Table of Contents

Introduction 1.1

Table of Contents 1.2

Tutorial: Install and Initialize IPFS 1.3

- Lesson: Download and Install IPFS 1.3.1
- Lesson: Initialize your IPFS Repository 1.3.2

Tutorial: Files on IPFS 1.4

- Lesson: Add Content to IPFS and Retrieve It 1.4.1
- Lesson: Wrap Filenames and Directory Info around Content 1.4.2
- Lesson: Pinning - Tell IPFS to Keep a File 1.4.3

Tutorial: Going Online - Joining the Distributed Web 1.5

- Lesson: Publish your node on the IPFS network 1.5.1
- Lesson: Find Peers on the Network 1.5.2
- Lesson: Retrieve content from a Peer 1.5.3

Tutorial: Interacting with the Classical (HTTP) Web 1.6

- Lesson: Use an HTTP browser to retrieve files from local IPFS gateway 1.6.1
- Lesson: Get content through the public ipfs.io gateway 1.6.2
- Lesson: Access IPFS content through any IPFS gateway 1.6.3
- (TODO) Lesson: Map DNS to IPNS 1.6.4
- (TODO) Lesson: Streaming Video over IPFS 1.6.5

Tutorial: The Myriad ways to Access and Distribute IPFS Content 1.7

- The Power of Content-addressing 1.7.1
- Retrieving content from a peer 1.7.2
- Review these lessons from the Tutorial on Interacting with the Classical (HTTP) Web 1.7.3
  - Review: Using an HTTP browser to retrieve files from local IPFS gateway 1.7.3.1
  - Review: Using the public IPFS gateways at ipfs.io 1.7.3.2
  - Review: Access IPFS content through any IPFS gateway 1.7.3.3

Lesson: Access IPFS content through Tor gateways (experimental) 1.7.4
Lesson: Run IPFS over Tor transport (experimental) 1.7.5
Lesson: Access IPFS content through a browser extension 1.7.6
Lesson: Sneakernets - moving the data on USB Drives and other Hardware 1.7.7
Tutorial: Publishing Changes on the Permanent Web 1.8
Lesson: Set up IPNS on your IPFS Node 1.8.1
Lesson: Create an IPNS entry that points to your file 1.8.2
Lesson: Modify your File and add the modified version to IPFS 1.8.3
Lesson: Update the IPNS entry to point to the new version 1.8.4
Privacy and Access Controls on the Distributed Web 1.9
Reader Privacy & Writer Privacy 1.9.1
Private Networks 1.9.2
Encrypting Content 1.9.3
More dynamic encryption: capabilities-based encryption 1.9.4
Comparing with the classic HTTP web (feudal security, etc) 1.9.5
Keeping Data Alive: Durable Data on the Permanent Web 1.10
IPFS Cluster 1.10.1
Filecoin 1.10.2
Tutorial: Merkle Trees and the IPFS DAG 1.11
Lesson: Turn a file into a tree of hashes 1.11.1
Lesson: Create a cryptographic hash 1.11.2
Lesson: Build a tree of data in IPFS using cryptographic hashes to link the pieces (a Merkle DAG) 1.11.3
Lesson: Explore the types of software that use hash trees to track data 1.11.4
Tutorial: Dynamic Content on IPFS 1.12
Disclaimer: Dynamic content on IPFS is a Work in Progress 1.12.1
Lesson: Add data to the DAG (locally) 1.12.2
Lesson: Tell peers about your Changes 1.12.3
Lesson: Use hashes to get someone's changes from IPFS 1.12.4
Lesson: Use a pub/sub strategy to pass around messages about changes 1.12.5

Lesson: Resolve conflicts with a merge strategy (CRDTs) 1.12.6
Distributed Computation 1.13

The Decentralized Web Primer

This primer contains a series of Tutorials explaining IPFS, Merkle Trees and the Decentralized Web. It's written and maintained as a gitbook so people can read it in many formats.

When this is finished, this primer will be a complete replacement for the examples at https://ipfs.io/docs/examples/

The github repository for this book is at https://github.com/flyingzumwalt/decentralized-web-primer

Getting Help

During these tutorials, if you have any questions feel free to ask them in ipfs/support or in the #ipfs channel on chat.freenode.net. We have a large, active community who use these venues as our main places to seek support and to provide it.

Tutorials

The primer contains Tutorials about

1. Downloading and Installing IPFS
2. Files on IPFS
3. Going Online - Joining the Distributed Web
4. Interacting with the Classical (HTTP) Web
5. The Myriad ways to Access and Distribute IPFS Content
6. Merkle Trees and the IPFS DAG
7. Dynamic Content on IPFS

   For a full list of Tutorials, look at the Table of Contents

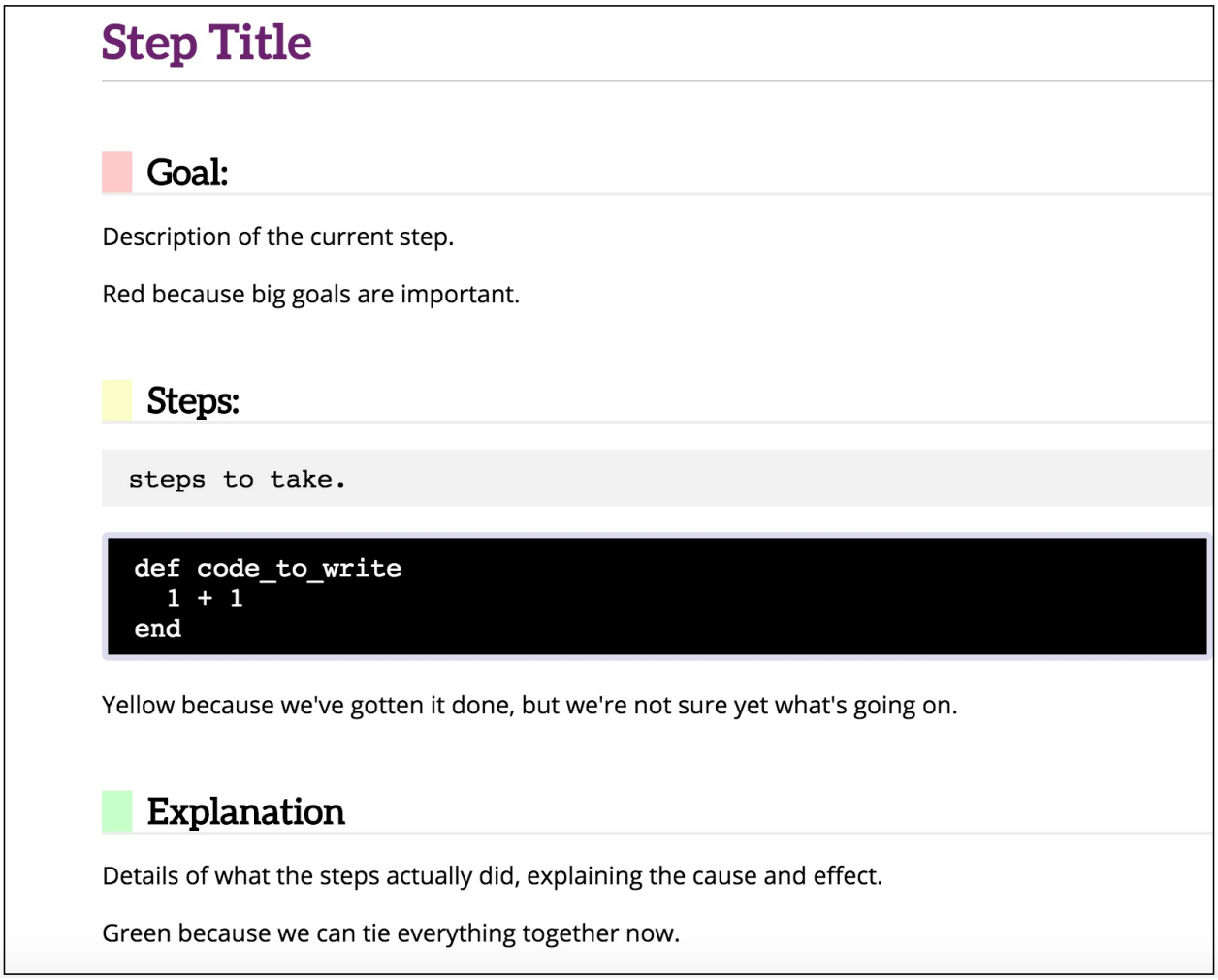
Concepts

- Cryptographic Hashes and Content Addressability
- Authenticated Graphs

- Turning Files into Trees
- Turning any Data into Trees
- Publishing hashes on the DHT
- Getting data from the Peer to Peer Network
- Immutability: "Changes" as *additions* to the tree
- CRDTs
- Pubsub
- Authenticated Streams (with pubsub)

Format

Each tutorial is a set of *lessons* that all use a format inspired by the Railsbridge Curriculum. Each lesson declares a set of *Goals*, or learning objectives, then lists the *Steps*, or activities, and finally provides an *Explanation* that reviews what you've done and connects those activities to the lesson's stated Goals. The format for each lesson looks like this:



Note for Contributors

For a concise explanation of Learning Objectives (which we're calling *Goals* in this book), read UC Denver's Assessment & Instructional Alignment Tutorial. Try to make your learning objectives Specific, observable and measurable and heed their tip to use the list of verbs in the taxonomy table worksheet to help you choose observable behaviors for your course learning objectives.

