Setting up the Environment

It’s possible to start JavaScripting just by opening your browser, but for the purposes of this course, we’re going to introduce two new pieces of software that work to make your programming life easier.

This establishes a programming **environment**, which is a workflow in which multiple pieces of software work together in order to make reaching your goals as easy as possible. Many programming rely on IDEs, or integrated development environments, such as [Xcode](https://developer.apple.com/xcode/) for writing MacOS and iOS apps or [R Studio](http://rstudio.com) for writing in R. These are one-stop shops that try to handle the entirety of your project within one piece of software. We’ll reach a similar state with Atom, below, but not before first making a detour through Git.

A key concept to application development is **versioning**. That is, by making use of a piece of software called a **version control system**, a project can work as though it has a time machine attached to it. But not just a time machine.

Before moving forward, note that above I don’t refer to a “document” or even a “file.” Most of the homework you’ve ever turned in before for an English class has probably been a document, most likely written in Microsoft Word. In this environment, we’re not creating one document or one file. Instead, we’re working on one **project**, which is made up of many files, where each file contributes its own bit to the project as a whole. A website (or a webapp) is an example of a project—one that is made up, of course, of many documents. For example, on even the most basic website, the information on the front page could be saved in one file, and the information in the “About” page could be saved in another.

Thinking in terms of *projects*, not documents, is perhaps the first vital leap for you to make. What we are building in this course is a website that is a project. As such, you will have a folder for your project, and everything in that folder will be under version control.

## Git

The version control system we will use in this course is called [Git](https://git-scm.com/). Git is famously opaque, and, worse, a novice user (like us) uses about three commands all the time except when things go wrong. Luckily, when things go wrong, Git is often there to help us out. Nevertheless, Git deserves its reputation that leads to websites like “[Oh Shit, Git!](http://ohshitgit.com)”

Git calls a project a **repository**, so in the context of using Git, I’ll do the same. A repository is a folder with files in it, like any regular folder of files you have on your computer. However it also has, hidden from view, a history of all the files in the folder. Git knows exactly when a file was added. It knows every change ever made to each file. It even knows about files that have been deleted, and it can resurrect them, if needed.

So far, this sounds like a backup system. And Git is *also* that. But it can do much more. For example, every change you record with Git, called a **commit**, also has a **commit message** attached to it, where you can leave a note for yourself (or a collaborator!) that tells you what you were thinking when you made a change. Why did you put this paragraph before that one? Read your old commit messages.

Additionally, Git lets you **push** your changes to a central version of the repository. This means that you can work on one computer, save your work, commit it, push it, and then, on another computer, **pull** the changes and get right back to work—with all that history still built in, even though you were working *on another computer!*

That covers all of what we’ll be doing with Git in this course, and that is already a lot. But as you can see, the system of committing, pushing, and pulling means that collaboration becomes very easy. Additionally, Git allows for **branching**, where you create new branches for your project. Now Git is not only a time machine, but it’s also an interdimensional portal leading you to alternate universes. In one universe, your text could be written full of jokes. In another, the writing can be very serious.

## GitHub

For the purposes of this course, we will be using Git via the website [GitHub](http://github.com). GitHub will serve as the remote point to which you push your project, and GitHub will even, down the road, host the project. At the end of this course, you’ll have a web address including “github.io” that you can send to your friends and family to show off your progress.

In addition to being a user-friendlier face for the Git software, GitHub also includes features that expand what Git offers, such as built in discussion boards where users can discuss issues. We won’t be using that functionality in this class, but that should stand in to remind you that *things on GitHub are public by default*. Keep this in mind when you’re typing commit messages or working on your project: it’s all visible to anyone who has a look at your repository. Don’t let that scare you from using GitHub. Just keep it in mind.

### Create yourself a GitHub user

The first step with GitHub is easy. Go to [GitHub.com](http://github.com) and create an account. Choose the free plan. You should get an email, and once you verify it, you can click on the “Start a project” button.

