How File Compression Works

by Tom Harris

If you download many programs and files off the Internet, you've probably encountered ZIP files before. This compression system is a very handy invention, especially for Web users, because it lets you reduce the overall number of bits and bytes in a file so it can be transmitted faster over slower Internet connections, or take up less space on a disk. Once you download the file, your computer uses a program such as WinZip or Stuffit to expand the file back to its original size. If everything works correctly, the expanded file is identical to the original file before it was compressed.

At first glance, this seems very mysterious. How can you reduce the number of bits and bytes and then add those exact bits and bytes back later? As it turns out, the basic idea behind the process is fairly straightforward. In this edition of **HowStuffWorks**, we'll examine this simple method as we take a very small file through the basic process of compression.

Finding Redundancy

Most types of computer files are fairly redundant -- they have the same information listed over and over again. File-compression programs simply get rid of the redundancy. Instead of listing a piece of information over and over again, a file-compression program lists that information once and then refers back to it whenever it appears in the original program.

As an example, let's look at a type of information we're all familiar with: words.

In John F. Kennedy's 1961 inaugural address, he delivered this famous line:

> *"Ask not what your country can do for you -- ask what you can do for your country."*

The quote has 17 words, made up of 61 letters, 16 spaces, one dash and one period. If each letter, space or punctuation mark takes up one unit of memory, we get a total file size of 79 units. To get the file size down, we need to look for redundancies.

Immediately, we notice that:

- **"ask" appears two times**
- **"what" appears two times**
- **"your" appears two times**
- **"country" appears two times**
- **"can" appears two times**
- **"do" appears two times**
- **"for" appears two times**
- **"you" appears two times**

Ignoring the difference between capital and lower-case letters, roughly half of the phrase is redundant. Nine words -- ask, not, what, your, country, can, do, for, you -- give us almost everything we need for the entire quote. To construct the second half of the phrase, we just point to the words in the first half and fill in the spaces and punctuation. In the next section, we'll see how file-compression systems accomplish this.

Looking it Up

Most compression programs use a variation of the **LZ adaptive dictionary-based algorithm** to shrink files. "LZ" refers to **Lempel and Ziv**, the algorithm's creators, and "dictionary" refers to the method of **cataloging** pieces of data.

The system for arranging dictionaries varies, but it could be as simple as a numbered list. When

we go through Kennedy's famous words, we pick out the words that are repeated and put them into the numbered index. Then, we simply write the number instead of writing out the whole word.

So, if this is our dictionary:

1. **ask**
2. **what**
3. **your**
4. **country**
5. **can**
6. **do**
7. **for**
8. **you**

Our sentence now reads:

"1 not 2 3 4 5 6 7 8 -- 1 2 8 5 6 7 3 4"

If you knew the system, you could easily reconstruct the original phrase using only this dictionary and number pattern. This is what the expansion program on your computer does when it expands a downloaded file. You might also have encountered compressed files that open themselves up. To create this sort of file, the programmer includes a simple expansion program with the compressed file. It automatically reconstructs the original file once it's downloaded.

But how much space have we actually saved with this system? "1 not 2 3 4 5 6 7 8 -- 1 2 8 5 6 7 3 4" is certainly shorter than "Ask not what your country can do for you; ask what you can do for your country;" but keep in mind that we need to **save the dictionary itself along with the file**.

In an actual compression scheme, figuring out the various file requirements would be fairly complicated; but for our purposes, let's go back to the idea that every character and every space takes up one unit of memory. We already saw that the full phrase takes up 79 units. Our compressed sentence (including spaces) takes up 37 units, and the dictionary (words and numbers) also takes up 37 units. This gives us a file size of 74, so we haven't reduced the file size by very much.

But this is only one sentence! You can imagine that if the compression program worked through the rest of Kennedy's speech, it would find these words and others repeated many more times. And, as we'll see in the next section, it would also be rewriting the dictionary to get the most efficient organization possible.

Searching for Patterns

In our example, we picked out all the repeated words and put those in a dictionary. To us, this is the most obvious way to write a dictionary. But a compression program sees it quite differently: It doesn't have any concept of separate words -- it only looks for patterns. And in order to reduce the file size as much as possible, it carefully selects which patterns to include in the dictionary.

If we approach the phrase from this perspective, we end up with a completely different dictionary.

