Using Git in the classroom

Posted 25 Jan 2016 | Michael Taggart (/users/mttaggart) | 136 | 6 comments

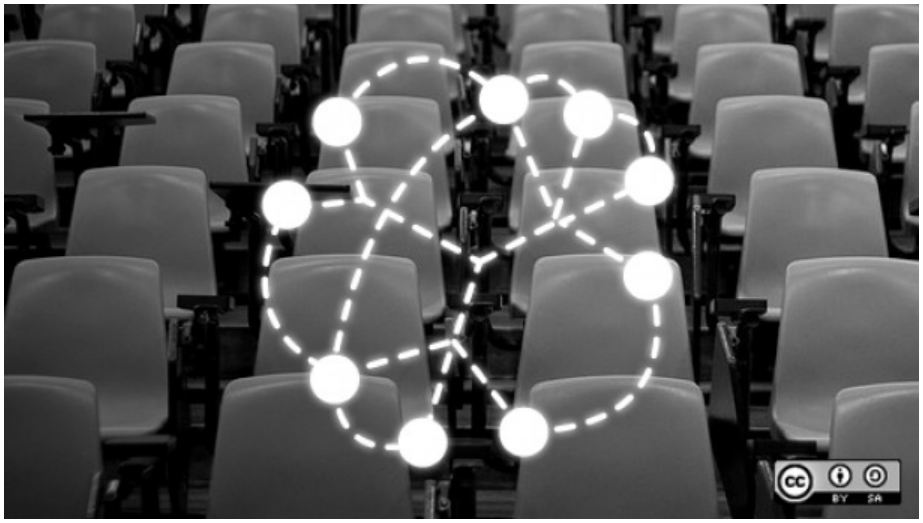


Image by : *opensource.com*

In my advanced programming classes I've discovered that middle school students are capable of far more complex operations than we often suspect. In many cases, they're wholly capable of using industry-standard tools to produce remarkable work.

In our class, one of the central tools for our work is Git (https://git-scm.com/). It's a version control system designed by none other than Linus Torvalds (https://en.wikipedia.org/wiki/Linus_Torvalds), the creator of the Linux kernel.

About the author

Michael Taggart - Taggart is an educational technologist, writer, and sometimes programmer. After spending his entire life in the Philadelphia area, he packed up and moved to Los Angeles, where he currently lives with his wife and several adorable Linux-based computers. He is committed to improving the implementation of technology in schools, and to making technology resources tools for a better future rather than simply the toys of the privileged.

» More about me (/users/mttaggart)

» Learn how you can contribute (/participate)

Looking at the front page of the Git site should give you an idea how well-established it is in the industry. It has as much place in a computer science class as any other tool we might use—even at the middle school level.

We'll talk about Git's features more deeply in later sections, but the big kahuna is this: syncing to a remote repository and tracking changes of code by multiple users. This is the technology that enables GitHub (https://github.com) to be the incredible library of open source projects that it is. The same features that enable GitHub to be GitHub also allow teachers to manage multiple students' projects easily, and even handle collaborative work fairly and clearly.

GUI helpers for Git abound (GitHub even offers their own (https://desktop.github.com/)), but we have found that the plain-ol' command line is easiest to use. What's more, getting comfortable with Git and the command line in middle school pays dividends in the long run.

Getting started with Git

On Mac and Windows, installing the desktop app linked above is an easy way to get started. This will also install the command line tools that undergird the Git system. Again, I **highly** recommend using the command line over GUI tools. Here's why.

Embrace the keyboard

Students **love** learning the command line. Like programming languages, the command line seems like a secret power, like arcane knowledge. With proper scaffolding, it can be as simple as teaching any other user interface.

Yes, the command prompt can be scary:

```
$ you-could-enter-anything-here
```

It's the computer version of looking long into the abyss—it looks into you. But consider this: the interface language is a human language. Just words. That's

something that, in truth, is *more* familiar than the arcane iconography employed by most UIs. With a little practice, I predict you and your students will have terminal windows open at all times and use them whenever you can. There's an old saying about GUIs: they make simple things simple and complex things impossible. Once you've learned `touch`, `mkdir`, and `cd`, see whether the GUI or CLI takes more time to get things done.

And not for nothing, but mastering the CLI provides a sense of accomplishment for students. They're proud to know it, and that kind of self-esteem can go a long way.