If you’re a current student, you can [request a discount](https://education.github.com/discount_requests/new) for GitHub. That discount entitles you to a limited number of private repositories.

### Fork the course repository

In GitHub, anyone can access anyone else’s (public) repositories, or projects. And, what’s more, anyone can make an identical copy of that repository for their own use. This is called **forking**, and the next step is to fork the repository I have already made for this course.

While logged into GitHub, point your browser to:

<https://github.com/muziejus/javascripting-english-major-project>

Once that page loads, there should be a “Fork” button in the top-right corner. Click on that. If it asks you where to fork it, click on your user, and now the repository should appear in your account under this URL:

https://github.com/YOURUSERNAME/javascripting-english-major-project

Where it reads YOURUSERNAME, it should read, of course, the GitHub username you chose for yourself. Make a note of this URL, as you’ll need it in a few steps.

## Atom

You’re probably used to writing your English homework in Microsoft Word or, maybe, Pages. Or maybe you use Google Docs. They are all fine word processors, and I’ve used them all. Actually, I’ve been using Microsoft Word since well before you were born, probably. But they all make two conceptual assumptions that go against the work we are doing in this course.

First, they deal with documents, not projects. Remember, in this course you have to think in terms of a project, not just in terms of a single document. Something like Word is great for a single document, like a business letter or five-page paper. When it comes to something with lots of moving parts, like a web project (or a dissertation…), it starts to creak.

Second, the big advance Word brought to the computing world was fusing what a document says along with how it looks.[[1]](#footnote-33) In Word, whatever the document looks like on your screen is a pretty good approximation of what it will look like in your printer.[[2]](#footnote-34) You can change fonts, font sizes, margins, and so on, for a single document, and when you print it, it will follow your instructions as closely as possible as it prints to a US Letter sheet of paper.[[3]](#footnote-35) This is, of course, because of the first assumption: you’re writing a document that is going to get printed.

We’re writing a website, however. And who knows what kind of device will be used to view it. Imagine if you write a page with 2-inch margins and visit it on a smartphone. You’ll see nothing but margins! When you’re writing for the web, you’re handing off a lot of decisions regarding how the content will look to the user, who could be using a giant monitor or a teeny smartphone to view your site. Of course, you still have *some* control over how things look. A lot, in fact. But in order to do that, you have to rethink the basics of writing. You now have to focus on the content much more than the look.

### A plain text editor

One way of “focusing on the content” is by stripping away a lot of the frippery that a program like Word provides. We won’t need something that can make charts or insert clip art. We won’t be changing fonts willy-nilly. And we definitely aren’t creating something that lives only to be printed out. We are creating, again, a *project*, not a document.

Furthermore, we will be writing code in addition to text. In your own project, you will be programming functions and writing paragraphs. The former requires simple text files that can be read by the JavaScript interpreter (also known as the browser). The latter will be converted into HTML, which means it has to be written in plain text that can be fed to the HTML renderer (also known as the browser). Instead of one .docx file, we will have .js files for our JavaScript programming and .html files for our HTML webpages. Finally, Git is much friendlier with regular text files (like .js or .html files) than it is with .docx files. You can use Git with Word, but then you’re basically just using it as a backup system, not as a version control system.

What we need, then, is a “plain text editor.” Something that just writes text, more or less agnostically. Though it’d be nice if maybe it colored the text certain ways when writing JavaScript or HTML. But let’s not jump ahead. You’ve probably used a plain text editor before, most notably the Notes application on your smartphone. Every personal computer has some kind of plain text editor installed, but we’ll install one that runs on nearly any personal computer, Atom.

### Enter Atom

[Atom](http://atom.io) is a fully-featured text editor that can also serve as an IDE. It also has Git (and GitHub) support built in. This makes sense; it’s part of the GitHub ecosystem. Atom is a serious program with lots and lots of features. I’ll only be teaching a few here, but I’ll also teach a few more as the course goes on. It also does not behave like Word, which will take a bit of getting used to. Still, I hope that, for project-based work, you will see that it works much better than Word or a similar word processor.