Contributors

This primer was created by @flyingzumwalt

The contents of these tutorials were initially pulled from documentation in the ipfs website and ipfs examples git repositories.

Contributors to those original docs included

- @whyrusleeping
- @jbenet
- @lgierth
- @lynnandtonic
- @wraithgar
- @adambrault
- @donothesitate
- @djdv plus a long list of contributors to the examples repository

Tutorial: Install and Initialize IPFS

These Lessons are tested with go-ipfs version 0.4.4. *Please update this file on github to reflect any other versions that have been tested.*

Prerequisites

- You should have some familiarity with the commandline

Learning Objectives

These Lessons will teach you how to

- Install IPFS
- Initialize an IPFS repository
- Locate where IPFS stores the contents of your local IPFS repository

Key Concepts

- IPFS Repositories

Lessons

1. Lesson: Download and Install IPFS
2. Lesson: Initialize your IPFS Repository

Next Steps

Once you know how to add files to IPFS and retrieve them, you will be ready to share those files on the P2P network following the Tutorial: Going Online - Joining the Distributed Web

If you're wondering how to update those files after you've shared them, see the Tutorial: Publishing Changes on the Permanent Web

If you want to see how to access those files from the conventional HTTP web, go to the Tutorial: Interacting with the Classical (HTTP) Web

Lesson: Download and Install IPFS

Goals

After doing this Lesson you will be able to

- Download IPFS and install it on your operating system
- Display which version of IPFS you're using
- Get a list of commands the ipfs binary supports

Steps

Step 1: Download the Prebuilt IPFS Package

Visit the IPFS installation page at https://ipfs.io/docs/install/ and download the prebuilt ipfs binaries for your operating system.

Why does the installation page talk about "Go IPFS"? There are multiple implementations of the IPFS protocol. The core IPFS team maintain implementations in Golang and Javascript. Those are commonly referred to as go-ipfs and js-ipfs. The official binaries are built from the Go implementation.

Step 2: Unzip the Prebuilt Package

The binaries for Mac OSX and Linux are in a gzipped tar format (`.tar.gz`). The binaries for Windows are in a zip file. Use the appropriate tool to unzip the file. There are some hints on https://ipfs.io/docs/install/ under the heading *Installing from a Prebuilt Package*

This will create a directory called go-ipfs.

```
LICENSE        README.md    build-log    install.sh ipfs
```

The file named `ipfs` is your executable ipfs binary.

Step 3: Install the IPFS Binary on your executable path

To install the binary, all you need to do is put the `ipfs` binary file somewhere on your executable PATH.

Note about permissions: Whichever approach you use to install the binary, make sure you have the necessary permissions. On Mac OSX or Linux, you probably want to use sudo, which is already installed on most systems.

If you're on Mac OSX or Linux, you can use the provided install script by running

```
cd go-ipfs
sudo ./install.sh
```

Read the output from running this. If it complains about being unable to write the file, you need to deal with permissions (see the note above about permissions)

Step 4: Display the IPFS version

When you're troubleshooting, it's important to know which version of ipfs you're using. To find out the current version, run

```
$ ipfs version
```

Step 5: Display the IPFS help page and list of commands

If you need help remembering how to use any ipfs commands, run

```
$ ipfs help
```

This should display information beginning with

```
USAGE:

    ipfs - Global p2p merkle-dag filesystem.
...
```

For a complete list of commands that the ipfs executable supports, run

```
$ ipfs commands
```

Next Steps

Next, Initialize your IPFS Repository

Lesson: Initialize your IPFS Repository

Goals

After doing this Lesson you will be able to

- Initialize a local ipfs repository
- Locate where IPFS stores the contents of your local IPFS repository
- Open the IPFS Configuration file

Steps

Step 1: Initialize the Repository

Use the `ipfs init` command to initialize the repository. This will generate a local ipfs repository for the current user account on your machine. It also generates a cryptographic keypair that allows your ipfs node to cryptographically sign the content and messages that you create.

```
$ ipfs init
initializing ipfs node at /Users/jbenet/.go-ipfs
generating 2048-bit RSA keypair...done
peer identity: Qmcpo2iLBikrdf1d6QU6vXuNb6P7hwrbNPW9kLAH8eG67z
to get started, enter:

  ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/readme
```

Note: If you have already initialized ipfs on your machine, you will get an error message like:

```
initializing ipfs node at /Users/sally/.ipfs
Error: ipfs configuration file already exists!
Reinitializing would overwrite your keys.
```

This is ok. It means you've already done this step. You can safely proceed to Step 2.

Step 2: Use IPFS to explore the post-install documentation

> If you installed a different version of ipfs, you may have gotten a slightly different path to use here. Either path will work for this tutorial. The path you got from the ipfs init command will give you documentation that's accurate for the version of ipfs you're using.

When you ran `ipfs init`, it provided a hint for how you can get started. It said:

```
to get started, enter:

  ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/readme
```

This `ipfs cat` command tells ipfs to read the content matching the path you provided. If the content isn't available locally, ipfs will attempt to find it on the peer-to-peer network.

Run the `ipfs cat` command with the path you got from the init message:

```
$ ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/readme
```

You should see something like this:

```
Hello and Welcome to IPFS!

                           ██╗██████╗ ███████╗███████╗
                           ██║██╔══██╗██╔════╝██╔════╝
                           ██║██████╔╝█████╗  ███████╗
                           ██║██╔═══╝ ██╔══╝  ╚════██║
                           ██║██║     ██║     ███████║
                           ╚═╝╚═╝     ╚═╝     ╚══════╝

If you're seeing this, you have successfully installed
IPFS and are now interfacing with the ipfs merkledag!

 ------------------------------------------------------
| Warning:                                             |
|   This is alpha software. use at your own discretion! |
|   Much is missing or lacking polish. There are bugs.  |
|   Not yet secure. Read the security notes for more.   |
 ------------------------------------------------------

Check out some of the other files in this directory:

  ./about
  ./help
  ./quick-start     <-- usage examples
  ./readme          <-- this file
  ./security-notes
```

You can explore other objects in there. For example, check out `security-notes`:

```
ipfs cat /ipfs/QmYwAPJzv5CZsnA625s3Xf2nemtYgPpHdWEz79ojWnPbdG/security-notes
```

Step 3: Locate where IPFS Stores the Repository Contents on your Machine

`ipfs` stores its local object repository in `~/.ipfs`

```
$ ls ~/.ipfs
```

The contents of that directory look like this:

```
blocks        config        datastore    version
```

All of the contents of your IPFS repository are stored within this directory. For example, the readme file from above is stored in here, along with the other files it links to. You can run a grep to find out the exact location.

Step 4: Open the IPFS Configuration file

The configuration for your ipfs repository is in a json file that's usually stored at `~/.ipfs/config`. To view the current config, run:

```
$ ipfs config show
```

One of the useful details in this config file is at `Datastore.Path`. This tells you where the ipfs repository's contents are being stored. As we saw in Step 3, this is usually `~/.ipfs`

Next Steps

Next, proceed to the Files on IPFS tutorial.

Tutorial: Files on IPFS

These Lessons are tested with go-ipfs version 0.4.8. *Please update this file on github to reflect any other versions that have been tested.*

Prerequisites

- You should have some familiarity with the command line.
- You should have `ipfs` installed - the previous tutorial has instructions for that

Learning Objectives

These Lessons will teach you how to

- Add files to your local IPFS node
- Read files out of your local IPFS node
- List the files in your IPFS node
- Tell IPFS to hold onto files by *pinning* them

Key Concepts

- Distinction between IPFS and your regular Filesystem
- Identifying files by their Hashes
- IPFS Garbage Collection
- Pinning files on an IPFS Node

Lessons

1. Lesson: Add Content to IPFS and Retrieve It
2. Lesson: Wrap Filenames and Directory Info around Content in IPFS
3. Lesson: Pinning - Tell IPFS to Keep a File

Next Steps

Once you know how to add files to IPFS and retrieve them, you will be ready to share those files on the P2P network following the Tutorial: Going Online - Joining the Distributed Web

If you're wondering how to update those files after you've shared them, see the Tutorial: Publishing Changes on the Permanent Web

If you want to see how to access those files from the conventional HTTP web, go to the Tutorial: Interacting with the Classical (HTTP) Web

If you want to know more about how IPFS stores this content intrernally using Merkle DAGs, go to the Tutorial: Merkle Trees and the IPFS DAG

Lesson: Add Content to IPFS and Retrieve It

Goals

After doing this Lesson you will be able to

- Add a file's content to IPFS
- Read content out of IPFS using its hash
- Explain the relationship between IPFS hashes and the content you've added

Steps

Step 1: Create a file that you will add to IPFS

> You can add any type of content to IPFS. For this lesson we will put some text content into a `.txt` file, but you can do this same process with any content or any file.