The steps in this article will all be done using CLI examples.

Your first repo

Time for the Git equivalent of *Hello, world!* Let's make a new directory to use as our Git repository:

```
mkdir gitdemo
cd gitdemo
```

Now we're inside our new folder. Time to make it a proper Git repo:

```
git init
```

You'll see Initialized empty Git repository in `/path/to/your/repo/.git/`. What's that `.git`? If you list all files in your directory (`ls -a`), you'll see a new hidden `.git/` directory. That's where Git stores the information about this new repository. Time to add some files.

```
touch new.txt
echo "Hello, World!" > new.txt
```

You'll have a new file, `new.txt` that now has one line of text in it: *Hello, world!* Pretty cool how we could do that from the command line without opening a text editor, huh?

But this isn't just any old folder; it's **Git repository**! Git has tracked that we have a new file. Enter the following command:

```
git status
```

We'll get back the following:

```
On branch master

Initial commit

Untracked files:
(use "git add ..." to include in what will be committed)

new.txt

nothing added to commit but untracked files present (use
"git add" to track)
```

Here's a point-by-point translation:

- We can have multiple versions of this folder, and you're looking at one called *master*.
- No commits (saves) have been made yet.
- I see some files in this folder you haven't told me to care about yet. Here they are...

Committing in Git is a little more nuanced than just hitting save. Git lets us choose what files we track inside a repository. It also lets us stage files to be saved into its history, rather than just saving everything all the time. This might seem like a hassle, but over time the advantages of such control reveal themselves.

So before we commit `test.txt` to the ANNALS OF HISTORY, we have to *stage* it.

```
git add new.txt
```

No output after that, but running git `status` again yields:

```
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached ..." to unstage)

new file: new.txt
```

Okay, excellent. Git knows about our file now. Time to commit our changes to Git's history.

```
git commit -m "Add new.txt"
```

The `-m` flag provides a commit message. Such a message is required for all commits. If you don't provide one with the `-m` flag, Git will kick you into a command line text editor, which can be unnerving when you're not used to it. I recommend using the `-m` flag for now.

After the commit command, you'll see output like this:

```
[master (root-commit) c6c1180]
Add new.txt 1 file changed, 1 insertion(+)
create mode 100644 new.txt
```

Translation:

- Our first commit! Also we're on the master branch. I've made a unique id for this commit.
- Here's what changed in this commit.
- I made one file.

We're almost done! But most of your commits won't be initial commits, right? They'll be updates of repos you've already worked with. So let's make some changes.

```
echo "Foobar!" >> new.txt
```

This adds a new line (again, no text editor needed) to our `new.txt`. A quick run of `git status` reveals:

```
On branch master
Changes not staged for commit:
(use "git add ..." to update what will be committed)
(use "git checkout -- ..." to discard changes in working
directory)

modified: new.txt

no changes added to commit (use "git add" and/or "git
commit -a")
```

Look at that! `modified: new.txt`. Git sees changes to our file. If you're curious exactly *what* has changed, you can run `git diff new.txt`. That yields something like:

```
diff --git a/new.txt b/new.txt
index 8ab686e..d8eeaac 100644
--- a/new.txt
+++ b/new.txt @@ -1 +1,2 @@
Hello, World!
+Foobar!
```

Notice the `+` around the added line. This seems like a reasonable change, so we're going to commit it. Don't forget to `git add new.txt` or `git add -A` to add that file or all files, respectively, to the staging area. Another call to `git`

`commit` with an appropriate commit message gives us a total of two commits in our history. Go ahead and see what Git is tracking for you with `git log`.

Helpful, no? Git's functionality goes much deeper, but you've just gone through a fairly standard Git workflow for developers.

Going remote

Using Git to manage local projects is very helpful, but the technology really shines when paired with remote repositories like GitHub or Bitbucket. So let's go ahead and do that. I've chosen to use GitHub for this article, since that's the preeminent remote repository service right now. I do like Bitbucket for personal projects, but there's a serious benefit to being on the same platform as other developers. Think of it like social media for programmers.

You can set up a GitHub account for this if you like, but you won't need it, since I've already made a repo for you. You're going to connect to my remote, online version of the `gitdemo` repository and download my changes, which will then merge with yours.

