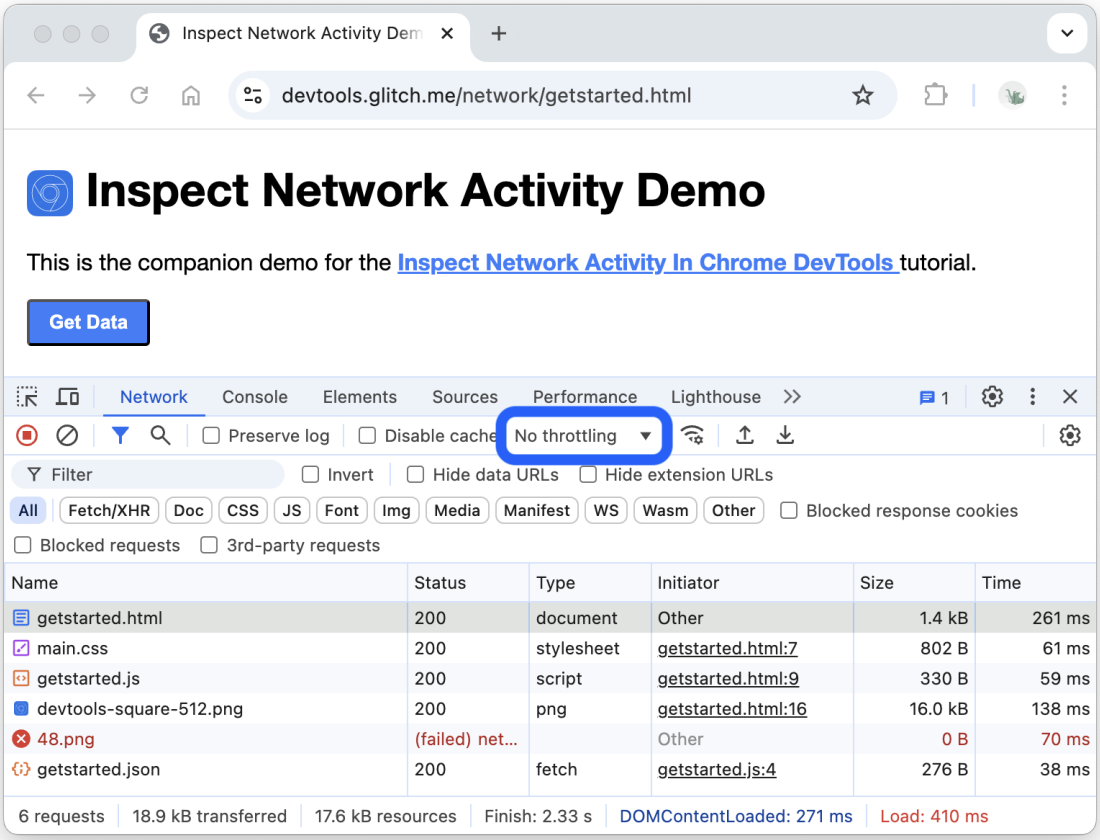
1. Right-click the header of the **Network Log** table and select **Domain**. The domain of each
   resource is now shown.