It would be a good idea to make a new directory for this example. Navigate to somewhere you are comfortable putting a new folder (such as `~/Desktop`), and then create a new directory and go into it. Here is an example command:

```
$ cd ~/Desktop
$ mkdir ipfs-tutorial
$ cd ipfs-tutorial
```

Now, create a file called `mytextfile.txt` and put the text "version 1 of my text" in it. One easy way to do this on the command line is with this command:

```
$ echo "version 1 of my text" > mytextfile.txt
```

You can read the file's contents using the `cat` command:

```
$ cat mytextfile.txt
version 1 of my text
```

Step 2: Add the File to IPFS

```
$ ipfs add mytextfile.txt
added QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy mytextfile.txt
```

Save the hash `QmZtmD2qt...` that ipfs returned. This is the content's cryptographic hash. If the file's content changes, the hash will change, but if the file's content remains the same, the hash will always be the same.

Step 3: Read the content out of IPFS

Just like the regular `cat` command lets you read the contents of a file, the `ipfs cat` command lets you read the contents of a file that has been added to ipfs.

Use the `ipfs cat` command to read the content by passing it the content's cryptographic hash -- this is the hash that ipfs returned when you ran `ipfs add mytextfile.txt`.

```
$ ipfs cat QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy
version 1 of my text
```

Notice that this returned the *content* of the file, not the text file itself. That's because `QmZtmD2qt...` is the hash of the content, not the file itself. We'll test that in the next step.

Step 4: Confirm that the hash points to the content, not the file

When we used `ipfs cat` to read the file's contents it returned the *content* of the file, not the text file itself. That's because the hash `QmZtmD2qt...` is the hash of the *content*. We can test that by adding the text content directly to IPFS without ever putting it in a file.

```
$ echo "version 1 of my text" | ipfs add
added QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy QmZtmD2qt6fJot32nabSP3CUjicny
pEBz7bHVDhPQt9aAy
```

The hash should be exactly the same as the hash you got when you added mytextfile.txt. If you want to triple-check, you can run each of these commands as many times as you want. The hash should always be the same.

```
$ ipfs add mytextfile.txt
added QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy mytextfile.txt
$ echo "version 1 of my text" | ipfs add
added QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy QmZtmD2qt6fJot32nabSP3CUjicny
pEBz7bHVDhPQt9aAy
$ cat mytextfile.txt | ipfs add
added QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy QmZtmD2qt6fJot32nabSP3CUjicny
pEBz7bHVDhPQt9aAy
```

As long as the content remains the same, you will always get the same hash. As far as IPFS is concerned, it *is* the same content.

Step 5: Change the content and get a different hash

Now change the text content to "version 2 of my text" and add it to ipfs. You will get a different hash.

As we confirmed in the previous step, you can add the new text directly to IPFS or you can modify mytextfile.txt and add it to IPFS. You will get the same hash either way.

```
$ echo "version 2 of my text" | ipfs add
added QmTudJSaoKxtbEnTddJ9vh8hbN84ZLVvD5pNpUaSbxwGoa QmTudJSaoKxtbEnTddJ9vh8hbN84Z
LVvD5pNpUaSbxwGoa
```

Step 5: Pipe content from IPFS into a File

You can read this content (any version) out of ipfs and write it into a file. For example, you can toggle the contents of mytextfile.txt from "version 1" to "version 2" and back as many times as you want:

```
$ ipfs cat QmTudJSaoKxtbEnTddJ9vh8hbN84ZLVvD5pNpUaSbxwGoa > mytextfile.txt
$ cat mytextfile.txt
version 2 of my text
$ ipfs cat QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy > mytextfile.txt
$ cat mytextfile.txt
version 1 of my text
```

You can also write the content from ipfs into a completely new file.

```
$ ipfs cat QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy > anothertextfile.txt
$ cat anothertextfile.txt
version 1 of my text
```

Explanation

IPFS tracks content based on its cryptographic hash. **This hash uniquely identifies exactly that content.** As long as the content stays the same, the hash stays the same, but if the content changes at all you will get a different hash.

If you have two different files that contain identical content, IPFS will track that content with one hash. The filenames are different, but the content is the same, so the hash of the content will be identical.

This leads to the question: how does IPFS track file names? That's the topic of the next lesson.

Next Lesson: Add Filenames and Directory Info to IPFS

Proceed to the next lesson to learn how to Wrap Filenames and Directory Info around Content in IPFS

Lesson: Wrap Filenames and Directory Info around Content in IPFS

Goals

After doing this Lesson you will be able to

- Add a file to IPFS, including its filename, permissions, etc.
- Add directories to IPFS
- Explain how IPFS represents two files that have identical content
- Read content out of IPFS using the hash of a directory that contains the file

Steps

Step 1: Create the file you're going to add

> You may already have this file from the previous lesson. If you do, make sure the content of the file matches. Otherwise the hashes you get won't match the examples in this lesson.

Create a file called `mytextfile.txt` and put the text "version 1 of my text" in it. One easy way to do this on the command line is with this command:

```
$ echo "version 1 of my text" > mytextfile.txt
```

Step 2: Add the file to IPFS

```
$ ipfs add -w mytextfile.txt
added QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy mytextfile.txt
added QmPvaEQFVvuiaYzkSVUp23iHTQeEUpDaJnP8U7C3PqE57w
```

In the previous lesson, when we ran `ipfs add mytextfile.txt` without the `-w` flag, ipfs only returned one hash. This time it returned two hashes. The first hash `QmZtmD2...` is the same as before — it's the hash of the content *inside* the file. The second hash `QmPvaEQF...` is the hash of the directory and filename information that ipfs "wrapped" around our content.

In the next steps, we will use ipfs commands to see what that directory and filename information looks like and how we can use it.

Step 3: List the directory information

The `-w` flag tells ipfs to include the directory and filename information along with the content — it "wraps the file in a directory". For more info about this, run `ipfs add --help` and read the description there.

To list this directory and filename information, use `ipfs ls`. We will use the `-v` flag to include header information. To learn more about this command, run `ipfs ls --help`

```
$ ipfs ls -v QmPvaEQFVvuiaYzkSVUp23iHTQeEUpDaJnP8U7C3PqE57w
Hash                                           Size Name
QmZtmD2qt6fJot32nabSP3CUjicnypEBz7bHVDhPQt9aAy 29   mytextfile.txt
```

This command `ipfs ls QmPvaEQFVvuiaYzkSVUp23iHTQeEUpDaJnP8U7C3PqE57w` translates to "list the files referenced by the directory whose hash is QmPvaEQFVvuiaYzkSVUp23iHTQeEUpDaJnP8U7C3PqE57w".

The response shows that the directory contains one file — "mytextfile.txt" — and the hash of that file's content is `QmZtmD2q...`

Note that we had to use `ipfs ls` instead of `ipfs cat` to read this info because it's a directory. If you try to read the directory using `ipfs cat` you will get an error:

```
$ ipfs cat QmPvaEQFVvuiaYzkSVUp23iHTQeEUpDaJnP8U7C3PqE57w
Error: this dag node is a directory
```

Step 4: Read the File's contents using the parent directory's hash

We can use the directory's hash to read the file's content like this:

```
$ ipfs cat QmPvaEQFVvuiaYzkSVUp23iHTQeEUpDaJnP8U7C3PqE57w/mytextfile.txt
version 1 of my text
```

This command translates to "return the content that's referred to as `mytextfile.txt` within the directory whose hash is QmPvaEQFVvuiaYzkSVUp23iHTQeEUpDaJnP8U7C3PqE57w"

Bonus Steps

Some things to try:

1. Create a directory with multiple files. Tell ipfs to recursively add the directory and all of its files.
2. Create two different files with the same content. Add them both to ipfs with `ipfs add -w` and confirm that ipfs is re-using the hash of that content when it builds the directory and filename information.

Explanation

When you add a file to your ipfs repository, ipfs calculates the cryptographic hash of the file's contents and returns that hash to you. You can then use the hash to reference the file's contents and read them back out of the ipfs repository.

In order to keep track of information like filenames and paths, ipfs lets you "wrap" directory and filename information around the file contents you've added. That directory and filename information has its own hashes, which makes it possible to retrieve content from the ipfs repository using "ipfs paths" that are a combination of hashes, filenames and directory names.

Next Steps

Next, learn how to Tell IPFS to Keep a File

Lesson: Making sure that a file persists in your IPFS Repository

Goals

This lesson covers the topic of "pinning" files in your IPFS repository and removing files with the ipfs garbage collector.

After doing this Lesson you will be able to

- Tell ipfs to hold onto specific files in your local ipfs repository
- Tell ipfs to clean up unwanted files from your local ipfs repository

Steps

TODO: Build these steps based on https://ipfs.io/ipfs/QmNZiPk974vDsPmQii3YbrMKfi12KTSNM7XMiYyiea4VYZ/example#/ipfs/QmRFTtbyEp3UaT67ByYW299Suw7HKKnWK6NJMdNFzDjYdX/pinning/readme.md

1. *todo*

Explanation

Next Steps

Tutorial: Going Online - Joining the Distributed Web

Prerequisites

To do the lessons in this tutorial you must:

- Be familiar with using the command line
- Install and Initialize IPFS on your local machine

Learning Objectives

Key Concepts

Lessons

These lessons have not been written yet. If you want to help work on them, or if you want to encourage us to give them attention, open an issue at https://github.com/flyingzumwalt/decentralized-web-primer/issues

1. Lesson: Publish your node on the IPFS network
2. Lesson: Find Peers on the Network
3. Lesson: Retrieve content from a Peer
4. Lesson: Streaming Video over IPFS

Lesson: Retrieve content from a Peer

This lesson shows how to use an IPFS node on your computer to request content from other peers on the network. Some of the underlying topics are covered in greater depth in the tutorials about Files on IPFS.