You can download the software from the [Atom page](http://atom.io), and installation should be rather straightforward[[4]](#footnote-39). Once you install it, when you open it, you will be greeted with a welcome tab and the welcome guide tab in a separate pane. Atom works on the visual metaphor of panes with tabs. You can change the widths of the panes and show and hide them with your mouse. Similarly, the tabs work like the tabs do in Chrome, for good reason.[[5]](#footnote-43)

### Start customizing with packages

From the Welcome Guide, I recommend immediately clicking on “Install a Package.” Ben Balter has come up with a [list of useful Atom packages for writing prose](https://ben.balter.com/2016/12/23/atom-for-prose/), but I’ll mention the most useful ones here. Once you click on “Install a Package,” you can click on “Open Installer” and start installing packages by searching for them and then clicking the “Install” button.

* [file-icons](https://atom.io/packages/file-icons): this gives you pretty icons in the Atom sidebar and tabs. These visual cues are, in my experience, more useful than file name extensions.
* [linter-jshint](https://atom.io/packages/linter-jshint): adds a linter based on [JSHint](http://jshint.com/docs/) for JavaScript, meaning the Atom will warn you when the JavaScript you write has problems.

For these, you have to type the name into the little search bar in the Install Packages part of the Settings tab. The linter may ask you to add some other packages to fulfill dependencies. That’s ok.

### Continue customizing by enabling autosave

Atom ships with autosave disabled by default. That’s probably not behavior you’re expecting, so you should enable it! After you’re done installing packages, you’ll be on the Settings tab, which has a few subcategories, like Core, Editor, and so on. Choose Packages. This gives a list of all the packages you have installed, as well as giving you a chance to configure them. Type autosave into the filter box at the top, and then click on “Settings” once the autosave package shows up under “Core Packages.”

Tick the box beside “Enabled” under settings, and now Atom will autosave any file as soon as you click away from its tab, even if you go to another application.

While the Settings tab is still open, click on “Core” and “Editor” and change things around these if you like. The defaults are pretty good, but you may want to change the “Font Family” in Editor. Type in the name of your preferred font, provided that it is installed. You can make the default text larger or smaller, and so on. One more setting to consider in Editor is “Soft Wrap.” With it off, your text will keep running off the side of the window as you type. Soft Wrap, like in Word or Notes, breaks lines so that everything fits inside the window.

Finally, under Themes, you can do some basic changing of the user interface, theme, and the syntax theme for the editor. Pick a light one or a dark one, whichever you think fits your personality. I personally use the solarized themes, based on Ethan Schoonover’s [Solarized](http://ethanschoonover.com/solarized) precision color project.

### Link Atom to GitHub

Atom is written by the people at GitHub, so it’s pretty easy to link the two together. In fact, that’s the main reason I encourage you to install Atom. The Atom people have [written a decent introduction](http://flight-manual.atom.io/using-atom/sections/github-package/) to most of what I describe here, so it’s useful to glance over at their screenshots, etc., if this part gets too confusing.

First, you have to get the URL of the repository you forked a few steps ago:

https://github.com/YOURUSERNAME/javascripting-english-major-project

Again, YOURUSERNAME should be replaced with your own GitHub username. Now add .git to the end of it, so you have:

https://github.com/YOURUSERNAME/javascripting-english-major-project.git

This is the URL you will need for cloning.

Second, return to Atom. Here, open up the command palette by going to the “Packages” menu, choosing “Command Palette” and then “Toggle.” In the little box that opens, type “git” and choose the option labeled “GitHub: Clone.” For “Clone from,” paste or type in the URL above ending in .git with your username. Atom will automatically save it to a GitHub folder it creates in your home folder, but you can change this location if you like.

