



Files & File Systems: Crash Course Computer Science #20

Crash Course: Computer Science

<https://youtube.com/watch?v=KN8YgJnShPM>

<https://nerdfighteria.info/v/KN8YgJnShPM>

Hi, I'm Carrie Anne, and welcome to Crash Course Computer Science!

Last episode we talked about data storage, how technologies like magnetic tape and hard disks can store trillions of bits of data, for long durations, even without power. Which is perfect for recording "big blobs" of related data, what are more commonly called computer files.

You've no doubt encountered many types, like text files, music files, photos and videos. Today, we're going to talk about how files work, and how computers keep them all organized with File Systems.

[intro]

It's perfectly legal for a file to contain arbitrary, unformatted data, but it's most useful and practical if the data inside the file is organized somehow.

This is called a file format. You can invent your own, and programmers do that from time to time, but it's usually best and easiest to use an existing standard, like JPEG and MP3. Let's look at some simple file formats.

The most straightforward are text files, also known as T-X-T files, which contain, surprise, text. Like all computer files, this is just a huge list of numbers, stored as binary. If we look at the raw values of a T-X-T file in storage, it would look something like this: We can view this as decimal numbers instead of binary, but that still doesn't help us read the text.

The key to interpreting this data is knowing that T-X-T files use ASCII, a character encoding standard we discussed way back in Episode 4. So, in ASCII, our first value, 72, maps to the capital letter H. And in this way, we decode the whole file.

Let's look at a more complicated example: a WAVE File – also called a WAV – which stores audio. Before we can correctly read the data, we need to know some information, like the bit rate and whether it's a single track or stereo. Data, about data, is called metadata.

This metadata is stored at the front of the file, ahead of any actual data, in what's known as a header. Here's what the first 44 bytes of a WAV file looks like. Some parts are always the same, like where it spells out W-A-V-E.

Other parts contain numbers that change depending on the data contained within. The audio data comes right behind the metadata, and it's stored as a long list of numbers. These values represent the amplitude of sound captured many times per second, and if you want a primer on sound, check out my video all about it in Crash Course Physics. Link in the dooblydoo.

As an example, let's look at a waveform of me saying: "hello!" Hello! Now that we've captured some sound, let's zoom into a little snippet.

A digital microphone, like the one in your computer or smartphone, samples the sound pressure thousands of times. Each sample can be represented as a number. Larger numbers mean higher sound pressure, what's called amplitude.

And these numbers are exactly what gets stored in a WAVE file! Thousands of amplitudes for every single second of audio! When it's time to play this file, an audio program needs to actuate the computer's speakers such that the original waveform is emitted. "Hello!" So, now that you're getting the hang of file formats, let's

talk about bitmaps or B-M-Ps, which store pictures.

On a computer, PICtures are made up of little tiny square ELelements called pixels. Each pixel is a combination of three colors: red, green and blue. These are called additive primary colors, and they can be mixed together to create any other color on our electronic displays.

Now, just like WAV files, bitmaps start with metadata, including key values like image width, image height, and color depth. As an example, let's say the metadata specified an image 4 pixels wide, by 4 pixels tall, with a 24-bit color depth - that's 8-bits for red, 8-bits for green, and 8-bits for blue. As a reminder, 8 bits is the same as one byte.

The smallest number a byte can store is 0, and the largest is 255. Our image data is going to look something like this: Let's look at the color of our first pixel. It has 255 for its red value, 255 for green and 255 for blue.

This equates to full intensity red, full intensity green and full intensity blue. These colors blend together on your computer monitor to become white. So our first pixel is white!

The next pixel has a Red-Green-Blue, or RGB value of 255, 255, 0. That's the color yellow! The pixel after that has a RGB value of 0,0,0 - that's zero intensity everything, which is black.

And the next one is yellow. Because the metadata specified this was a 4 by 4 image, we know that we've reached the end of our first row of pixels. So, we need to drop down a row.

The next RGB value is 255,255,0 – yellow again. Okay, let's go ahead and read all the pixels in our 4x4 image... tada! A very low resolution pac-man!

Obviously this is a simple example of a small image, but we could just as easily store this image in a bitmap. I want to emphasize again that it doesn't matter if it's a TXT file, WAV, BMP, or fancier formats we don't have time to discuss, like ZIPs and PPTs. Under the hood, they're all the same: long lists of numbers, stored as binary, on a storage device. File formats are the key to reading and understanding the data inside.

Now that you understand files a little better, let's move on to how computers go about storing them. Even though the underlying storage medium might be a strip of tape, a drum, a disk, or integrated circuits, hardware and software abstractions let us think of storage as a long line of little buckets that store values.

In the early days, when computers only performed one computation, like calculating artillery range tables, the entire storage operated like one big file. Data started at the beginning of storage, and then filled it up in order as output was produced, up to the storage capacity. However, as computational power and storage capacity improved, it became possible, and useful, to store more than one file at a time.

The simplest option is to store files back-to-back. This can work... but how does the computer know where files begin and end? Storage devices have no notion of files – they're just a mechanism for storing lots of bits.

