What is IPFS?

**IPFS is a distributed system for storing and accessing files, websites, applications, and data.**

What does that mean, exactly? Let's say you're doing some research on aardvarks. You might start by visiting the Wikipedia page on aardvarks at:

https://en.wikipedia.org/wiki/Aardvark

When you put that URL in your browser's address bar, your computer asks one of Wikipedia's computers, which might be somewhere on the other side of the country (or even the planet), for the aardvark page.

However, that's not the only option for meeting your aardvark needs! There's a mirror of Wikipedia stored on IPFS, and you could use that instead. If you use IPFS, your computer asks to get the aardvark page like this:

/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Aardvark.html

**TIP**

The easiest way to view the above link is by opening it in your browser through an *IPFS Gateway*. Simply add https://ipfs.io to the start of the above link and you'll be able to [view the page →(opens new window)](https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Aardvark.html)

IPFS knows how to find that sweet, sweet aardvark information by its [contents](https://docs.ipfs.io/concepts/content-addressing/), not its location (more on that, which is called content addressing, below). The IPFS-ified version of the aardvark info is represented by that string of numbers in the middle of the URL (QmXo…), and instead of asking one of Wikipedia's computers for the page, your computer uses IPFS to ask lots of computers around the world to share the page with you. It can get your aardvark info from anyone who has it, not just Wikipedia.

And, when you use IPFS, you don't just download files from someone else — your computer also helps distribute them. When your friend a few blocks away needs the same Wikipedia page, they might be as likely to get it from you as they would from your neighbor or anyone else using IPFS.

IPFS makes this possible for not only web pages but also any kind of file a computer might store, whether it's a document, an email, or even a database record.

**Decentralization**

Making it possible to download a file from many locations that aren't managed by one organization:

* **Supports a resilient internet.** If someone attacks Wikipedia's web servers or an engineer at Wikipedia makes a big mistake that causes their servers to catch fire, you can still get the same webpages from somewhere else.
* **Makes it harder to censor content.** Because files on IPFS can come from many places, it's harder for anyone (whether they're states, corporations, or someone else) to block things. We hope IPFS can help provide ways to circumvent actions like these when they happen.
* **Can speed up the web when you're far away or disconnected.** If you can retrieve a file from someone nearby instead of hundreds or thousands of miles away, you can often get it faster. This is especially valuable if your community is networked locally but doesn't have a good connection to the wider internet. (Well-funded organizations with technical expertise do this today by using multiple data centers or CDNs — [content distribution networks (opens new window)](https://en.wikipedia.org/wiki/Content_delivery_network). IPFS hopes to make this possible for everyone.)

That last point is actually where IPFS gets its full name: the **InterPlanetary File System**. We're striving to build a system that works across places as disconnected or as far apart as planets. While that's an idealistic goal, it keeps us working and thinking hard, and almost everything we create in pursuit of that goal is also useful here at home.

**Content addressing**

For a beginner-friendly primer on why cryptographic hashing and content addressing matter, take a look at ProtoSchool's tutorial, [Content Addressing on the Decentralized Web (opens new window)](https://proto.school/content-addressing).

What about that link to the aardvark page above? It looked a little unusual:

/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Aardvark.html

That jumble of letters after /ipfs/ is called a [*content identifier*](https://docs.ipfs.io/concepts/content-addressing/) and it’s how IPFS can get content from multiple places.

Traditional URLs and file paths such as…

* https://en.wikipedia.org/wiki/Aardvark
* /Users/Alice/Documents/term\_paper.doc
* C:\Users\Joe\My Documents\project\_sprint\_presentation.ppt

…identify a file by *where it's located* — what computer it's on and where on that computer's hard drive it is. That doesn't work if the file is in many places, though, like your neighbor's computer and your friend's across town.

Instead of being location-based, IPFS addresses a file by *what's in it*, or by its *content*. The content identifier above is a *cryptographic hash* of the content at that address. The hash is unique to the content that it came from, even though it may look short compared to the original content. It also allows you to verify that you got what you asked for — bad actors can't just hand you content that doesn't match. (If hashes are new to you, check out [the concept guide on hashes](https://docs.ipfs.io/concepts/hashing/) for an introduction.)