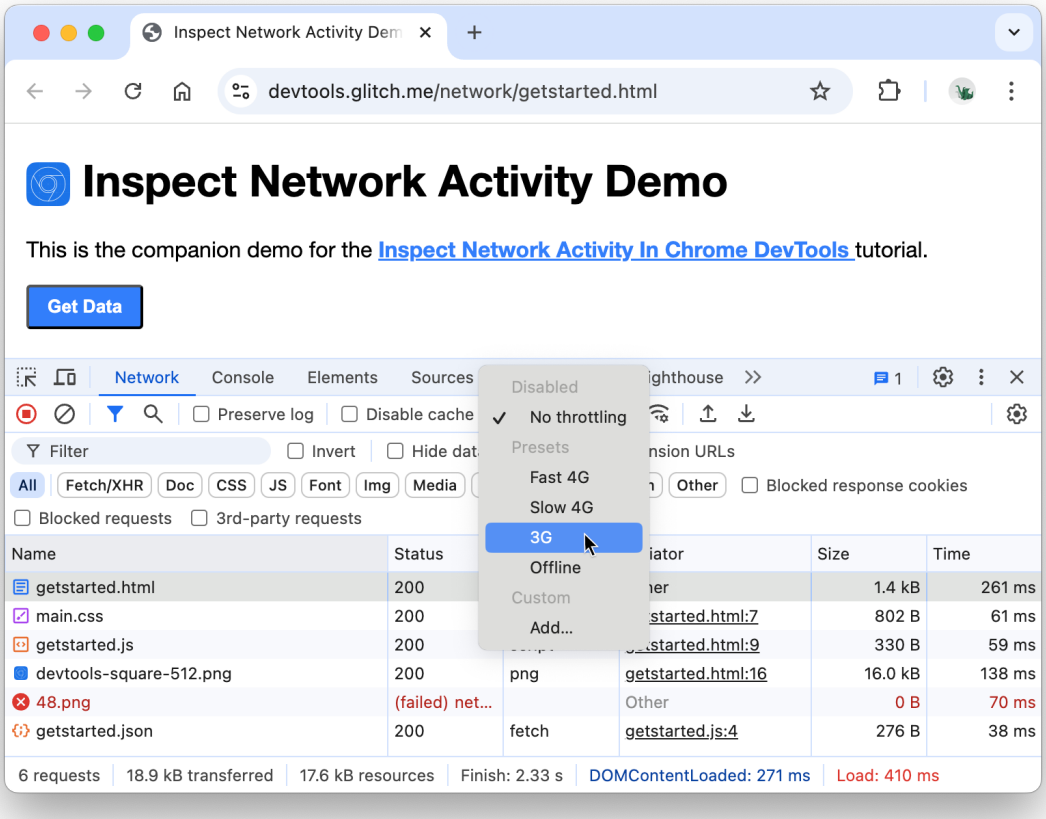
Simulate a slower network connection

The network connection of the computer that you use to build sites is probably faster than the
network connections of the mobile devices of your users. By throttling the page you can get a better
idea of how long a page takes to load on a mobile device.

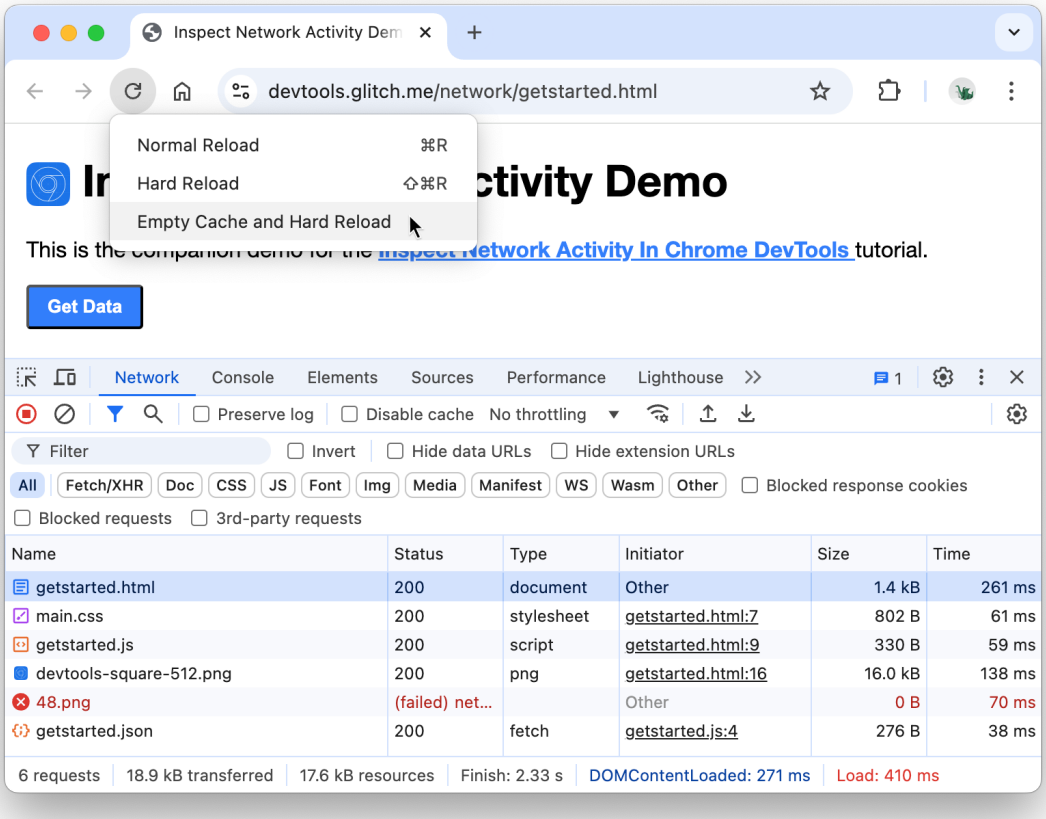
1. Click the **Throttling** drop-down, which is set to **No throttling** by default.