So, for this to work, we need to have a special file that records where other ones are located. This goes by many names, but a good general term is Directory File. Most often, it's kept right at the front of storage, so we always know where to access it. Location zero!

Inside the Directory File are the names of all the other files in



Files & File Systems: Crash Course Computer Science #20

Crash Course: Computer Science

<https://youtube.com/watch?v=KN8YgJnShPM>

<https://nerdfighteria.info/v/KN8YgJnShPM>

storage. In our example, they each have a name, followed by a period, and end with what's called a File Extension, like "BMP" or "WAV". Those further assist programs in identifying file types.

The Directory File also stores metadata about these files, like when they were created and last modified, who the owner is, and if it can be read, written or both. But most importantly, the directory file contains where these files begin in storage, and how long they are.

If we want to add a file, remove a file, change a filename, or similar, we have to update the information in the Directory File. It's like the Table of Contents in a book, if you make a chapter shorter, or move it somewhere else, you have to update the table of contents, otherwise the page numbers won't match! The Directory File, and the maintenance of it, is an example of a very basic File System, the part of an Operating System that manages and keep track of stored files.

This particular example is called a Flat File System, because they're all stored at one level. It's flat! Of course, packing files together, back-to-back, is a bit of a problem, because if we want to add some data to let's say "todo.txt", there's no room to do it without overwriting part of "carrie.bmp".

So modern File Systems do two things. First, they store files in blocks. This leaves a little extra space for changes, called slack space.

It also means that all file data is aligned to a common size, which simplifies management. In a scheme like this, our Directory File needs to keep track of what block each one is stored in. The second thing File Systems do, is allow files to be broken up into chunks and stored across many blocks.

So let's say we open "todo.txt", and we add a few more items then the file becomes too big to be saved in its one block. We don't want to overwrite the neighboring one, so instead, the File System allocates an unused block, which can accommodate extra data. With a File System scheme like this, the Directory File needs to store not just one block per file, but rather a list of blocks per file.

In this way, we can have files of variable sizes that can be easily expanded and shrunk, simply by allocating and deallocating blocks. If you watched our episode on Operating Systems, this should sound a lot like Virtual Memory. Conceptually it's very similar!

Now let's say we want to delete "carrie.bmp". To do that, we can simply remove the entry from the Directory File. This, in turn, causes one block to become free.

Note that we didn't actually erase the file's data in storage, we just deleted the record of it. At some point, that block will be overwritten with new data, but until then, it just sits there. This is one way that computer forensic teams can "recover" data from computers even though people think it has been deleted. Crafty!

Ok, let's say we add even more items to our todo list, which causes the File System to allocate yet another block to the file, in this case, recycling the block freed from carrie.bmp. Now our "todo.txt" is stored across 3 blocks, spaced apart, and also out of order.

Files getting broken up across storage like this is called fragmentation. It's the inevitable byproduct of files being created, deleted and modified. For many storage technologies, this is bad news.

On magnetic tape, reading todo.txt into memory would require seeking to block 1, then fast forwarding to block 5, and then rewinding to block 3 – that's a lot of back and forth! In real world

File Systems, large files might be stored across hundreds of blocks, and you don't want to have to wait five minutes for your files to open. The answer is defragmentation!

That might sound like techno-babble, but the process is really simple, and once upon a time it was really fun to watch! The computer copies around data so that files have blocks located together in storage and in the right order. After we've defragged, we can read our todo file, now located in blocks 1 through 3, in a single, quick read pass.

So far, we've only been talking about Flat File Systems, where they're all stored in one directory. This worked ok when computers only had a little bit of storage, and you might only have a dozen or so files. But as storage capacity exploded, like we discussed last episode, so did the number of files on computers.

Very quickly, it became impractical to store all files together at one level. Just like documents in the real world, it's handy to store related files together in folders. Then we can put connected folders into folders, and so on.

This is a Hierarchical File System, and it's what your computer uses. There are a variety of ways to implement this, but let's stick with the File System example we've been using to convey the main idea. The biggest change is that our Directory File needs to be able to point not just to files, but also other directories.

To keep track of what's a file and what's a directory, we need some extra metadata. This Directory File is the top-most one, known as the Root Directory. All other files and folders lie beneath this directory along various file paths.

We can see inside of our "Root" Directory File that we have 3 files and 2 subdirectories: music and photos. If we want to see what's stored in our music directory, we have to go to that block and read the Directory File located there; the format is the same as our root directory. There's a lot of great songs in there!

In addition to being able to create hierarchies of unlimited depth, this method also allows us to easily move around files. So, if we wanted to move "theme.wav" from our root directory to the music directory, we don't have to re-arrange any blocks of data. We can simply modify the two Directory Files, removing an entry from one and adding it to another.

Importantly, the theme.wav file stays in block 5. So that's a quick overview of the key principles of File Systems. They provide yet another way to move up a new level of abstraction.

File systems allow us to hide the raw bits stored on magnetic tape, spinning disks and the like, and they let us think of data as neatly organized and easily accessible files. We even started talking about users, not programmers, manipulating data, like opening files and organizing them, foreshadowing where the series will be going in a few episodes. I'll see you next week.