SECURITY NOTE: We're all friends here in the FOSS community. And we download code from other computers all the time. But what I'm about to ask you to do requires a certain amount of trust, and I don't think you should do so blindly. Please feel free to review this tiny repository (https://github.com/mttaggart/gitdemo) to assure yourself that I am not attempting to infect your machine with malicious code. Or any code, for that matter.

With that out of the way, let's set up your local repository to look at mine for upstream changes by entering:

```
git remote add upstream https:// github.com/mttaggart/gitdemo
```

No output means it worked. We're now ready to download (fetch) and merge the files on GitHub with your local repository. We can do both at the same time with `git pull`, or do both steps separately with `git fetch` and `git merge`. For simplicity's sake, let's pull:

```
git pull upstream master
```

Hey, what's that master thing? No worries; that just means we're on the *master* branch of the repository, the main branch.

I guess that'd be the trunk. Anyway, you see this output:

```
Merge made by the 'recursive' strategy.
README.md | 1 +
1 file changed, 1 insertion(+)
create mode 100644 README.md
```

Looks like you have a new file there! Go check out `README.md` and see what's up. If you want a nice visualization of what just occurred, go ahead and run the following:

```
git log --graph  # q quits, j and k scroll
```

You can see that there was a merge of two branches—remote and local, resulting in the current state of affairs.

If you and I were working together on this repository and you then made some changes you need to commit to our shared GitHub repository, you'd simply `git push` to upload your changes to GitHub, merging them with what was there previously.

"Great Taggart, now I can use Git. How does that make me a better teacher?" The secret is thinking outside of code.

Git portfolios

Lots of educators talk about meaning digital portfolios for students, but few actually pull the trick off. It's tough—how do you represent so many different documents? How do you keep track of what should and shouldn't be included?

How do you ensure that the portfolio is built on a platform students will be able to access, even after they graduate or leave the school?

Git, Git, Git, HTML, and Git.

Imagine having students maintain a directory of documents (ideally Open Document Format, or even better, HTML) that they curate throughout their school career. Perhaps every quarter they make some decisions about what work they're proudest of. They add these files to a Git repository, commit, and push to a remote repo. It needn't be GitHub if privacy is a concern. Projects like GitLab (https://gitlab.com) allow schools to create their own Git remote repositories without relying on third parties.

With the tracking and versioning features that developers rely on for clean code, students get a nice timeline of their work history. They can even create *releases* of their portfolio at different points in the school career to demonstrate growth. I realize Git was never intended for this purpose, but I believe it to be a possible solution to the portfolio problem.

Computer science with Git

I run my middle school programming classes with Git/GitHub. I realize that GitHub has a rather nice solution (https://classroom.github.com/) for managing multiple student repos, but I decided to roll my own—in part because the school already uses a learning management system that can track assignments, and in part to force myself to learn Git well enough that I'd be confident to use it to run the class.

Here's how I did it:

1. Create the upstream repo

Upstream is the repository where you submit *stub* or starter code for your students to work from. I also put my Markdown-formatted lesson objectives in this repo. Each lesson gets its own folder. Every week, a new commit brings a new project.

2. Set up student accounts

GitHub's Classroom (https://classroom.github.com/) solution allows educators to create private repositories for student work—a major advantage if privacy is a concern. Alternatively, you can just allow students to create personal public accounts on GitHub or Bitbucket. If the students own the account from the outset, they've gotten an early start on their career as developers. What's more, creating the account could be an opportunity to discuss online safety, digital citizenship, and intellectual property.

3. Fork upstream

After students create their accounts, they should *fork* your upstream repository.

Forking creates a personal copy of your code (open source is great, isn't it?) that students can change and extend on their own. You can add as little or as much to upstream as you like; students will take it and run from there.

4. Clone/pull student repos

`git clone` grabs a remote repository and downloads a local copy to your computer, complete with the repo's commit history. I have a folder in which I've `git clone` d all of my students' repos. Then, using a little Bash script I wrote, I pull all the repos as once, creating a combined log file of each student's last two commits. Their commits tell me what they've worked on since last I checked, so I know to go back and look at old lessons if they've improved their solutions.