Prerequisites

To do the steps in this lesson you must:

- Be familiar with using the command line
- Install and Initialize IPFS on your local machine

Goals

After doing this Lesson you will be able to

- Access any content through your local IPFS node using its command line interface

Steps

Step 1: Start the IPFS daemon

Start the IPFS daemon by running

```
$ ipfs daemon
```

If the daemon is not running, your IPFS node won't be able to retrieve content from other nodes on the network.

Step 2: Read the content on the command line

You can use the command line to request content from your IPFS node. If the node does not have a copy of that content, it will attempt to find another peer node that does have the content. For example, the IPFS team have published a snapshot of the turkish

version of wikipedia. The hash of that snapshot, which contains about 15GB of Turkish-language wikipedia pages, is `Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1`. We can use the command line to have your IPFS node read pages from that snapshot.

```
# get the article about "Peer to Peer"
ipfs cat Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Peer-to-peer.html > P
eer-to-peer.html

# get a picture of Alexis de Tocqueville
ipfs cat Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/m/Alexis_de_Tocqueville.jp
g > Alexis_de_Tocqueville.jpg

# explore the articles in the snapshot
ipfs ls Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html
```

If you're not familiar with the `ipfs cat` and `ipfs ls` commands they are explained in the Tutorial about Files on IPFS

Explanation

You can use a local IPFS node to read content from the worldwide IPFS network. One way to do this is through the command line using commands like `ipfs cat` and `ipfs ls`. When you pass the content-addressed (hash) identifiers of the content you want into these commands, your IPFS node will check to see if it has a local copy of the content you're requesting. If your node has a local copy, it will return that content to you immediately. If your node does not have a local copy, it will attempt to find a peer on the IPFS network that does have the content. As long as *at least one peer* has the content you want, your IPFS node will be able to find that peer, retrieve the content from the peer, and return that content to you.

This is the essential function of an IPFS node. It uses content-addressed (hash) identifiers to find content on the peer to peer network. It also provides that content to other peers who want it.

Next Steps

This lesson covered how to use the command line to request content from your IPFS node, but there are many other ways to interact with IPFS nodes. If you want to learn about the many other ways you can use IPFS to access the same content using the

same content-addressed link, go to the Tutorial on Avenues for Access.

Otherwise return to the tutorial about Going Online - Joining the Distributed Web

Tutorial: Interacting with the Classical (HTTP) Web

Prerequisites

To do the lessons in this tutorial you must:

- Be familiar with using the command line
- Install and Initialize IPFS on your local machine
- Take your IPFS Node Online

Learning Objectives

Key Concepts

Lessons

Some of these lessons have not been written yet. If you want to help work on them, or if you want to encourage us to give them attention, open an issue at https://github.com/flyingzumwalt/decentralized-web-primer/issues

1. Lesson: Use an HTTP browser to retrieve files from local IPFS gateway
2. Lesson: Get content through the public ipfs.io gateway
3. Lesson: Access IPFS content through any IPFS gateway
4. (TODO) Lesson: Map DNS to IPNS
5. (TODO) Lesson: Streaming Video over IPFS

Lesson: Using an HTTP browser to retrieve files from a local IPFS gateway

This lesson shows some of the different ways to access content through an IPFS node that you have installed locally on your computer. Some of the underlying topics are covered in greater depth in the tutorials about Files on IPFS and Going Online - Joining the Distributed Web.

Prerequisites

To do the steps in this lesson you must:

- Install and Initialize IPFS on your local machine

Goals

After doing this Lesson you will be able to

- Access any content through your local IPFS node's HTTP gateway

Steps

Step 1: Start the IPFS daemon

Start the IPFS daemon by running

```
$ ipfs daemon
```

If the daemon is not running, your IPFS node won't be able to retrieve content from other nodes on the network. It also won't start the HTTP gateway that you're going to use in Step 2.

Step 2: Read request content through your IPFS node's HTTP gateway

As described in the Lesson on Using an HTTP browser to retrieve files from local IPFS gateway, you must tell the gateway whether you're requesting content with an IPFS hash or an IPNS hash. If you're using the hash of a specific snapshot of content -- for example a file that someone added to IPFS, use a path that starts with `/ipfs/`. If you're using an IPNS hash to get the *latest* version of some content that gets updated over time, for example a website that gets fresh content every day, use a path that starts with `/ipns/`.

To view the wikipedia page we're using as an example in all of the lessons in the Tutorial on Avenues for Access, use these links:

- 2017-04-30 snapshot: http://localhost:8080/ipfs/Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html
- latest (IPNS): http://localhost:8080/ipns/QmQP99yW82xNKPxXLroxj1rMYMGF6Grwjj2o4svsdmGh7S/wiki/Anasayfa.html
- latest (DNS): http://localhost:8080/ipns/wikipedia-on-ipfs.io

Explanation

You can use a local IPFS node to read content from the worldwide IPFS network. The two ways of interacting with your local node are 1) through the command line and 2) through the HTTP gateway. You can use either of those interfaces to pass IPFS the content-addressed (hash) identifiers of the content you want. The IPFS node will use those identifiers to find that content on the network and retrieve it for you.

Next Steps

If you want to learn about the many other ways you can use IPFS to access the same content using the same content-addressed link, go to the Tutorial on Avenues for Access.

Otherwise proceed to the next lesson to learn how to Get content through the public ipfs.io gateway

Lesson: Using the public IPFS gateways at ipfs.io to access content

This lesson explains how to retrieve IPFS content from the public IPFS gateways at ipfs.io. This topic is covered in greater depth in the tutorial on Interacting with the Classical (HTTP) Web.

Goals

After doing this Lesson you will be able to

- Use the public gateway at ipfs.io to access IPFS content

How to Do It

This process is the same as using any other IPFS gateway -- only the address of the gateway is different: If you're using the hash of a specific snapshot of content, use the path `https://ipfs.io/ipfs/<your-ipfs-hash>`. If you're using an IPNS hash to get the *latest* version of some content, use the path `https://ipfs.io/ipns/<your-ipns-hash>`

To view the wikipedia page we're using as an example in all of the lessons in the Tutorial on Avenues for Access, use these links:

- 2017-04-30 snapshot: http://ipfs.io/ipfs/Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html
- latest (IPNS): http://ipfs.io/ipns/QmQP99yW82xNKPxXLroxj1rMYMGF6Grwjj2o4svsdmGh7S/wiki/Anasayfa.html
- latest (DNS): http://ipfs.io/ipns/wikipedia-on-ipfs.io

Explanation

The IPFS project maintains public IPFS gateways that you can use to access any content from the IPFS network. When sharing HTTP links to IPFS content, people often use ipfs.io addresses but you can use the address of any gateway.

Next Steps

If you want to learn about the many other ways you can use IPFS to access the same content using the same content-addressed link, go to the Tutorial on Avenues for Access.

Otherwise proceed to the next lesson to learn how to Access IPFS content through any IPFS gateway

Lesson: Access IPFS content through any IPFS gateway

Goals

This lesson covers using *any* IPFS gateway to access IPFS content. It's a condensed review of the Lesson on Using an HTTP browser to retrieve files from a local IPFS gateway

After doing this Lesson you will be able to

- Use the HTTP address of any IPFS gateway to access IPFS content

Steps

Step 1: Get the address of a gateway

As we covered in Tutorial: Going Online - Joining the Distributed Web, when you run an IPFS daemon, it exposes an HTTP endpoint that acts as a gateway between HTTP and the IPFS network. This means that you can, in theory, point your web browser at any IPFS node's HTTP endpoint and use it as a gateway. In reality, the person operating that node usually needs to take extra steps to make their gateway available over HTTP (NAT traversal, etc).

For these examples we will use the gateway at `https://dweb.link`

Step 2: Build the Path to your Content

As described in the Lesson on Using an HTTP browser to retrieve files from local IPFS gateway, you must tell the gateway whether you're requesting content with an IPFS hash or an IPNS hash. If you're using the hash of a specific snapshot of content -- for example a file that someone added to IPFS, use the path `/ipfs/<your-ipfs-hash>`. If you're using an IPNS hash to get the *latest* version of some content that gets updated over time, for example a website that gets fresh content every day, use the path `/ipns/<your-ipns-hash>`

Step 3: Request the content from the gateway

Combine the gateway's address (ie. `https://dweb.link`) with the path to your content (ie. `/ipfs/<your-ipfs-hash>`). Use that to request the content.

To view the wikipedia page we're using as an example in all of the lessons in the Tutorial on Avenues for Access, use these links:

- 2017-04-30 snapshot: http://dweb.link/ipfs/Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html
- latest (IPNS): http://dweb.link/ipns/QmQP99yW82xNKPxXLroxj1rMYMGF6Grwjj2o4svsdmGh7S/wiki/Anasayfa.html
- latest (DNS): http://dweb.link/ipns/wikipedia-on-ipfs.io

Explanation

This explanation has not been written yet. If you want to help work on it, or if you want to encourage us to give it attention, open an issue at https://github.com/flyingzumwalt/decentralized-web-primer/issues

TODO

- Restricting the content that your gateway will serve
- Security concerns -- the gateway can see all the things that an HTTP server can see.