Inspect Network Activity Dem
devtools.glitch.me/network/getstarted.html
Inspect Network Activity Demo
This is the companion demo for the Inspect Network Activity In Chrome DevTools tutorial.
Get Data
Network
Console
Elements
Sources
Performance
Lighthouse
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
Hide extension URLs
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
WS
Wasm
Other
Blocked response cookies
Blocked requests
3rd-party requests
Name
Status
Type
Initiator
Size
Time
getstarted.html 200 document Other 1.4 kB 261 ms
main.css 200 stylesheet getstarted.html:7 802 B 61 ms
getstarted.js 200 script getstarted.html:9 330 B 59 ms
devtools-square-512.png 200 png getstarted.html:16 16.0 kB 138 ms
48.png (failed) net... Other 0 B 70 ms
getstarted.json 200 fetch getstarted.js:4 276 B 38 ms
6 requests
18.9 kB transferred
17.6 kB resources
Finish: 2.33 s
DOMContentLoaded: 271 ms
Load: 410 ms

2. Select **3G**.



3. Long-press **Reload** refresh and then select **Empty Cache And Hard Reload**.



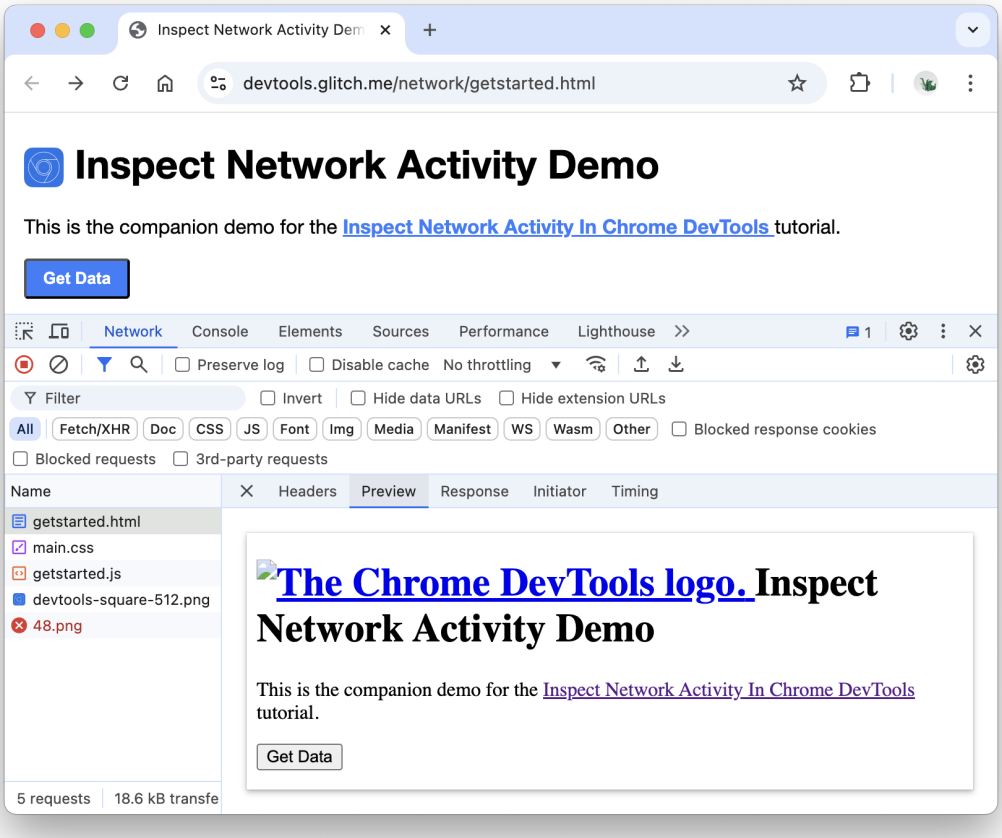
