Downloading files with curl

How to download files straight from the command-line interface

The `curl` tool lets us fetch a given URL from the command-line. Sometimes we want to save a web file to our own computer. Other times we might pipe it directly into another program. Either way, `curl` has us covered.

See its documentation here.

This is the basic usage of `curl` :

```
curl http://some.url --output some.file
```

That `--output` flag denotes the filename (`some.file`) of the downloaded URL (`http://some.url`)

Let's try it with a basic website address:

```
curl http://example.com --output my.file
```

Besides the display of a progress indicator (which I explain below), you don't have much indication of what `curl` actually downloaded. So let's confirm that a file named `my.file` was actually downloaded.

Using the `ls` command will show the contents of the directory:

```
ls
```

Which outputs:

```
my.file
```

And if you use `cat` to output the contents of `my.file`, like so:

```
cat my.file
```

– you will the HTML that powers `http://example.com`

I thought Unix was supposed to be quiet?

Let's back up a bit: when you first ran the `curl` command, you might have seen a quick blip of a progress indicator:

```
  % Total    % Received % Xferd  Average Speed
Time    Time
                                 Dload  Upload
Total   Spent
100  1270  100  1270    0     0  50573      0 -
-:--:-- --:--:--
```

If you remember the Basics of the Unix Philosophy, one of the tenets is:

> Rule of Silence: When a program has nothing surprising to say, it should say nothing.

In the example of `curl`, the author apparently believes that it's important to tell the user the progress of the download. For a very small file, that status display is not terribly helpful. Let's try it with a bigger file (this is the baby names file from the Social Security Administration) to see how the progress indicator animates:

```
curl http://stash.compciv.org/ssa_baby_names/nam
es.zip  \
  --output babynames.zip
```

Quick note: If you're new to the command-line, you're probably used to commands executing every time you hit **Enter**. In this case, the command is so long (because of the URL) that I broke it down into two lines with the use of the **backslash**, i.e. `\`

This is *solely to make it easier for you to read*. As far as the computer cares, it just joins the two lines together as if that backslash weren't there and runs it as one command.

Make curl silent

The `curl` progress indicator is a nice affordance, but let's just see if we get `curl` to act like all of our Unix tools. In `curl`'s documentation of options, there is an option for silence:

> `-s, --silent`
>
> Silent or quiet mode. Don't show progress meter or error messages. Makes Curl mute. It will still output the data you ask for, potentially even to the terminal/stdout unless you redirect it.

Try it out:

```
curl http://example.com --output my.file --silen
t
```

Repeat and break things

So those are the basics for the `curl` command. There are many, many more options, but for now, we know how to use `curl` to do something that is actually quite powerful: fetch a file, anywhere on the Internet, from the simple confines of our command-line.

Before we go further, though, let's look at the various ways this simple command can be re-written and, more crucially, screwed up:

Shortened options

As you might have noticed in the `--silent` documentation, it lists the alternative form of `-s`. Many options for many tools have a shortened alias. In fact, `--output` can be shortened to `-o`

```
curl http://example.com -o my.file -s
```

Now watch out: the number of hyphens is **not** something you can mess up on; the following commands would cause an error or other unexpected behavior:

```
curl http://example.com -o my.file -silent

curl http://example.com -output my.file -s

curl http://example.com --o my.file --s
```

Also, mind the position of `my.file`, which can be thought of as the **argument** to the `-o` **option**.
The **argument** *must*follow after the `-o` ...because `curl`.

If you instead executed this:

```
curl http://example.com -o -s my.file
```

How would `curl` know that `my.file`, and not `-s` is the **argument**, i.e. what you want to name the content of the downloaded URL?

In fact, you might see that you've created a file named `-s` ...which is not the end of the world, but not something you want to happen unwittingly.

Order of options

By and large (from what I can think of at the top of my head), the *order* of the options doesn't matter:

```
curl http://example.com -s -o my.file
```

In fact, the URL, `http://example.com`, can be placed anywhere in the mix:

```
curl -s http://example.com -o my.file

curl -s -o my.file http://example.com
```

A couple of things to note:

1. The way that the URL, what you might consider the main **argument** for the `curl` command, can be placed anywhere after the command is *not* the way that *all* commands have been designed. So it always pays to read the documentation with every new command.
2. Notice how `-s http://example.com` doesn't cause a problem. That's because the `-s` option doesn't take an argument. But try the following:

```
curl -s -o http://example.com my.file
```

And you *will* have a problem.

No options at all

The last thing to consider is what happens when you just `curl` for a URL with no options (which, after all, should be *optional*). Before you try it, think about another part of the Unix philosophy:

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. **Write programs to handle text streams, because that is a universal interface**.

If you `curl` without any options except for the URL, the content of the URL (whether it's a webpage, or a binary file, such as an image or a zip file) will be printed out to screen. Try it:

```
curl http://example.com
```

Output:

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="tex
t/html; charset=utf-8" />
    <meta name="viewport" content="width=device-
width, initial-scale=1" />

... (and so forth)
```

Even with the small amount of HTML code that makes up the http://example.com webpage, it's too much for human eyes to process (and reading raw HTML wasn't meant for humans).

Standard output and connecting programs

But what if we wanted to send the contents of a web file to *another* program? Maybe to `wc`, which is used to count words and lines? Then we can use the powerful Unix feature of **pipes**. In this example, I'm using `curl`'s silent option so that only the output of `wc` (and not the progress indicator) is seen. Also, I'm using the `-l` option for `wc` to just get the number of *lines* in the HTML for example.com:

```
curl -s http://example.com | wc -l
```

Number of lines in `example.com` is: `50`

Now, you could've also done the same in two lines:

```
curl -s http://example.com -o temp.file
wc -l temp.file
```

But not only is that less *elegant*, it also requires *creating* a new file called `temp.file`. Now, this is a trivial concern, but someday, you may work with systems and data flows in which temporarily saving a file is not an available luxury (think of massive files).