Next Steps

If you want to learn about the many other ways you can use IPFS to access the same content using the same content-addressed link, go to the Tutorial on Avenues for Access.

Otherwise return to the tutorial on Interacting with the Classical (HTTP) web

Tutorial: The Myriad ways to Access and Distribute IPFS Content

These Lessons are tested with go-ipfs version 0.4.8. *Please update this file on github to reflect any other versions that have been tested.*

IPFS hashes are permanent, content-addressed identifiers for your content. This means that you can use many different ways to access, replicate, and/or redistribute the same content using the same link/identifier. The lessons in this tutorial explore many of the ways that you can do this. If you want to learn about why its valuable to have all of these options, read the lesson on the power of content-addressing

All of the lessons use the same content: TODO [the entry on Content-addressable networks] within a TODO[complete snapshot of wikipedia]

(based on https://en.wikipedia.org/wiki/Content_addressable_network)

Learning Objectives

These Lessons will teach you how to

- Define *content addressing* and compare it with *location-addressing*
- Use IPFS content hashes to access the same content in many ways with the same link
- Access content through the public IPFS gateways at ipfs.io
- Access content through any IPFS node's http gateway
- Access content using the IPFS browser extension
- Access IPFS content through Tor
- Use a sneakernet to move and redistribute IPFS content
- Explain the implications of being able to access IPFS content through so many different paths

Lessons

1. Read about The Power of Content-addressing
2. Review the lesson on Retrieving content from a peer

3. Review these lessons from the Tutorial on Interacting with the Classical (HTTP) Web
    - Review: Lesson: Using an HTTP browser to retrieve files from local IPFS gateway
    - Review: Lesson: Using the public IPFS gateways at ipfs.io
    - Review: Lesson: Access IPFS content through any IPFS gateway
4. Lesson: Access IPFS content through Tor gateways (experimental)
5. Lesson: Run IPFS over Tor transport (experimental)
6. Lesson: Access IPFS content through a browser extension
7. Lesson: Sneakernets - moving the data on USB Drives and other Hardware

Next Steps

If you're wondering how to update content after you've shared it, see the Tutorial: Publishing Changes on the Permanent Web

If you want to know more about how IPFS stores this content intrernally using Merkle DAGs, go to the Tutorial: Merkle Trees and the IPFS DAG

Lesson: The Power of Content-addressing

Goals

This lesson introduces the concept of *content addressing* and explores the powerful implications of using this approach.

After doing this Lesson you will be able to

- Define *content addressing* and compare it with *location-addressing*
- Explain the implications of being able to access IPFS content through so many different paths

Explanation

The Problem: Identifying Content by its Location

When you use an `http://` or `https://` link to point to a webpage, image, spreadsheet, dataset, tweet, etc, you're identifying content by its location. The link is an identifier that points to a particular location on the web, which corresponds to a particular server, or set of servers, somewhere on the web. **Whoever controls that location controls the content.** That's how HTTP works. It's *location-addressed*. Even if a thousand people have downloaded copies of a file, meaning that the content exists in a thousand locations, HTTP points to a single location. This location-addressed approach forces us all to pretend that the data are in only one location. Whoever controls that location decides what content to return when people use that link. They also decide whether to return any content at all.

To get a sense of how impractical it is to address content by its location, imagine if I used location-addressing to recommend the book Why Information Grows.

If I identify the book by its content, saying "Check out the book called *Why Information Grows* by César Hidalgo. The ISBN is 0465048994.", you will be able to get any copy of the book from any source and know that you're reading the information I recommended.

You might even say "Oh. I already read it." or "My roommate has it in the other room. I'll borrow it from him.", saving yourself the cost or effort of getting another copy.

By contrast, if I used location-addressing to identify the book, I would have to point to a location, saying something like "Go to the news stand at Market & 15th in Philadelphia and ask for the thing 16 inches from the south end of the third shelf on the east wall" Those instructions are confusing and awkward, but that's how http links work. They identify content by its location and they rely on the 'host' at that location to provide the content to visitors. There are lots of things that could go wrong with this approach. It also puts a lot of power and responsibility on the shoulders of whoever controls the location you're pointing to - in this case the news stand.

Let's consider the responsibilities of whoever controls the location we've pointed to. If the people running the news stand want my directions (aka. my "link") to remain valid, allowing people to access the book, they have to:

- Always be open, 24/7, in case someone wants to read the book.
- Provide the book to *everyone* who seeks the book, whether it's one person or hundreds of thousands of people.
- Protect the integrity of the book by preventing anyone from tampering with it.
- Never remove the book from its shelf - if they get rid of it, or even move it, my link is broken and nobody will be able to use my instructions to find the book.

Along with those responsibilities come a great amount of power. The proprietors of the news stand control the location that my directions point to, so they can choose to:

- Dictate who is allowed to see the book.
- Move the book without telling anyone.
- Destroy the book.
- Charge people money to access the book or force them to watch ads when they walk in the door.
- Collect data about everyone who accesses my book, using that information however they want.
- Replace the book with something else -- They might not even put a book there, since my instructions are just describing a location, a malicious actor could replace the book with something dangerous, turning the location into a trap!

Location-addressing has worked on the web for 25 years, but it's starting to get painful and It's about to get much worse. As long as we continue to rely on it, the web will continue to be unstable, insecure, and prone to manipulation or exploitation.

The Solution: Identify Information by its Fingerprint, not its Location

The alternative is to identify content by its "fingerprint" rather than identifying it by its location. That way, when someone says "Look at the content with this fingerprint" you can get it from anyone who has a copy of the content. To do this, we identify content by its cryptographic hash. A cryptographic hash is a short string of letters and numbers that's calculated by feeding your content into a *cryptographic hash function* like SHA.

When we identify content in this way, using the content's cryptographic hash instead of its location to identify it, this is called *content-addressing*. The cryptographic hash for a piece of content never changes, which means **content addressing guarantees that the links will always return the same content**, regardless of *where* I retrieve the content from, regardless of *who* added the content to the network, and regardless of when the content was added. That's the essential power of using a *content-addressed* protocol like IPFS instead of using a *location-addressed* protocol like HTTP.

The Implications of Content Addressing

Content-addressed links are permanent. The link permanently points to *exactly* that content. This has many powerful implications. From a computer science perspective, any time we create data that uses content-addressed links, we are creating a persistent data structure. There is a great amount of literature on the applications for persistent data strucutres. For this lesson, we will call out just a few implications of *storing* and *sharing* data using a content-addressed protocol:

It lets us store data together.

This decentralized, content-addressed approach radically increases the durability of data. It ensures that data will not become endangered as long as anyone is still relying on it because anyone can hold a valid copy of the data they care about. If you hold a copy of a dataset on any of your devices, or if you pay someone to host it on an IPFS node for you, you become part of the network of stewards who protect that dataset from being lost. You won't have to worry about whether someone is going to turn off the servers where your data are hosted because *you are one of the hosts*. You and your peers hold the data among yourselves and are able to share the data directly with each other without relying on centralized points of failure.

It increases the integrity of data.

Decentralization also increases the integrity of data because links are content-addressed. This means we can validate data by checking the data's fingerprints against the links. That kind of validation is impossible with location-addressed links. This is especially powerful on the large scale, where millions of websites and datasets reference each other billions of times. With location-addressed links, all of those connections are brittle. With content-addressed links, the connections become resilient and reliable.

Links can come back to life.

As soon as any node has the content, everyone's links start working. Even if someone destroys all the copies on the network, it only takes one node adding the content in order to restore availability. A cryptographic hash *permanently* points to the content it was derived from, so IPFS links permanently point to their content. Even if the content becomes unavailable for a period, the links will work as soon as anyone starts providing the content again.

Harder to attack, easier to recover.

Even if the original publisher is taken down, the content can be served by anyone who has it. As long as at least one node on the network has a copy of the content, everyone will be able to get it. This means the responsibility for serving content can change over time without changing the way people link to the content and without any doubt that the content you're reading is exactly the content that was originally published.

The content you download is cryptographically verified to ensure that it hasn’t been tampered with.

IPFS can work in partitioned networks - you don’t need a stable connection to the rest of the web in order to access content through IPFS. As long as your node can connect to at least one node with the content you want, it works!

If one IPFS gateway gets blocked, you can use another one. IPFS gateways are all capable of serving the same content, so you’re not stuck relying on one point of failure.

Lightening the load: With IPFS, people viewing the content are also helping distribute the content (unless they opt out) and anyone can choose to pin a copy of some content on their node in order to help with access and preservation.

You can read anonymously. As with HTTP, IPFS can work over Tor and other anonymity systems

IPFS does not rely on DNS. If someone blocks your access to DNS or spoofs DNS in your network, it will not prevent IPFS nodes from resolving content over the peer-to-peer network. Even if you're using the DNSlink feature of IPFS, you just need to find a gateway that *does* have access to DNS. As long as the gateway you're relying on has access to DNS it will be able to resolve your DNSlink addresses.

IPFS does not rely on the Certificate Authority System, so bad or corrupt Certificate Authorities do not impact it.