If the compression program scanned Kennedy's phrase, the first redundancy it would come across would be only a couple of letters long. In "ask not what your," there is a repeated pattern

of the letter "t" followed by a space -- in "not" and "what." If the compression program wrote this to the dictionary, it could write a "1" every time a "t" were followed by a space. But in this short phrase, this pattern doesn't occur enough to make it a worthwhile entry, so the program would eventually overwrite it.

The next thing the program might notice is "ou," which appears in both "your" and "country." If this were a longer document, writing this pattern to the dictionary could save a lot of space -- "ou" is a fairly common combination in the English language. But as the compression program worked through this sentence, it would quickly discover a better choice for a dictionary entry: Not only is "ou" repeated, but the entire words "your" and "country" are both repeated, and they are actually repeated together, as the phrase "your country." In this case, the program would overwrite the dictionary entry for "ou" with the entry for "your country."

The phrase "can do for" is also repeated, one time followed by "your" and one time followed by "you," giving us a repeated pattern of "can do for you." This lets us write 15 characters (including spaces) with one number value, while "your country" only lets us write 13 characters (with spaces) with one number value, so the program would overwrite the "your country" entry as just "r country," and then write a separate entry for "can do for you." The program proceeds in this way, picking up all repeated bits of information and then calculating which patterns it should write to the dictionary. This ability to rewrite the dictionary is the "adaptive" part of **LZ adaptive dictionary-based algorithm**. The way a program actually does this is fairly complicated, as you can see by the discussions on Data-Compression.com.

No matter what specific method you use, this in-depth searching system lets you compress the file much more efficiently than you could by just picking out words. Using the patterns we picked out above, and adding "__" for spaces, we come up with this larger dictionary:

1. **ask__**
2. **what__**
3. **you**
4. **r__country**
5. **__can__do__for__you**

And this smaller sentence:

"1not__2345__--__12354"

The sentence now takes up 18 units of memory, and our dictionary takes up 41 units. So we've compressed the total file size from 79 units to 59 units! This is just one way of compressing the phrase, and not necessarily the most efficient one. (See if you can find a better way!) In the next section, we'll see some of the ways in which compression percentage might vary.

How Much Can You Trim?

So how good is this system? The **file-reduction ratio** depends on a number of factors, including file type, file size and compression scheme.

In most languages of the world, certain letters and words often appear together in the same pattern. Because of this high rate of redundancy, **text files** compress very well. A reduction of 50 percent or more is typical for a good-sized text file. Most **programming languages** are also very redundant because they use a relatively small collection of commands, which frequently go

together in a set pattern. Files that include a lot of unique information, such as graphics or MP3 files, cannot be compressed much with this system because they don't repeat many patterns (more on this in the next section).

If a file has a lot of repeated patterns, the rate of reduction typically increases with file size. You can see this just by looking at our example -- if we had more of Kennedy's speech, we would be able to refer to the patterns in our dictionary more often, and so get more out of each entry's file space. Also, more pervasive patterns might emerge in the longer work, allowing us to create a more efficient dictionary.

This efficiency also depends on the specific algorithm used by the compression program. Some programs are particularly suited to picking up patterns in certain types of files, and so may compress them more succinctly. Others have dictionaries within dictionaries, which might compress efficiently for larger files but not for smaller ones. While all compression programs of this sort work with the same basic idea, there is actually a good deal of variation in the manner of execution. Programmers are always trying to build a better system.

Lossy and Lossless

The type of compression we've been discussing here is called **lossless compression**, because it lets you recreate the original file exactly. All lossless compression is based on the idea of breaking a file into a "smaller" form for transmission or storage and then putting it back together on the other end so it can be used again.

Lossy compression works very differently. These programs simply eliminate "unnecessary" bits of information, tailoring the file so that it is smaller. This type of compression is used a lot for reducing the file size of bitmap pictures, which tend to be fairly bulky. To see how this works, let's consider how your computer might compress a scanned photograph.

A lossless compression program can't do much with this type of file. While large parts of the picture may look the same -- the whole sky is blue, for example -- most of the individual pixels are a little bit different. To make this picture smaller without compromising the resolution, you have to change the color value for certain pixels. If the picture had a lot of blue sky, the program would pick one color of blue that could be used for every pixel. Then, the program rewrites the file so that the value for every sky pixel refers back to this information. If the compression scheme works well, you won't notice the change, but the file size will be significantly reduced.

Of course, with lossy compression, you can't get the original file back after it has been compressed. You're stuck with the compression program's reinterpretation of the original. For this reason, you can't use this sort of compression for anything that needs to be reproduced exactly, including software applications, databases and presidential inauguration speeches.