Now, if you have the file-icons package installed, in the Projects pane, you should see a small book icon with your repository name beside it, and, underneath it, you should see a folder called .git, a file called README.md, and a few other files. These are the contents of your repository, and they are now on your computer.

Third, let’s make an explicit connection between GitHub and Atom. Open up the GitHub pane (“Packages” > “GitHub” > “Toggle GitHub Tab”), and you should now see a button asking you to log in or a message encouraging you to go to the [GitHub Atom login page](http://github.atom.io/login). If you don’t, make sure that your project is open and loaded in Atom. Otherwise, click on “Login” and then click on the web address, which will open a new page in your browser asking you for permission to have Atom talk to your GitHub account. Click on “Authorize atom” and then copy the extremely long code that appears. Paste it back into the GitHub pane back on Atom.

You should get a message saying “No pull request could be found for the branch master,” and so on. You’ve made the connection.

### Make a change, stage a file, commit, and push

We’re in the homestretch now, but this section is the most important, because it’s a description of what you will be doing with Git in Atom most of the time. In the “Projects” tab, there should be a file called README.md. Double click on it, and it should open in a new tab. You should see some text that I’ve written. Go ahead and delete it all and type in something of your own, like your name and your goals for your project (it’s ok if they’re vague for now).

Just clicking away from the window should autosave the document, which you do by now opening the Git tab (“Packages” > “GitHub” > “Toggle Git Tab”). You will use this tab much more often than the GitHub tab, so it might be worthwhile to remember the keyboard shortcut, control-shift-9.

The Git tab is split into three horizontal sections, **Unstaged**, **Staged**, and **Commit**. Unstaged lists all the files you have saved but have not yet committed. If the icon beside the file is a green cross, that means it is a new file, never before saved to the repository. README.md, on the other hand, should appear in unstaged changes with a yellow box with a dot. That means that the file is in the repository, but changes have been made to it that have not yet been committed. Finally, if you were to delete a file, it would show up there with a red minus sign. Yes, in Git, a file is never truly deleted, remember.

If you click on one of the files in the Unstaged area, a new tab will open with a lot of green (and maybe some red) text. That shows everything you have added or removed from the file since its last commit. With README.md, it will be a lot of red and hopefully a lot of green.

Click on the “Stage All” button at the top of the Unstaged area to move all the files to the Staged area. Or, if you like, you can only move one file at a time. I usually try to commit things thematically. So if I make some changes on one file and totally unrelated changes to another file, I will make two separate commits. But if both are related (say I change the name of something in both places), then it can be one commit. There’s no right way to committing. Do what feels like a good balance between often enough to be useful yet infrequent enough so that you actually get work done.

Once files are in the Staged area, you can commit. That will take the files in the Staged area and log the changes to the files in the Git time machine. Any changes you make backwards or forwards between commits doesn’t matter. Git only tracks commits, not individual saves.

Type in a useful commit message (like Edit README) in the Commit box and press “Commit.” The changes have been recorded.

The final step is pushing the changes up to GitHub. Once you have a commit, the up arrow at the bottom right corner of the Atom window will have a little “1” appear next to it. If you make another commit, that “1” will become a “2.” Once you’ve got enough commits and want to push, click on that arrow, and Atom will ask you for your GitHub username and password. Type them in, wait a minute, and then if you go to GitHub and refresh your repository, you will see the changes that you’ve made.[[6]](#footnote-55)

### Parting Atom thoughts

There’s a whole lot going on in this section, but it’s mostly stuff you just have to do once to set up the environment. As I wrote above, Atom is a very heavy duty program that can do a whole lot more than what we will use it for. Whenever you get frustrated with Atom’s menus and the like, always remember that you can launch the command palette by typing command-shift-p on a Mac or control-shift-p on Windows. That brings up a small window that lets you type in whatever command you want to execute.

Also, Atom will be frustrating the first few times. Any new piece of software is. By the end of this course, though, I hope that you’ll see why I insisted on this unpleasantness at the beginning. And if you are interested in learning more about Atom, [please have a look at their documentation](https://atom.io/docs), that includes the *Flight Manual* book, and an introductory video.