Git: Many problems, one solution

Surely, dear educator, there is a tremendous learning curve to implementing something like Git into a classroom—many times that if considering Git as a portfolio tool. 'Twas always thus, though. No major change in education practice, even education technology practice, comes without considerable difficulty. As open source advocates *and* education advocates, we should strive for solutions that are effective, fair, and accessible. Although unfamiliar to the non-

programmers, Git as a technology meets all three criteria. Its future is assuredly bright, and I hope you consider exploring its potential use in education.

Tags:

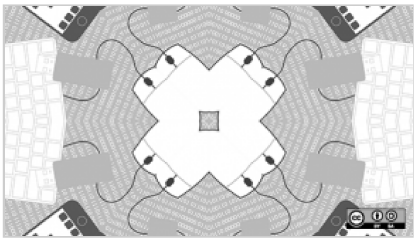
Education (/tags/education), Git (/tags/git), GitHub (/tags/github), Getting started (/tags/getting-started)

Recommended reading

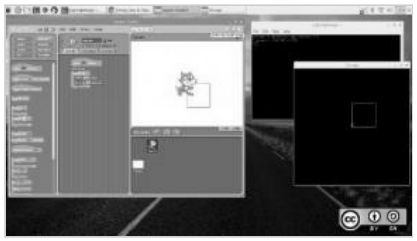
What does SVG have to do with teaching kids to code? (/article/17/5/coding-scalable-vector-graphics-

A new Android app for teaching kids how to read (/article/17/4/phoenicia-education-software)

Happy birthday, Git (/article/17/4/happy-birthday-git)



Developer opportunities to code for good (/article/17/3/interview-anh-bui-benetech-labs)



Is Scratch today like the Logo of the '80s for teaching kids to code? (/article/17/3/logo-scratch-teach-

Open project collaboration from elementary to university classrooms (/article/17/3/education-

6 Comments

Outsider on 25 Jan 2016

I love this idea just a shame git is terrible with anything other than txt files for remote use.

0 0

Sachin Patil (/users/psachin) on 02 Feb 2016

https://git-lfs.github.com/ (https://git-lfs.github.com/)

0 0

Tobin Davis on 25 Jan 2016

Never underestimate their potential. My youngest son (now 24) wrote Tuxblocks (SF project) in 8th grade (~2005 - before git existed). He even wrote his own SCM tools that function similar to git (although not nearly as feature complete as git today).

0 0

Suhendro on 26 Jan 2016

I hope that Mr.Taggart can explain more on how to use the git in his class. For me (I already know a little bit about Git) really like to know on how to use it in the classroom for my case study.

0 0

Michael Taggart (/users/mttaggart) on 29 Jan 2016

Suhendro, thanks for reading! I've made the tools I use in my class available on GitHub: https://github.com/mttaggart/gitclass (https://github.com/mttaggart/gitclass). I'd also be happy to answer any questions about how our class runs.

0 0

Matthias Otto on 23 Feb 2016

Thanks, Michael the last 4 steps of your blog are exactly what I had in mind for my upcoming Software Development classes.

One or two questions: The "GitHub for the Classroom" tutorials recommend creating an "Organization" first, and creating the master Assignment repo in there. They then fail to explain to a beginner like me how that benefits the running of the class.

Students can fork my (public) repo whether or not it lives in an organization. I realize I can invite people into the organization (if I had to do this for all students manually I would regard this as a hassle, but would do it if I saw the benefit). I also can set up "Teams", but again, not sure what to do with all this power.

In other words, what can a member of my Organization do (with my public Assignment repo) that somebody outside the Organization cannot?

Question two: Version control is about conflict resolution, i.e. how to resolve issues when local and remote repos are synced or branches are merged/pulled. In the scenario above (steps 1-4), that situation never occurs, as students only sync (pull push) against their own repos.

I'd like to come up with some scenario, where local and remote repos differ, but without having all students merge back into the same place, which will get too messy to handle. See someone's experience here: https://mdnahas.github.io/doc/Git_in_the_Classroom.html (https://mdnahas.github.io/doc/Git_in_the_Classroom.html)

I could introduce small 2-student teams that work off the same remote repo, but group work like this is always problematic in education (e.g. groups becoming dysfunctional, members not pulling their weight, although this would become evident in the history, I guess).

Alternatively I thought of "helping" my students remotely when they are stuck with their code, and push my clone back up to their repo. This is a little "light-weight" but better than nothing? Interested in your thoughts.