You can move content via sneakernet! *This is very useful in areas with poor connectivity, due to resource limitations, security reasons, or censorship.* Even if your network is physically disconnected from the rest of the internet, you can write content from IPFS onto USB drives or other external drives, physically move them to computers connected to a new network, and re-publish the content on the new network. Even though you're on a separate network, IPFS will let nodes access the content using the same identifiers in both networks as long as at least one node on the network has that content.

IPFS nodes work hard to find each other on the network and to reconnect with each other after connections get cut.

(experimental) **You can even form private IPFS networks** to share information *only* with computers you've chosen to connect with.

Further Reading

Further light Reading:

- HTTP is obsolete. It's time for the distributed, permanent web
- Instructions for Saving Endangered Data

Videos to Watch:

- this part of a talk by Juan Benet talk

Academic Papers:

- The IPFS Whitepaper

Next Steps

Read the Tutorial on Avenues for Access to learn about the many different ways you can use IPFS to access the same content using the same content-addressed link.

Lesson: Access IPFS through Tor gateways (experimental)

Goals

This lesson covers accessing IPFS content through Tor gateways.

After doing this Lesson you will be able to

- Use the Tor browser and a public IPFS gateway on the Tor network to access IPFS content

Steps

Step 1: Download the Tor browser

If you do not already have the Tor browser installed, download the Tor browser from the Tor project by visiting https://www.torproject.org/projects/torbrowser.html.en

In some countries the Tor Project website is blocked or censored and it is not possible to download Tor directly. The Tor Project also hosts a mirror of Tor Browser Bundle on Github.

The GetTor service can also be used to download Tor Browser when the Project website and mirrors are blocked.

Step 2: Request the content you want from the IPFS-Tor gateway

`ipfs4uvgthshqonk.onion` is a volunteer-run IPFS Gateway on the Tor network. You will use this gateway to request IPFS content. *(Warning: The IPFS project does not run this gateway. We cannot guarantee stability or security.)* There are probably many other IPFS gateways on the Tor network. You can use any of them in this way -- simply replace `ipfs4uvgthshqonk.onion` with the name of the gateway you're trying to access.

With the Tor browser running, enter the hash of the IPFS content you want to retrieve. This part is the same as using any other IPFS gateway -- only the address of the gateway is different: If you're using the hash of a specific snapshot of content, use the path `https://ipfs4uvgthshqonk.onion/ipfs/<your-ipfs-hash>`. If you're using an IPNS hash to get the *latest* version of some content, use the path `https://ipfs4uvgthshqonk.onion/ipns/<your-ipns-hash>`

To view the wikipedia page we're using as an example in all of the lessons in the Tutorial on Avenues for Access, use these links:

- 2017-04-30 snapshot: https://ipfs4uvgthshqonk.onion/ipfs/Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html
- latest (IPNS): https://ipfs4uvgthshqonk.onion/ipns/QmQP99yW82xNKPxXLroxj1rMYMGF6Grwjj2o4svsdmGh7S/wiki/Anasayfa.html
- latest (DNS): https://ipfs4uvgthshqonk.onion/ipns/wikipedia-on-ipfs.io
- (you can verify it works with onion.link)

Explanation

This approach relies on the IPFS gateway at `ipfs4uvgthshqonk.onion` to retrieve content from the IPFS network for you. The difference with this gateway, as opposed to the gateways at ipfs.io, is that it's listening for requests directly over Tor protocol. This allows you to access the gateway anonymously.

Next Steps

Read about how you can configure an IPFS node to use the Tor transport or return to the Tutorial on Avenues for Access to learn about the many other ways you can use IPFS to access the same content using the same content-addressed link.

Lesson: Run IPFS over Tor transport (experimental)

IPFS has an experimental feature that allows an IPFS node to interact with other IPFS nodes over the Tor transport protocol. The goal of this feature is to allow IPFS nodes to anonymously communicate with each other. **This feature is experimental!** Until we have tested this feature and removed the "experimental" designation, you should assume that information about your node might leak.

In the meantime, a more secure way to protect your anonymity would be to access data using the tor browser and an IPFS tor gateway.

Prerequisites

To do the steps in this lesson you must:

- Be familiar with using the command line
- Install and Initialize IPFS on your local machine

Goals

After doing this Lesson you will be able to

- Configure an IPFS node to use the Tor transport
- Request content through that node

Steps

Step 1: Configure IPFS to use the Tor transport

TODO - *This explanation has not been written yet. If you want to help work on it, or if you want to encourage us to give it attention, open an issue at https://github.com/flyingzumwalt/decentralized-web-primer/issues*

Step 2: Start the IPFS daemon

Start the IPFS daemon

```
$ ipfs daemon
```

For more info about this step, read Tutorial: Going Online - Joining the Distributed Web

Step 3: Request the content you want from your local IPFS node's gateway

This step is the same as using any other IPFS gateway -- only the address of the gateway is different: If you're using the hash of a specific snapshot of content, use the path `http://localhost:8080/ipfs/<your-ipfs-hash>`. If you're using an IPNS hash to get the *latest* version of some content, use the path `http://localhost:8080/ipns/<your-ipns-hash>`

To view the wikipedia page we're using as an example in all of the lessons in the Tutorial on Avenues for Access, use these links:

- 2017-04-30 snapshot: http://localhost:8080/ipfs/Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html
- latest (IPNS): http://localhost:8080/ipns/QmQP99yW82xNKPxXLroxj1rMYMGF6Grwjj2o4svsdmGh7S/wiki/Anasayfa.html
- latest (DNS): http://localhost:8080/ipns/wikipedia-on-ipfs.io

Explanation

This feature is experimental! Until we have tested this feature and removed the "experimental" designation, you should assume that the explanation here is aspirational and provisional. We are describing what *should* be true but we have not yet tested and confirmed that the approach works without leaking information.

When you configure an IPFS node to use the Tor transport, the node will pipe all of its peer-to-peer communications through the Tor onion network. This means that when you request content from your local node, whether through its http gateway at localhost:8080 or through the command line, the node will access the IPFS network over the tor transport protocol. When it connects with peer nodes on the IPFS network, the peers will not know which node they are talking to nor where it is.

Next Steps

Return to the Tutorial on Avenues for Access to learn about the many other ways you can use IPFS to access the same content using the same content-addressed link.

Lesson: Access IPFS content through a browser extension

This is a placeholder. There are currently four web browser extensions that help your retrieve content from IPFS. Each works in slightly different ways. We are in the process of consolidating that code and making it more secure before we encourage people to rely on it.

When the IPFS browser extension is complete, we will publish it on the app stores for all of the browsers that support it. When you download the extension, it will automatically recognize IPFS links and will use the IPFS peer-to-peer network to retrieve the content for you -- no HTTP gateway needed, nothing else to install on your computer, no need to use the command line. You will only have to install the browser extension and the whole IPFS network will become available to you.

We consider this the next big step to getting IPFS natively supported in web browsers. You can track this work in the github repository at https://github.com/ipfs/in-web-browsers. This comment on a github issue describes the state of these efforts as of April 2017.

Among other things, this support for IPFS in browsers will make it possible to start using links that are truly content-addressed, without any reference to HTTP locations, even when you access content through a web browser. We are advocating for this to be done using a new `dweb:` address scheme. Using the `dweb:` scheme, the links to the wikipedia page we're using as an example in all of the lessons in the Tutorial on Avenues for Access will look like this:

- 2017-04-30 snapshot: dweb:/ipfs/Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html
- latest (IPNS): dweb:/ipns/QmQP99yW82xNKPxXLroxj1rMYMGF6Grwjj2o4svsdmGh7S/wiki/Anasayfa.html
- latest (DNS): dweb:/ipns/wikipedia-on-ipfs.io

Next Steps

Return to the Tutorial on Avenues for Access to learn about the many other ways you can use IPFS to access the same content using the same content-addressed link.

Lesson: Sneakernets - moving the data on USB Drives and other Hardware

We're not kidding. If you'd like to move IPFS content between networks via sneakernet, IPFS is just fine with that. This lesson covers how to load IPFS content onto storage devices like USB drives so that you can physically move the content to new networks and then re-publish it. Without ever relying on a direct connection between the networks, this will make the IPFS links for your content valid on both sides of the air gap.

Examples of how this is useful:

- Make internet content available in places that don't have direct connections to the internet backbone (remote locations with limited connectivity, space stations)
- Circumvent censorship by governments, corporations and overbearing parents.

For this example we will pretend you're using an external drive to move a snapshot of wikipedia from IPFS to a new network where wikipedia isn't available.

Prerequisites

To do the steps in this lesson you must:

- Be familiar with using the command line
- Install and Initialize IPFS on your local machine

Goals

After doing this Lesson you will know how to physically move IPFS content across an airgap, making it available over IPFS and HTTP on the other side.

Steps

Step 1: Download the content you want to move, plus a copy of IPFS

Use `ipfs get` command o download the content you want to move across networks. In this example, we download a complete snapshot of Wikipedia archive to disk, saving it as a folder called `WikipediaSnapshot` *(Warning: this snapshot is 15 GB. You might want to use something smaller)*:

```
$ ipfs get Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html -o Wi
kipediaSnapshot
```

Download the latest IPFS binaries into your drive too. You will need this in order to publish the content on the other side. Make sure to download the appropriate go-ipfs binary for the computer you will be moving the data to. *Note: there might be a newer version of ipfs available when you read this. Find out the most current version number at https://dist.ipfs.io/#go-ipfs*

```
$ ipfs get /ipns/dist.ipfs.io/go-ipfs/v0.4.8 -o go-ipfs-v0.4.8
```

Step 2: Copy the files to your external drive

- Copy the snapshot (ie. the folder "WikipediaSnapshot") to your external drive
- Copy the IPFS binaries you downloaded (ie. the folder "go-ipfs-v0.4.8") into your external drive

Step 3: Move to the next computer

- Eject your external drive
- Physically carry your external drive to the next computer you want to use the information from.

Step 4: Install IPFS on the next computer and load the content

First Install and initialize the IPFS binary on the new computer. The instructions in the installation tutorial might help.

Then import the data into ipfs (in this case, the folder called `WikipediaSnapshot`)

```
$ ipfs add -r WikipediaSnapshot
```

Step 5: Confirm that the content is now available on the new computer

Start the ipfs daemon:

```
ipfs daemon
```

The snapshot link should now work: http://localhost:8080/ipfs/Qme2sLfe9ZMdiuWsEtajWMDzx6B7VbjzpSC2VWhtB6GoB1/wiki/Anasayfa.html

Explanation

This approach allows you to physically move IPFS content into networks where it was previously unavailable.

Because IPFS uses content-addressing, as long as the content you added to the second network is identical to the original content you originally exported from IPFS, the IPFS identifier for your content will be identical in both networks.

Next Steps

Return to the Tutorial on Avenues for Access to learn about the many other ways you can use IPFS to access the same content using the same content-addressed link.

Tutorial: Making Changes on the Permanent Web

These Lessons are tested with go-ipfs version 0.4.8. *Please update this file on github to reflect any other versions that have been tested.*

Prerequisites

Learning Objectives

After doing this Lesson you will be able to

- Update Files on IPFS using IPNS
- Explain how version histories of files appear on IPFS
- Explain why IPFS is called the Permanent Web
- Track and Publish an entire website as it changes over time using IPFS and IPNS

Key Concepts

- IPNS
- The Permanent Web
- Using hashes to uniquely identify versions of content

Conceptual Framework

Normally, updating content means replacing a file - for instance, if I update a blog post, then people will see the edited file, and not the new one. However, with IPFS, both verisons of the file will be accessible in the network. It's not a matter of replacing: you add the new one, too. This raises the question: how do we actually update our links, so that people will see the new version of a file? They can't go to the file's location, because IPFS locates files by looking for their hashes (that's what content-addressed means). So, you need to have a way of pointing people to the new hash easily.

The trick is to add new the content, and then update a pointer to that content. So, there needs to be a way of having a mutable pointer.

This is where IPNS comes in, the InterPlanetary Naming System (Name Service?). IPNS is a simple service that uses your peer ID to point to a particular hash. This hash can change, but your peer ID doesn't. That means that you can point to content in IPFS that may also change, and people can still access it without needing to know the new hash before hand.

Author Question: Does IPNS point to a constant hash that is in your config, or does it just use your peerId? Does your IPNS hash ever change?

Lessons

1. If you have not already done it, follow the Lesson to **Add a file to IPFS**
2. Set up IPNS on your node
3. Create an IPNS entry that points to your file
4. Modify your File and add the modified version to IPFS
5. Update the IPNS entry to point to the new version Advanced:
6. add multiple files (ie. an entire website) to IPFS
7. Use IPNS to link to the entire website, or any file in the website

Notes

From IRC

```
??. Use the files API.

15:14:11 <•Kubuxu> you are limited to one IPNS per node
15:14:18 <•Kubuxu> but you can have directories in IPNS
15:14:26 <•Kubuxu> combine it with files API
15:15:19 <•Kubuxu> ipfs files mkdir /public
15:15:46 <•Kubuxu> ipfs files cp /ipfs/$HASH_OF_PAPER /public/my-faviourite-paper.
pdf
15:16:16 <•Kubuxu> ipfs name publish $(ipfs files stat --hash /public)
15:16:41 ⇐ jaboja quit (~jaboja@2a00:f41:3875:fd4b:de85:deff:fe55:967a) Ping timeo
ut: 264 seconds
15:16:42 <•Kubuxu> you can reference your paper with: /ipns/$PEERID/my-faviourite-
paper.pdf
15:16:50 <•Kubuxu> and you can have many more files in there
```

Tutorial: Merkle Trees and the IPFS DAG

Concepts

These Lessons introduce the following concepts:

- Cryptographic Hashes and Content Addressability
- Authenticated Graphs
- Turning Files into Trees
- Turning any Data into Trees
- Publishing hashes on the DHT
- Getting data from the Peer to Peer Network

Prerequisites

Lessons

1. Lesson: Turn a file into a tree of hashes
2. Lesson: Create a cryptographic hash
3. Lesson: Build a tree of data in IPFS using cryptographic hashes to link the pieces (a Merkle DAG)
4. Lesson: Explore the types of software that use hash trees to track data

Lesson: Turn a File into a Merkle Tree

Work in Progress This is the content from this existing Lesson vaguely re-framed to fit the Lesson framework.

Goals

- Explain how IPFS represents Files as Merkle trees
- Explore the Merkle Tree Blocks that make up a File in IPFS

Steps

Step 1: Download the sample file and add it to IPFS

For this lesson we need a file that's larger than 256kb. Download this image: tree-in-cosmos.jpg (863kb)

Save it as "tree-in-cosmos.jpg" and then add it to IPFS

```
$ ipfs add tree-in-cosmos.jpg
added QmWNj1pTSjbauDHpdyg5HQ26vYcNWnubg1JehmwAE9NnU9
```

Step 2:

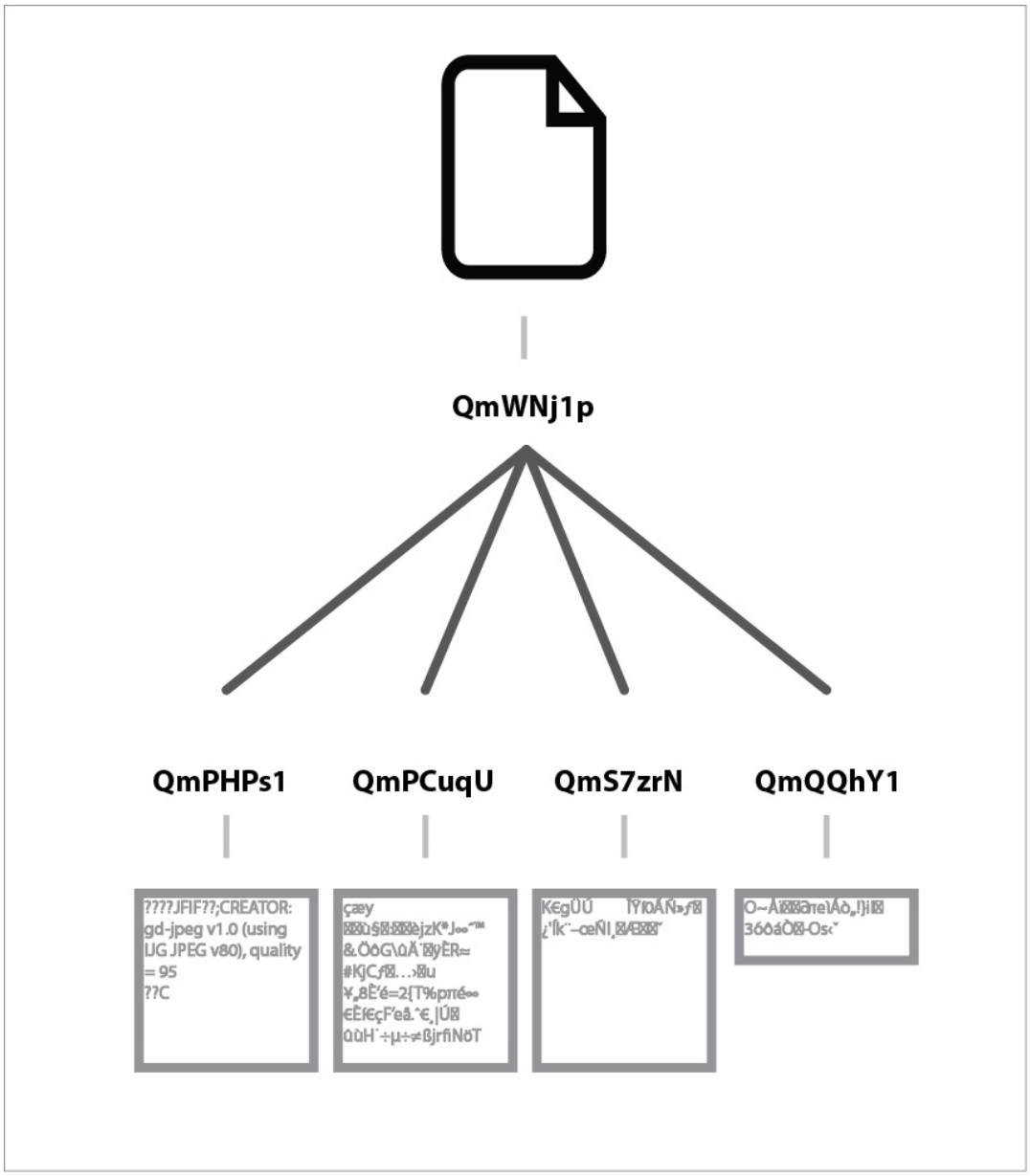
Let's look at how IPFS represented that file internally by passing the content's hash to the `ipfs ls` command:

```
ipfs ls -v QmWNj1pTSjbauDHpdyg5HQ26vYcNWnubg1JehmwAE9NnU9
Hash                                           Size   Name
QmPHPs1P3JaWi53q5qqiNauPhiTqa3S1mbszcVPHKGNWRh 262158
QmPCuqUTNb21VDqtp5b8VsNzKEMtUsZCCVsEUBrjhERRSR 262158
QmS7zrNSHEt5GpcaKrwdbnv1nckBreUxWnLaV4qivjaNr3 262158
QmQQhY1syuqo9Sq6wLFAupHBEeqfB8jNnzYUSgZGARJrYa 76151
```

This returned a bunch of hashes. That's different from what happened in the lesson on adding file content to ipfs, where you only got one hash back. This is because ipfs breaks files into content *blocks* that are each about 256kb and then uses a **hash tree** to represent how they fit together.

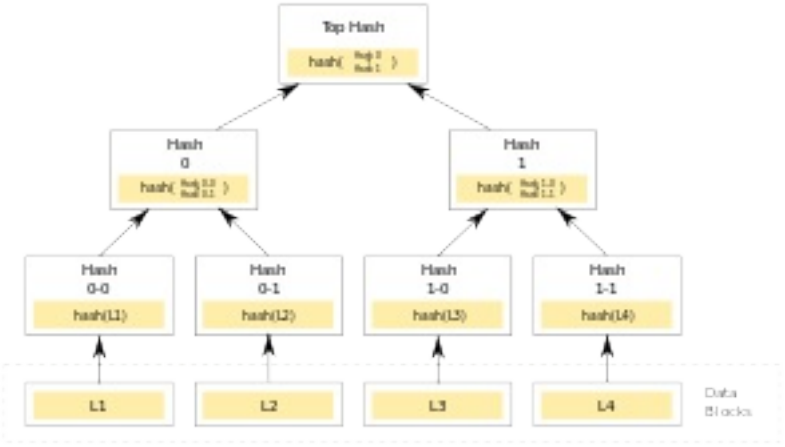
This is one example of how IPFS uses hash trees, also known as **Merkle DAGs**, to represent information.

> This kind of hash tree is formally known as a **Merkle DAG** -- this is because the technical term for this type of data structure is a *Directed Acyclic Graph*, or *DAG*, and a mathematician named Ralph *Merkle* invented them. Hence: Merkle DAG, or merkledag.



In this case, the hash for our file `QmWNj1pTS...` is the hash of the **root block** in a DAG that contains 4 sub-blocks. The output from `ipfs ls` lists those sub-blocks and their size.

Sometimes sub-blocks have sub-blocks of their own. That's when a Merkle DAG starts looking like a tree. This diagram shows a Merkle DAG with three layers of sub-blocks.:



Do you think it looks like an upside-down tree?

Step 3: Explore The Hash Tree

The `ipfs refs` and `ipfs object links` commands are other ways to get the listing of sub-blocks in the tree.

Try these:

```
$ ipfs refs QmWNj1pTSjbauDHpdyg5HQ26vYcNWnubg1JehmwAE9NnU9
ipfs object links -v QmWNj1pTSjbauDHpdyg5HQ26vYcNWnubg1JehmwAE9NnU9
```

If the sub-blocks had more sub-blocks within them, you would be able to use these commands to get the hashes of those sub-sub-blocks. For example:

```
$ ipfs object links -v QmPHPs1P3JaWi53q5qqiNauPhiTqa3S1mbszcVPHKGNWRh
```

But this doesn't return anything because there aren't sub-blocks within `QmPHPs1P...`

Step 4: Read the content back out of IPFS

If you use `ipfs cat` to read the content back out of ipfs, it handles re-assembling the file from the hash tree. For example, the following command will read our sample image out of ipfs and write the content into a new file called "copy-of-tree-in-cosmos.jpg". Run the command and then open the new file to confirm that the image is still intact.

```
$ ipfs cat QmWNj1pTSjbauDHpdyg5HQ26vYcNWnubg1JehmwAE9NnU9 > copy-of-tree-in-cosmos
.jpg
```

Step 5: Examine the blocks individually

Use these commands to examine the blocks in the hash tree:

- `ipfs block stat` will tell you the exact size of a given block (without its children)
- `ipfs refs` will tell you all the children of that block. This is a more suitable command for scripting something to run on each child block of a given object.
- `ipfs ls` or `ipfs object links` will show you all children and their sizes.

Step 6: Read the Contents of a sub-block

In some cases you want to retrieve sub-blocks from a tree. You can use `ipfs cat` to do that. You can test that with the sub-bocks from our image.

```
$ ipfs cat QmPHPs1P3JaWi53q5qqiNauPhiTqa3S1mbszcVPHKGNWRh
```

The output will look similar to this because it's image content, not text:

```
<FF><D8><FF><E0>^@^PJFIF^@^A^A^@^@^A^@^A^@^@<FF><FE>^@;CREATOR: gd-jpeg v1.0 (usin
g IJG JPEG v80), quality = 95
<FF><DB>^@C^@^B^A^A^A^A^A^B^A^A^A^B^B^B^B^B^D^C^B^B^B^B^E^D^D^C^D^F^E^F^F^F^E^F^F^
F^G  ^F^G     ^G^F^F^H^K^H


???w!1AQaq"2B????    #3R?br?
$4?%?&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz??????????????????????????????????
????????????????????????????????????????
,?y?▫|ŵiMg▫k?_?WN??W????F!%T 8?W???I$uZ?????K?\?olng?H|?????@?#8?En?1??;
```

Step 7: Assemble the Pieces Manually

`ipfs cat` allows you read the contents of each block and it also allows you to *concatenate* many inputs. This means we can use `ipfs cat` to re-build our image by passing the hashes of all our sub-blocks into that command.

```
ipfs cat QmPHPs1P3JaWi53q5qqiNauPhiTqa3S1mbszcVPHKGNWRh QmPCuqUTNb21VDqtp5b8VsNzKE
MtUsZCCVsEUBrjhERRSR QmS7zrNSHEt5GpcaKrwdbnv1nckBreUxWnLaV4qivjaNr3 QmQQhY1syuqo9S
q6wLFAupHBEeqfB8jNnzYUSgZGARJrYa > manually-rebuilt-tree-in-cosmos.jpg
```

Bonus Steps

Some things to try:

- Write a script that uses `ipfs refs` and `ipfs cat` to rebuild a file from its root hash

Explanation

Merkle DAGs are the core concept of IPFS. Merkle DAGs are also at the core of technologies like git, bitcoin and dat.

Hash trees are made up of *content blocks* that are each identified by their cryptographic hash. You can reference any of these blocks using its hash, which allows you to build trees of blocks that reference their "sub blocks" using the hashes of those sub blocks.

The `ipfs add` command will create a Merkle DAG out of the data in the files you specify. It follows the unixfs data format when doing this. What this means is that your files are broken down into blocks, and then arranged in a tree-like structure using 'link nodes' to tie them together. A given file's 'hash' is actually the hash of the root (uppermost) node in the DAG. for a given DAG, you can easily view the sub-blocks under it with `ipfs ls`.

Next Steps

Next, use IPFS to build your own Merkle DAG from scratch

Lesson: Creating a Merkle Tree from Scratch

Work in Progress This is the content from this existing Lesson vaguely re-framed to fit the Lesson framework.

Goals

- Build a tree of data in IPFS using cryptographic hashes to link the pieces (a Merkle DAG)

Explanation: Blocks vs Objects

In ipfs, a block refers to a single unit of data, identified by its key (hash). a block can be any sort of data, and does not necessarily have any sort of format associated with it. an object, on the other hand, refers to a block that follows the merkledag protobuf data format. it can be parsed and manipulated via the `ipfs object` command. any given hash may represent an object or a block.

Steps

Step 1

Creating your own blocks is easy! simply put your data in a file and run `ipfs block put <yourfile>` on it, or you can pipe your filedata into `ipfs block put`, like so:

Step 2

```
$ echo "This is some data" | ipfs block put
QmfQ5QAjvg4GtA3wg3adpnDJug8ktA1BxurVqBD8rtgVjM
$ ipfs block get QmfQ5QAjvg4GtA3wg3adpnDJug8ktA1BxurVqBD8rtgVjM
This is some data
```

Note: When making your own block data, you wont be able to read the data with `ipfs cat`, this is because you are inputting raw data without the unixfs data format. To read raw blocks use `ipfs block get` as shown in the example.

Tutorial: Dynamic Content on IPFS

Concepts

These Lessons introduce the following concepts:

- Immutability: "Changes" as *additions* to the tree
- CRDTs
- Pubsub
- Authenticated Streams (with pubsub)

Prerequisites

Lessons

1. Disclaimer: Dynamic content on IPFS is a Work in Progress
2. Lesson: Add data to the DAG (locally)
3. Lesson: Tell peers about your Changes
4. Lesson: Use hashes to get someone's changes from IPFS
5. Lesson: Use a pub/sub strategy to pass around messages about changes
6. Lesson: Resolve conflicts with a merge strategy (CRDTs)