**NOTE**

Why do we say "content" instead of "files" or "web pages" here? Because a content identifier can point to many different types of data, such as a single small file, a piece of a larger file, or metadata. (In case you don't know, metadata is "data about the data." You use metadata when you access the date, location, or file size of your digital pictures, for example.) So, an individual IPFS address can refer to the metadata of just a single piece of a file, a whole file, a directory, a whole website, or any other kind of content. For more on this, check out our guide to [how IPFS works](https://docs.ipfs.io/concepts/how-ipfs-works/).

Because the address of a file in IPFS is created from the content itself, links in IPFS can't be changed. For example ...

* If the text on a web page is changed, the new version gets a new, different address.
* Content can't be moved to a different address. On today's internet, a company could reorganize content on their website and move a page at http://mycompany.com/what\_we\_do to http://mycompany.com/services. In IPFS, the old link you have would still point to the same old content.

Of course, people want to update and change content all the time and don't want to send new links every time they do it. This is entirely possible in an IPFS world, but explaining it requires a little more info than what's within the scope of this IPFS introduction. Check out the concept guides on [IPNS](https://docs.ipfs.io/concepts/ipns/), the [Mutable File System (MFS)](https://docs.ipfs.io/concepts/file-systems/#mutable-file-system-mfs), and [DNSLink](https://docs.ipfs.io/concepts/dnslink/) to learn more about how changing content can work in a content-addressed, distributed system.

It's important to remember in all of these situations, using IPFS is participatory and collaborative. If nobody using IPFS has the content identified by a given address available for others to access, you won't be able to get it. On the other hand, content can't be removed from IPFS as long as *someone* is interested enough to make it available, whether that person is the original author or not. Note that this is similar to the current web, where it is also impossible to remove content that's been copied across an unknowable number of websites; the difference with IPFS is that you are always able to find those copies.

**Participation**

While there's lots of complex technology in IPFS, the fundamental ideas are about changing how networks of people and computers communicate. Today's World Wide Web is structured on *ownership* and *access*, meaning that you get files from whoever owns them — if they choose to grant you access. IPFS is based on the ideas of *possession* and *participation*, where many people *possess* each others' files and *participate* in making them available.

That means IPFS only works well when people are actively participating. If you use your computer to share files using IPFS, but then you turn your computer off, other people won't be able to get those files from you anymore. But if you or others make sure that copies of those files are stored on more than one computer that's powered on and running IPFS, those files will be more reliably available to other IPFS users who want them. This happens to some extent automatically: by default, your computer shares a file with others for a limited time after you've downloaded it using IPFS. You can also make content available more permanently by *pinning* it, which saves it to your computer and makes it available on the IPFS network until you decide to *unpin* it. (You can learn more about this in our [guide to persistence and pinning](https://docs.ipfs.io/concepts/persistence/).)

If you want to make sure one of your own files is permanently shared on the internet today, you might use a for-pay file-sharing service like Dropbox. Some people have begun offering similar services based on IPFS called *pinning services*. But since IPFS makes this kind of sharing a built-in feature, you can also collaborate with friends or partner with institutions (for example, museums and libraries might work together) to share each others' files. We hope IPFS can be the low-level tool that allows a rich fabric of communities, business, and cooperative organizations to all form a distributed web that is much more reliable, robust, and equitable than the one we have today.

Why IPFS?

Explain the original web model and the limitation

\* Content addressing instead of location addressing

\* decentralized content distributed among peers Content

Content is hashed as CID

\* Content is immutable each update generates new CID

\* Content addressing Routing

Distributed Hash Table (DHTs) maps CID / Peer IP address

\* DHT server hosts content and DHT Publishing Content

New Content that you want to share on ipfs

\* hash the content creating new CID

\* Update your local DHT CID / your ip address

\* DHT will be updated to all the content peer (NOT the CONTENT)

\* People searching for your CID will be connected to you and only you. Consuming Content

ipfs client (dht client) want to consume Ipfs://cid/

\* ipfs client consults its local DHT table to see where this CID is located, gets back a collection of IP addresses