## Browser

Our last step in this chapter is making sure your browser has a JavaScript console. I recommend against using Internet Explorer in this course for a number of reasons, so I’ll only give directions for Firefox, Chrome, and Safari.

* In [Firefox](https://developer.mozilla.org/en-US/docs/Tools/Web_Console/Opening_the_Web_Console), the console is hidden the “Tools” menu, under “Web Developer” and “Web Console” on the Mac. On Windows/Linux, it is in the “Web Developer” submenu of the “Firefox” menu. Or, type control-shift-k for Windows/Linux or command-option-k for Mac.
* In [Chrome](https://developers.google.com/web/tools/chrome-devtools/console/), open the DevTools palette either by typing control-shift-j (Windows/Linux) or command-option-j (Mac). Or, find the tools in the Chrome menu (upper right, beside the address bar), under “More Tools” and “Developer Tools.” Once the DevTools palette opens, you can click on the “Console” tab.
* In Safari, look in the “Develop” menu and choose “Show JavaScript Console.” Or type command-option-c.

Newer browsers will behave similarly. Brave, for example, is based on Chrome, so the keyboard shortcut is the same. Incidentally [Brave](https://brave.com) was co-founded by Brendan Eich, the man who invented JavaScript.

Whichever browser you’re using, the console looks more or less the same. It’s a large empty window with a > at the bottom. This > is called the “prompt.” Beside the prompt, type:

console.log('Hello, World!');

Hit return. The console should respond with Hello, World! Throughout the rest of this course, when something needs to be typed at the prompt, I will include the >. This time I skipped it, to make things a bit clearer for those who have never used a prompt before.

By typing the above, you’ve written your first bit of JavaScript. In other words, you’re ready for the [next chapter](/2-calculator/) after completing the exercises below.

## Exercises

1. Create a GitHub account.
2. Fork the blank repository for this course from <https://github.com/muziejus/javascripting-english-major-project>.
3. Install Atom on your computer with some useful packages.
4. Link Atom to your GitHub account.
5. Use Atom to clone the repository you forked.
6. Flesh out your hopes for your personal project in the README.md file and commit the changes.
7. Push your commit(s) from your computer up to GitHub.

## Footnotes

1. Of course, whether a text still means the same thing when it looks different is an unsettled question in literary study. [↑](#footnote-ref-33)
2. Microsoft Word was not, of course, the first “WYSIWYG” (“What You See Is What You Get”) word processor. It appeared around the same time as MacWrite did, and both of those applications were building on ideas established years earlier at Xerox. Nevertheless, as Windows began to gain dominance on the PC market, Word became ubiquitous, leaving it as the “default” example of a WYSIWYG word processor to this day. [↑](#footnote-ref-34)
3. Or A4 paper, if you’re outside the United States. [↑](#footnote-ref-35)
4. There are two hiccups that will come up during installation when you aim to use git with Atom. On Macs, you will be prompted to install the command-line tools. It should work automagically, but if it doesn’t, see [this post](http://osxdaily.com/2014/02/12/install-command-line-tools-mac-os-x/). On Windows, you need to install the [gitbash shell](https://gitforwindows.org) and then [configure git](https://help.github.com/articles/setting-your-username-in-git/#platform-windows) using gitbash. [↑](#footnote-ref-39)
5. Atom is, basically, a very customized version of Chrome that talks to a JavaScript server that you run in the background when you launch the application. [↑](#footnote-ref-43)
6. I, personally, overpush. I usually commit and immediately push, which is generally fine, but it can be embarrassing sometimes. Also, since the default means by which Atom pushes to GitHub asks you for your username and password, it makes sense to push only every few commits or so. Find a balance that works for you, but remember to always finish your work before a break with a commit and a push, just in case! [↑](#footnote-ref-55)