Lesson | Wednesday

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Further Exploration: Aliases

Text

Sometimes we'll need to use the command line for the same command over and over again. This can get annoying, especially when we are typing in a very long command. While we can't do away with typing altogether, we can make our lives easier by using **aliases**. An alias is simply a shortcut, allowing us to reference a longer command via a short command of our choosing.

Note: You aren't expected to have or use aliases for an independent project but it can make your development experience more pleasant — and some students will encounter recommended aliases for certain commands in future lessons.

Creating an Alias

Let's say, for instance, we regularly want to check the size of every file in a directory. We can do this in the shell but it's a little unwieldy:

```
$ du -sh * | sort -nr
```

`$` is the symbol for the command prompt, and indicates that we're inputting this command into the command line.

`du` is short for disk usage. `-sh` is short for human-readable summary while the `*` means all. So the part before the pipe `|` means to display the disk usage of all the files in the directory where we run the above command.

The part after `|` is `sort`, which means the list of files should be sorted. The `-nr` flag indicates that the files should be listed in descending order by their size.

Finally, the `|` is a pipe between the two that *chains* the commands together — the same as when we chain functions together in a language like JavaScript.

In short, the command above lists the disk usage of all files in a directory in descending order.

Here's an example of what `du -sh * | sort -nr` outputs in my Address Book project directory:

```
4.0K    js
4.0K    index.html
1.0K    css
0       README.md
```

Take note that if you try running this command in your own Address Book directory, your file sizes may be different.

This is a pretty useful command, especially if you want to keep an eye on which files are taking up a lot of space. But remembering the `du -sh * | sort -nr` command is pretty tough — and not too much fun to type, either.

Wouldn't it be nice if we could just type `ls_size` instead? It's easy to remember and easy to type. (And you can customize it however you like to make it more convenient for your personal environment.)

That's exactly what an alias does — and many developers use them to make their command line experience better. Here's how we'd create an `ls_size` alias.

First, we need to open up the configuration file for our shell. If you are using bash, you'll need to type in the following command:

```
$ code ~/.bash_profile
```

This will open the bash configuration file in the root directory of your machine.

If you're using zsh, then you'll type in the following command instead:

```
$ code ~/.zshrc
```

This will open the zsh configuration file in the root directory of your machine.

If you aren't sure which shell you're using, you should be able to run the following in the terminal to check:

```
$ echo $SHELL
```

Once you have the configuration file for your shell open, it's time to add an alias. We can do so with the `alias` keyword:

```
# Put this in your configuration file. This will likely be
either .bash_profile or .zshrc.

alias ls_size="du -sh * | sort -nr"
```

As we can see, the structure of the alias includes the `alias` keyword and the name of the alias (`ls_size`). We then use the `=` to set the alias equal to some command (which should be enclosed in quotations). In this case, that command is the following (with the quotes included): `"du -sh * | sort -nr"`

If you have a terminal window open, your new alias won't work just yet. Whenever we add an alias or *any* change to our shell configuration, we need to refresh the terminal window. There are two ways to do this:

- Run `source ~/.zshrc` or `source ~/.bash_profile` (depending on which shell you are using). This tells the terminal window to use the updated shell configuration file as its source, not the outdated one.
- Alternatively, you can close out of all terminal windows and reopen the terminal.

Once you've done that, you'll be able to use the `ls_size` command in your terminal.

At this point, you should start thinking about which commands you might want to alias. Are there any commands that you're using a lot and that you'd like to shorten? They might be good candidates for

aliasing.

However, there's a very important and big gotcha that you need to remember when you create your own aliases. **If you name an alias the same name as an existing shell command, the alias will override the existing command.** That's not always a bad thing. For instance, if you want the `ls` command not just to list all files in a directory but do some extra work (such as colorizing the output), and you *always* want this to happen, you might alias it.

But here's an example of something you *should not* do:

```
# Don't add this to your shell configuration! It will overr
ide the ls command.

alias ls="echo 'hi!'"
```

If we were to add this alias, whenever we type in `ls`, our terminal will just output `hi!`. We pretty much just got rid of the `ls` command on our machine.

An easy way around this gotcha is to try typing the name of your alias in the terminal (or looking it up first). If we wanted to check if there is a `hi` command, for instance, we could type `hi` in the terminal and we'd get a message like this (for bash): `-bash: hi: command not found`. That means the command doesn't exist and it's safe to use `hi` as an alias.

Using bash Functions as Shortcuts

Let's say we want to run a bunch of commands in the terminal. We often run these commands together and it's a hassle to type them all in. We can write a simple bash function to run the commands for us. (Most bash functionality is compatible with zsh.)

Here's an example that will create a new directory, `cd` into that directory, and then create a `README.md`:

```
# This would go in your configuration file – .bash_profile
or .zshrc. However, this is just an example. You can try it
out yourself and then remove it from your configuration if
you like.

make_move ()
{
  mkdir $1
  cd $1
  touch README.md
}
```

This function is called `make_move`. When we call it, it will `mkdir $1`. `$1` is the first argument passed into `make_move`. For instance, if we were to call `make_move hello` in the command line, the value of `$1` would be `hello`, which means this function would make a new directory called `hello`. It would then `cd hello` (due to `cd $1`) and finally it would add a `README.md` file in the `hello` directory (`touch README.md`).

As you might guess, `$2` represents the second argument passed into a bash function, `$3` represents the third argument, and so on. That makes it easy to write bash functions that can handle multiple arguments.

The example above is arbitrary and not particularly helpful so we don't recommend that you add this specific function to your configuration file. However, the example should give you a good sense of how to customize bash functions to make your command line experience more convenient.

You aren't expected to add aliases to your personal environment while you are a student at Epicodus. However, to improve your personal experience, you might find it helpful to think about any

pain points you're experiencing with the terminal and then determine whether aliases might be helpful. Also, you may be encouraged to use specific aliases in future lessons to make your development experience easier.

Previous (/intermediate-javascript/object-oriented-javascript/further-exploration-local-storage)
Next (/intermediate-javascript/object-oriented-javascript/whiteboard-practice-javascript-objects)

Lesson 30 of 33
Last updated March 23, 2023

disable dark mode

(http://www.epicodus.com)