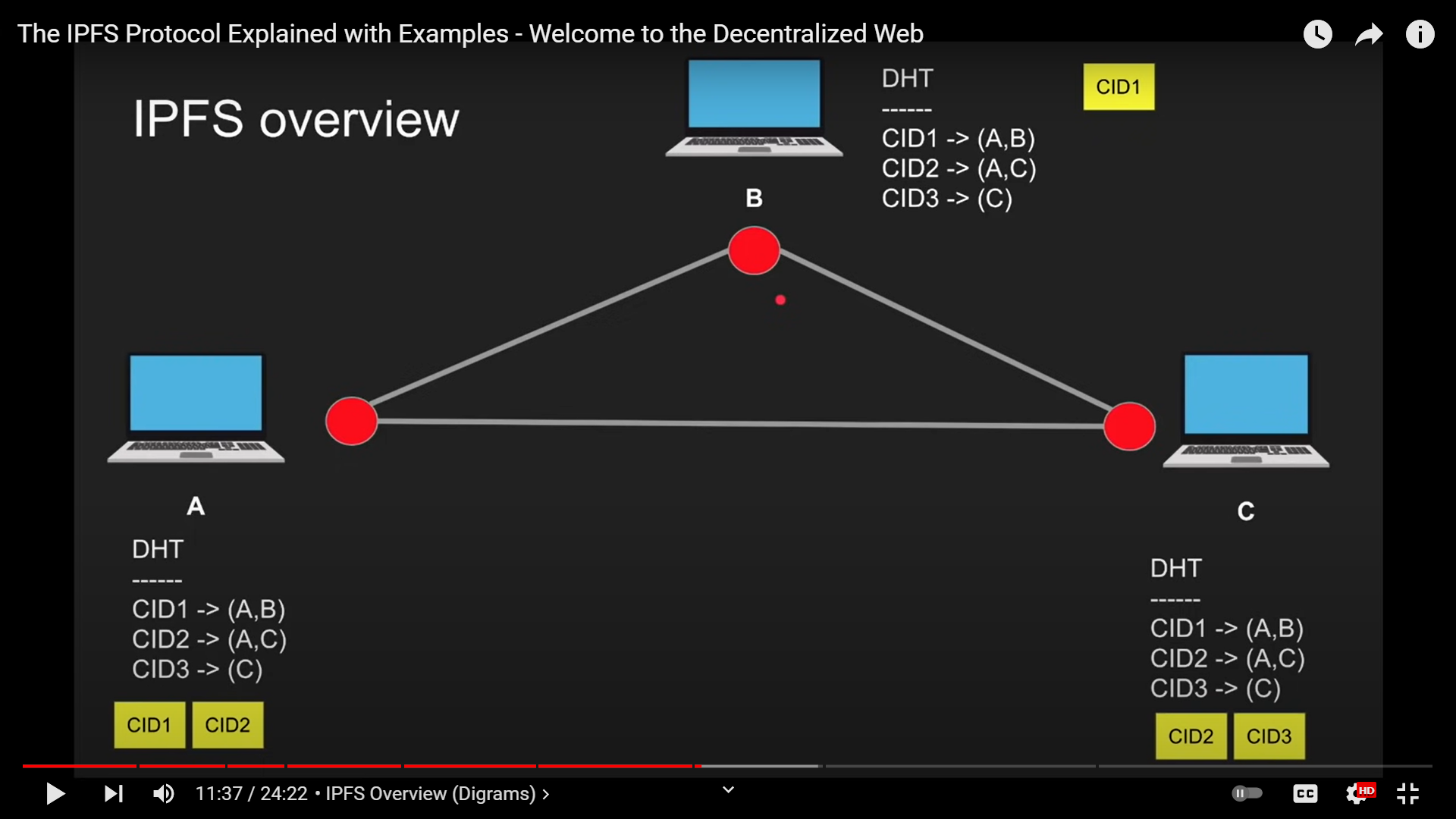
\* client connects to some or all the peers found hosting that CID