On repeat visits, the browser usually serves some files from its cache, which speeds up the
page load.
Empty Cache And Hard Reload forces the browser to go the network for all resources. This is helpful when you want to see how a first-time visitor experiences a page
load.

Capture screenshots

- Click **Network Settings** settings.
- Enable the **Screenshots** check_box checkbox.
- Reload the page again using the **Empty Cache And Hard Reload** workflow.

Inspect a resource's details

1. Click `getstarted.html`. The **Headers** tab is shown. Use this tab to inspect HTTP headers.
2. Click the **Preview** tab to view a basic HTML rendering.



This tab is helpful when an API returns an error code in HTML and it's easier to read the
rendered HTML than the HTML source code, or when inspecting images.

3. Click the **Response** tab to view the HTML source code.
4. Click the **Initiator** tab to view a tree that maps the request initiator chain.
5. Click the **Timing** tab to view a breakdown of the network activity for this resource.

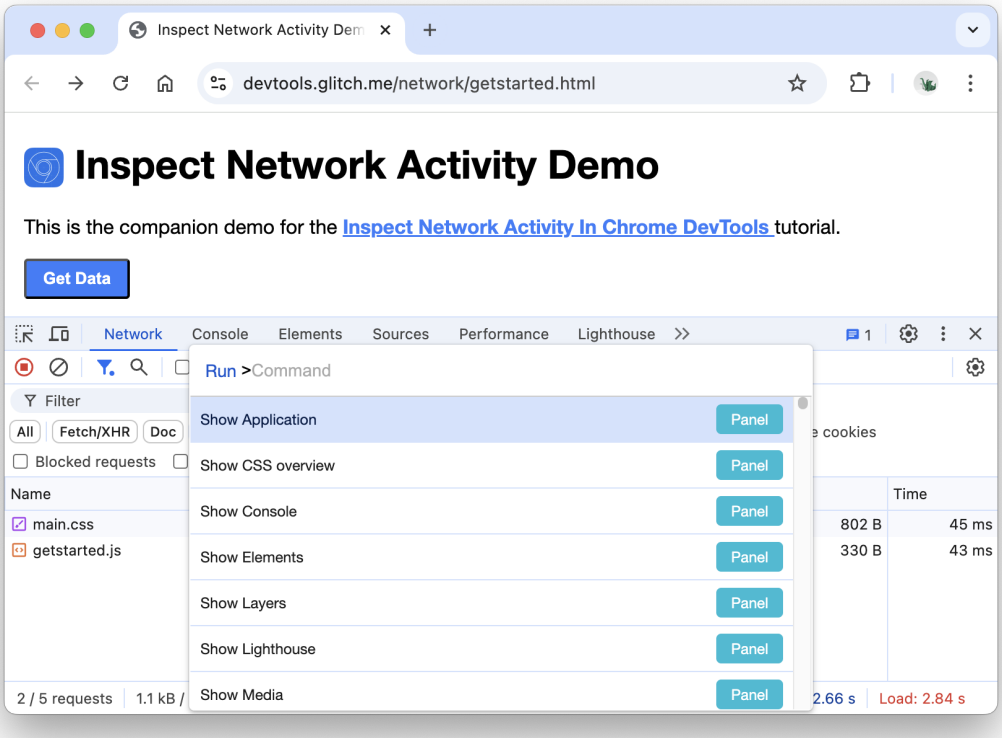
6. Click **Close** X to view the **Network Log** again.