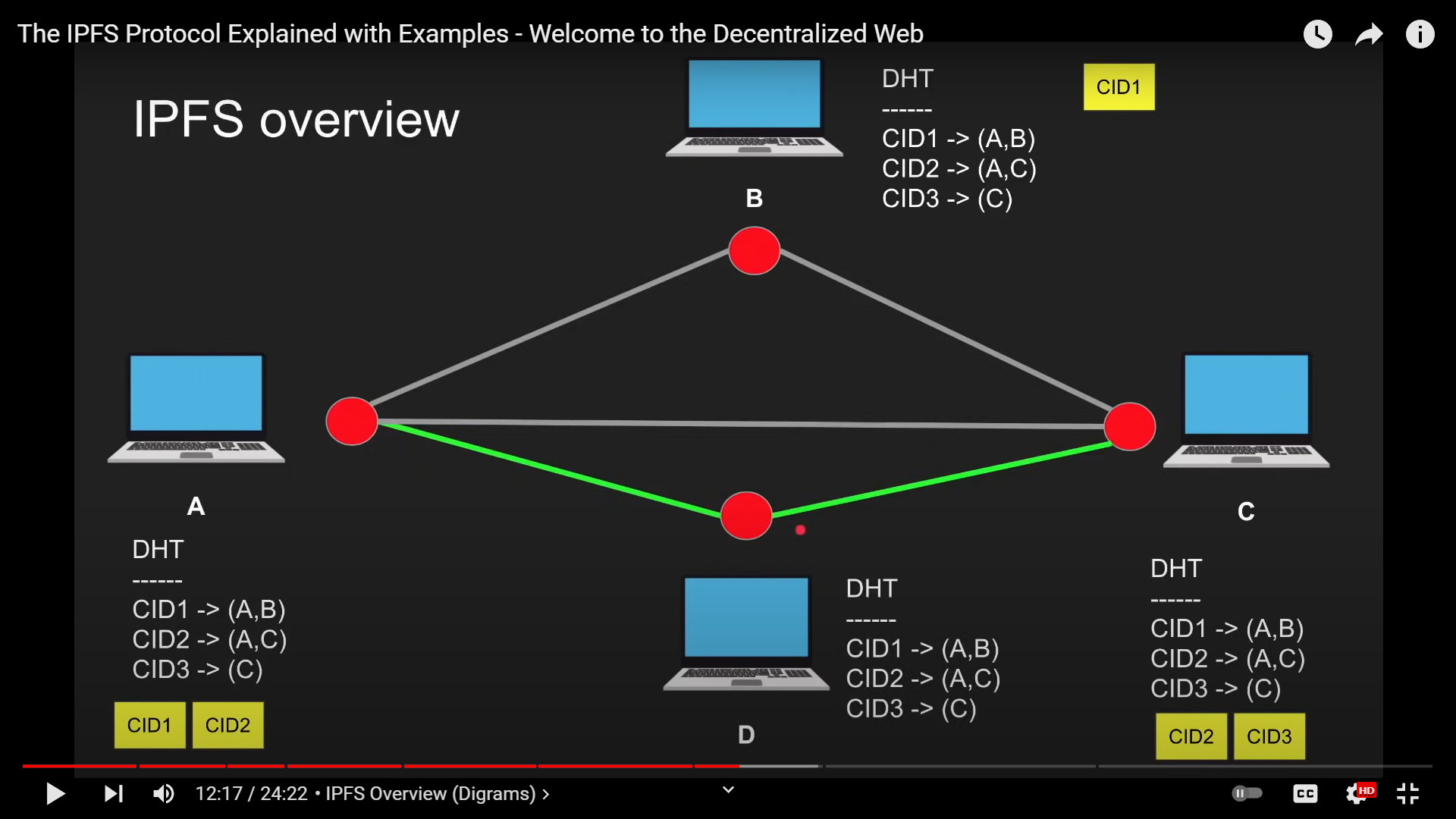
\* client downloads chunks of the content from each peer so it speeds up

\* Once the client has the content it is now also updating its local DHT table that it now also hosts that CID (if it supports being a DHT server)

\* New updated DHT is propogated across peer

IPFS Overview





# **Intro to Web3 Storage, the easiest way to use IPFS(hello text file)**

**n the past, storing data on the decentralized web wasn't always easy — but that's where Web3.Storage comes in.** Most data on the internet today is hosted by massive storage providers like Amazon, Google, and Microsoft. This makes it simpler for developers to store application data, but big corporate platforms like these create silos by locking developers and users alike into walled gardens of services. What's more, as the market continues to consolidate, a small oligopoly of providers are essentially becoming the storage backbone of the internet.

One solution to this problem is using *decentralized storage* instead of big, corporate platforms to store data for apps and services. However, decentralized storage can be difficult to manage and add extra time and effort to an already crowded developer workflow — for example, most decentralized storage services need you to compile your data into a specific format, find a storage provider to host your data, buy some cryptocurrency to pay the storage provider, and then send your data across the internet. **This is where Web3.Storage comes in.**

With Web3.Storage, you get all the benefits of decentralized storage technologies with the frictionless experience you expect in a modern dev workflow. **All you need to use Web3.Storage is an API token and your data.**

## Quickstart[​](https://web3.storage/docs/" \l "quickstart" \o "Direct link to heading)

**Ready to get started using Web3.Storage right now?** Get up and running in minutes by following this quickstart guide. In this guide, we'll walk through the following steps:

1. [**Creating a Web3.Storage account.**](https://web3.storage/docs/#create-an-account)
2. [**Getting a free API token.**](https://web3.storage/docs/#get-an-api-token)
3. [**Creating and running a simple script**](https://web3.storage/docs/#create-the-upload-script) to upload a file.
4. [**Getting your uploaded file**](https://web3.storage/docs/#get-your-file) using your browser or curl.

**This guide uses Node.js since it's the fastest way to get started using the Web3.Storage JavaScript client programmatically**, but don't worry if Node isn't your favorite runtime environment — or if you'd rather not do any coding at all. You can also use Web3.Storage in the following ways:

* Work with the API methods in the [**JavaScript client library**](https://web3.storage/docs/reference/js-client-library) using the JS runtime of your choice.
* Upload and retrieve files directly from your [**Files page**](https://web3.storage/files/) on the Web3.Storage website.

##### PREREQUISITES

You'll need **Node version 14** or higher and **NPM version 7** or higher to complete this guide. Check your local versions like this:

node --version && npm --version  
> v16.4.2  
> 7.18.1

## Create an account[​](https://web3.storage/docs/#create-an-account)

You need a Web3.Storage account to get your API token and manage your stored data. You can sign up **for free** using your email address or GitHub.

* Email
* GitHub

##### Sign up using email[​](https://web3.storage/docs/#sign-up-using-email)

1. Go to [**web3.storage/login**](https://web3.storage/login) to get started.
2. Enter your email address.
3. Check your inbox for a verification email from Web3.Storage, and click the **Log in** button in the email.
4. You're all set!

Now that you're signed up and logged in, it's time to [**get your API token.**](https://web3.storage/docs/#get-an-api-token)

## Get an API token[​](https://web3.storage/docs/#get-an-api-token)

It only takes a few moments to get a free API token from Web3.Storage. This token enables you to interact with the Web3.Storage service without using the main website, enabling you to incorporate files stored using Web3.Storage directly into your applications and services.

1. Click **Account** to go to your [**Web3.Storage account page**](https://web3.storage/account).
2. Click **Create an API token**.
3. Enter a descriptive name for your API token and click **Create**.
4. Make a note of the **Token** field somewhere secure where you know you won't lose it. You can click  to your new API token to your clipboard.

##### Keep your API token private

Do not share your API token with anyone else. This key is specific to your account.

Now that you have your new API token, it's time to use a simple script to [**upload a file to Web3.Storage.**](https://web3.storage/docs/#create-the-upload-script)

## Create the upload script[​](https://web3.storage/docs/#create-the-upload-script)

You can use the Web3.Storage site to upload files, but it's also quick and easy to create and run a simple upload script — making it especially convenient to add large numbers of files. This script contains logic to upload a file to Web3.Storage and get a [**content identifier (CID)**](https://web3.storage/docs/concepts/content-addressing) back in return.

##### CAUTION

All data uploaded to Web3.Storage is available to anyone who requests it using the correct CID. Do not store any private or sensitive information in an unencrypted form using Web3.Storage.

1. Create a folder for this quickstart project, and move into that folder:

mkdir web3-storage-quickstart  
cd web3-storage-quickstart

1. Create a file called put-files.js and paste in the following code:

*import* process *from* 'process'  
*import* minimist *from* 'minimist'  
*import* { Web3Storage, getFilesFromPath } *from* 'web3.storage'  
  
*async* *function* main () {  
 *const* args = minimist(process.argv.slice(2))  
 *const* token = args.token  
  
 *if* (!token) {  
 *return* console.error('A token is needed. You can create one on https://web3.storage')  
 }  
  
 *if* (args.\_.length < 1) {  
 *return* console.error('Please supply the path to a file or directory')  
 }  
  
 *const* storage = *new* Web3Storage({ token })  
 *const* files = []  
  
 *for* (*const* path *of* args.\_) {  
 *const* pathFiles = *await* getFilesFromPath(path)  
 files.push(...pathFiles)  
 }  
  
 console.log(`Uploading ${files.length} files`)  
 *const* cid = *await* storage.put(files)  
 console.log('Content added with CID:', cid)  
}  
  
main()