Block requests

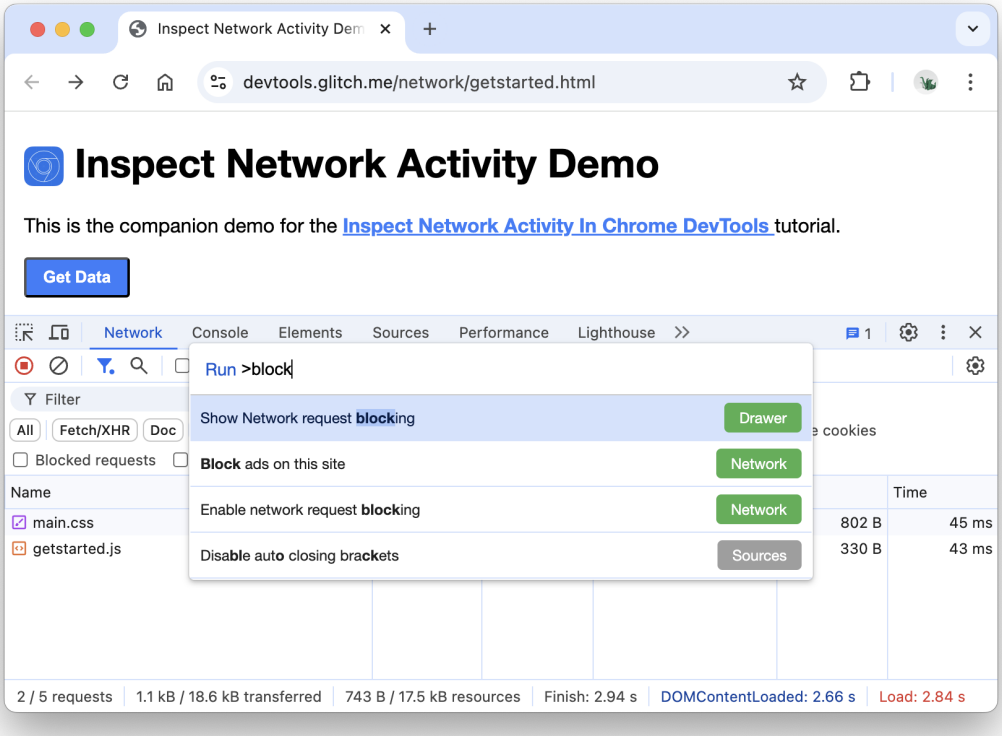
Block requests

How does a page look and behave when some of its resources aren't available? Does it fail
completely, or is it still somewhat functional? Block requests to find out:

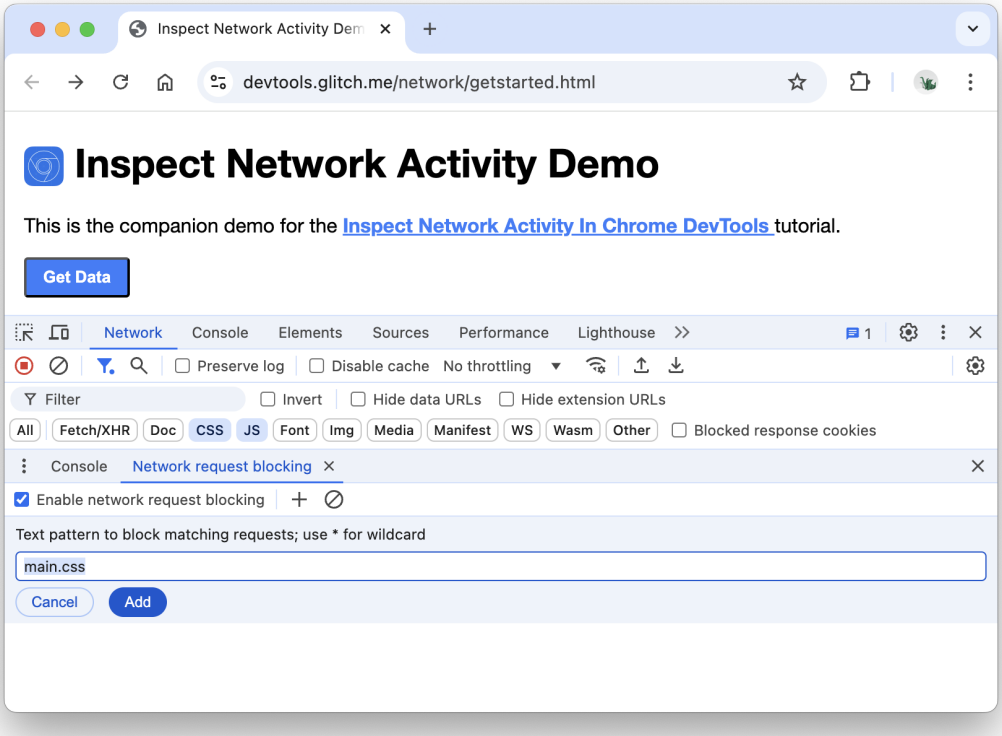
1. Press Control+Shift+P or Command+Shift+P (Mac) to open the **Command Menu**.



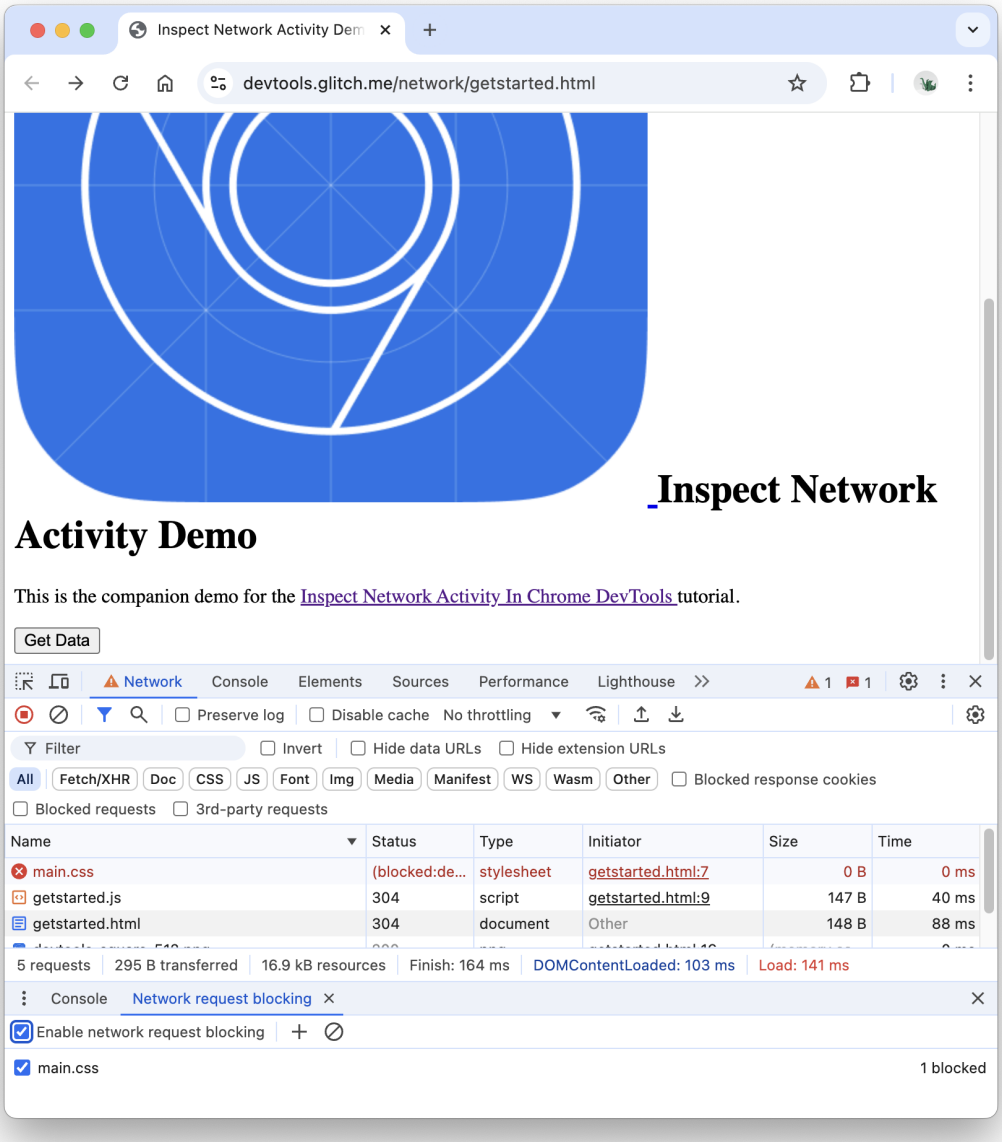
2. Type `block`, select **Show Request Blocking**, and press Enter.



3. Click the **Add Pattern** button.
4. Type `main.css`.



5. Click **Add**.
6. Reload the page. As expected, the page's styling is slightly messed up because its main
   style sheet has been blocked. Note the
   `main.css` row in the Network Log. The red text means that
   the resource is blocked.



7. Clear the **Enable request blocking** checkbox.

Record network requests

By default, DevTools records all network requests in the **Network** panel, so long as DevTools is open.

Stop recording network requests

To stop recording requests:

- Click **Stop recording network log** on the **Network** panel. The icon turns grey to indicate that DevTools is no longer recording requests.

- Press Command + E (Mac) or Control + E (Windows, Linux) while the **Network** panel is in focus.

Clear requests

Click **Clear**

on the **Network** panel to clear all requests from the **Requests** table.

Save requests across page loads

To save requests across page loads, check the **Preserve log** checkbox on the **Network** panel. DevTools saves all requests until you disable **Preserve log**.

The Network Waterfall

A visual breakdown of each request's activity.

Let's talk about the life of a network request.

- **Queueing**
  - There are higher priority requests.
  - There are already six TCP connections open for this origin, which is the limit. Applies to HTTP/1.0 and HTTP/1.1 only.
  - The browser is briefly allocating space in the disk cache
- **Stalled**
  - The request is stuck Queueing

- 🟩 **DNS Lookup**
  - Resolving an IP address
- 🟧 **Initial Connection**
  - TCP handshakes or establishing SSL
- 🟧 **Service Worker Startup**
  - Starting up a Service Worker
- 🟪 **Service Worker respondsWith**
  - Service Worker sending data to browser
- 🟩 **Waiting (TTFB)**
  - Waiting for the first byte from the server
- 🟦 **Content Download**
  - Browser is receiving a response from a server