CODE EXPLAINATION: -

First we import the process module from node.js this is to access the command line arguments

Then we import library that we are going to install called minimist to easily pass the command line argument into javascript

then we import the java script library web3 storage and here the most important is webster storage this is an object to easily use the api and here this ia a utility function

so we define a function and first with mist library we pass the argument provided to the script and then we extract the api token {5,6,7]}

So here we do some checks{9-16}

After we instantiate our connection to the web storage api by providing the token we also define an array of the file that we’re going to upload to ipfs {17,18}

We’re going to loop through all the files that we provided as argument to our script and after we use the utility function get files from pass so if the pass is already a file its just going to be an array with a single element otherwise if this is a directory its going to give you an array of all the file inside and after we use the spread operator to spread this array and add all element inside to the file array{20,21,22}

and after we are going to store our file into ipfs with this line {26}here storage.put and after in return its going to give us a CID so that the identifier for our data{23,24,25,26}

and we’re going to print this and here we execute our main function {30}

1. Create another file called package.json and paste in the following code:

{  
 "name": "web3-storage-quickstart",  
 "version": "0.0.0",  
 "private": true,  
 "description": "Get started using web3.storage in Node.js",  
 "type": "module",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "dependencies": {  
 "minimist": "^1.2.5",  
 "web3.storage": "^3.1.0"  
 },  
 "author": "YOUR NAME",  
 "license": "(Apache-2.0 AND MIT)"  
}

1. Save both files, and then run npm install from your project folder:

npm install

This step may take a few moments. Once it's done, the command should output something like this:

added 224 packages, and audited 225 packages *in* 14s  
  
40 packages are looking *for* funding  
 run `npm fund` *for* details  
  
found 0 vulnerabilities

Your script is good to go! Next, we'll [**run the script to upload a file. ↓**](https://web3.storage/docs/#run-the-script)

## Run the script[​](https://web3.storage/docs/#run-the-script)

Now that you've got your script ready to go, you just need to run it in your terminal window using node.

1. Run the script by calling node put-files.js, using --token to supply your API token and specifying the path and name of the file you want to upload. If you'd like to upload more than one file at a time, simply specify their paths/names one after the other in a single command. Here's how that looks in template form:

node put-files.js --token=<YOUR\_TOKEN> ~/filename1 ~/filename2 ~/filenameN

Once you've filled in your details, your command should look something like this:

node put-files.js --token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkaWQ6ZXRocjoweGZFYTRhODlFNUVhRjY5YWI4QUZlZUU3MUE5OTgwQjFGQ2REZGQzNzIiLCJpc3MiOiJ3ZWIzLXN0b3JhZ2UiLCJpYXQiOjE2MjY5Njk3OTY1NTQsIm5hbWUiOiJib25maXJlIn0.0S9Ua2FWEAZSwaemy92N7bW8ancRUtu4XtLS3Gy1ouA ~/hello.txt

##### Multiple files

You can upload a whole directory full of files at once by giving the script the path to a local directory. You can also upload multiple files by passing in more than one file path when you run the script.

1. The command will output a CID:

Content added with CID: bafybeig7sgl52pc6ihypxhk2yy7gcllu4flxgfwygp7klb5xdjdrm7onse

1. **Make a note of the CID, which looks like bafyb....** You'll need it in order to get your file.

##### Get the status of your upload

It's possible to get a variety of details about your upload, including its CID, upload date, size on the network, and info on IPFS pinning and Filecoin storage deals, by using the status() method within the JavaScript client library. Check out the [**Query how-to guide**](https://web3.storage/docs/how-tos/query#querying-for-status-information) for more information.

Next up, we'll go over two methods for you to [**retrieve your data from Web3.Storage ↓**](https://web3.storage/docs/#get-your-file)

## Get your file[​](https://web3.storage/docs/#get-your-file)

You've already done the most difficult work in this guide — getting your files from Web3.Storage is simple.

1. Go to https://dweb.link/ipfs/YOUR\_CID, replacing YOUR\_CID with the CID you noted in the last step.
2. You should see a link to your file. If you uploaded multiple files at once, you'll see a list of all the files you uploaded.
3. Click on a file's link to view it in your browser!

### Finding your files again[​](https://web3.storage/docs/#finding-your-files-again)

If you ever need to find your files again, and you've forgotten the CID, head over to the [**Files section**](https://web3.storage/files/) in Web3.Storage:

