**VERSION CONTROL (SOURCE CODE MANAGEMENT)**

**Why Version Control?**

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Every file on the computer file system can be version controlled: directories, regular files, binary files. “Every file” means every file, not just source code files. But since we’re programmers we’d be *version controlling* source code files.

**Version Control Tools**

VCS take and store snapshots (or versions) of our files. They don’t act autonomously, though. Everything they do under our instruction; including asking them to start tracking changes to certain files, saving snapshots, traveling back in time (and staying in the past), completing forgetting some part of our history or all of it. What I’m trying to say is this: when something fucks up, and your obligatorily pointing your accusing finger, never ever let it settle on the version control system. It is a boringly obedient master.

Just so you know, before the cutlery was invented, people ate. Sometimes same foods that seem uneatable without the fork. Also, by the way who made the world start using the fork to eat rice? [Ideally we’d go on and discuss this topic for hours, days, before we continue with the class. But this is not an ideal world.]

How have people done version control in the past?

* curl + gzip + email + prayers + diff + patch + ftp
* Shell script (source of the many **.conf.bak**)
* VCS
* Subversion
* Mercurial
* Git, my Git

**Installing Our Favorite Tool**

So I took a look at your CVs and online profile, etc, and it was obvious that everybody’s favorite tool was [Git](https://git-scm.com/) (By the way, Git is English so look it up in your dictionary. Don’t judge a book by its title #protip).

If I was wrong about that, treat is as our first assumption. And there are many more to come. In fact, one follow immediately that we’ve all being saved by [Linus Torvalds](https://en.wikipedia.org/wiki/Linus_Torvalds). Any Bill Gates boys in the building? So we’d go ahead and install Git on our [Ubuntu](http://www.ubuntu.com/) boxes.

Follow the instructions below to install Git on your computer.

$ sudo apt-get update

$ sudo apt-get install git

The astute programmer immediately recognizes that sudo apt-get install git gets you an outdated version of Git and is so disconcerted, perhaps angry at me. (Mind you, it still works better than latest versions of others out there.) So he sets out to compile the source for herself. Trust me that makes sudo apt-get install look like a walk in the park.

**Easiest Things First**

Now let’s track some files like we’re the NSA. By the way, the NSA tracks people, but they don’t use Git for that. We’d start learning how to ask Git to track both regular files and directories. We’re going to use your HTML and CSS files from the previous class. If they are all over the place could you please put them in a directory? Thank you.

Now find your way into that directory ($ cd path/to/the/directory), and we’re ready to Git.

**/path/to/the/directory $** git init (1)

Did you see anything change? Did you see any rainbows in your command line? If not run this

yes “$(seq 231 -1 16)” | while read i; do printf “\x1b[48;5;${i}m\n”; sleep .02; done

Well, something changed, even if it’s not really clear: git created a new directory, .git here. That’s where Git will store everything it knows about our directory and its contents. That’s where everything is. So don’t delete it unless you want Git to lose its “mind”. And don’t run git init again some time in the future. I think it’d do some horrible things to you. Also don’t touch the contents of .git, now. It also created a hidden file called .gitignore.

**Checking on you**

Our next Git command, and perhaps one that we’d be using more than anything else:

**/path/to/the/directory $** git status (2)

Ideally our outputs should be the same but I guess they’re not. A few things are the same though so we have shared interest in knowing what they are. Everyone go figure their differences for themselves.

On branch master

Initial commit

Untracked files

This is not the right time to discuss **branching**. It’s a pretty significant feature and a big win for Git. You’d learn more about it while doing working on Assignment 1. And we’d have a date with it (and ultimately enough of it) before the end of the year so dry your eyes. For now it’s ok to be on master. Take my word for it.

On branch master. Check.

Even though we initialized a Git repository in the current directory, Git can only stalk its contents. It **doesn’t track** them yet. But Git is capable of tracking them, so it tells us, hey these are the contents of the directory, and I can track them all. Files that can be tracked but not yet are the **Untracked files**. Now tell Git to track any one file of your choice. (See the git status output for help.)

Initial commit. Check.

Untracked files. Check.

We can keep adding/staging the files one after the other, or lump them together. The command is the same though:

**/path/to/the/directory $** git add <filename> (3)

Replace filename with **.** (the period), the symbol for the current directory, to lump everything together. There’s another option which requires 2 characters. Impress me! (But here’s a cheat for those who could use one: read the output of git help add.)

**If you love it then you better put a ring on it**

Then we have to commit, or save (for those allergic to commitment). That is how to ask Git to take one snapshot. Thus, creating a snapshot requires 2 steps: (1) adding changes, and (2) commiting the changes. But wait, that is how Microsoft Word works too: type in more words, press Ctrl + S, delete more words, press Ctrl + S. One huge (and I mean huge literally) difference is, Git won’t accept changes that you keep mum about. Or rather it is good behavior to label your changes. So you say something small but immensely useful about the latest changes you’re saving.

**/path/to/the/directory $** git commit -m “Describe the changes” (3)

My favorite is:

**/path/to/the/directory $** git commit --verbose (3)

which opens up a full blown editor with details on the changes you’re about to save. I think I love it because of how my editor displays it. Also because I can see all the changes I made, which directly contributes to the quality of the commit message. **Commit messages are serious business. Approach them with your most intense seriousness**.

Because of that Git allows you to rewrite inappropriate commit messages. The world had lost a lot due to shitty commit messages. Therefore, don’t write shitty commit messages. It takes practice and a lot of feedback to be better at writing them, so have someone take a look before you save. Or after. And if they don’t wax poetic,

**/path/to/the/directory $** git commit --amend (4)

Write a better one. (See the links below for the internet’s favorite guide on how to write Git commit messages.)

Let’s make more commits, about 20 of them, and then we’d traverse the graph! Go!

**Going up and down the graph**

Not now, but there’d be a better time to get lost in Git’s graph.

**Distributed Version Control**

I don’t want to break your heart but everything you’ve done up to this point is only a tiny part of the Git fun. Granted, we’ve intentionally skipped some highgrade fun stuff because [yagni](http://martinfowler.com/bliki/Yagni.html). And, unless you badly need it, you won’t see the fun in them. We’d introduce them as an when they become necessary. Next item.

Let’s talk about remote. It means anything not local, and if local means anywhere on your machine then remote mean anywhere outside your machine. Nothing to impress your business friends with but it’d save you a hell of Googling. Here’s a table I came up with (or rather made up):

|  |  |  |
| --- | --- | --- |
| **Term** | **Local (on my computer)** | **Remote (not on my computer)** |
| Host | localhost | remote host |
| IP | Local IP | Remote IP |
| Port | Local Port | Remote Port |
| Access | Local Access | Remote Access |
| Repository | Local Repository | Remote Repository |

Why would anyone need a remote? And what’s the physical distance that should separate 2 or more computers before they fit the local/remote definition?

People have unique use cases and so we can’t tell why they need remote something. But for us it’s because we need to protect our work against damage. And maybe in the future when someone wants to improve our code we don’t have to share our personal computer with them. So we’d borrow space on some server somewhere to store our code.

And that could be on your partner’s computer, my AWS EC2 instance, your AWS EC2 instance, [Bitbucket](https://bitbucket.org/), definitely not in the ground, or [GitHub](https://github.com/). And, speaking of the devil,

[**GitHub**](https://github.com/) is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git **as well as adding its own features**. (Source <https://en.wikipedia.org/wiki/GitHub>, emphasis mine). This definition applies to every other service out there—including what I provide on EC2 instance—except when they add their own features.

GitHub is accessible here <https://github.com>. Why GitHub? Free. Great UI/UX, ton of great features, that’s where the whole world is: Facebook, Google, Twitter, Microsoft, Rails, Linux, Node, Ruby, Your Own Company. Getting used to GitHub significantly prepares you. Otherwise you could always return to free private repos on Bitbucket. But read their termsheet first.

TODO:

* Sign up
* Set up Hub (<https://hub.github.com>), a nice little thing that makes Git work better with GitHub.
* Create a repository.
  + Choose a license for your open source project.
  + Add a README

**Adding a remote repository to your local repository**

There’s 2 ways because you have to choose how you want to be identified by the remote server anytime you want to make changes to the remote repository:

* **Via SSH**

[SSH](https://en.wikipedia.org/wiki/Secure_Shell) (Secure SHell) securely connects an SSH client (installed on your computer) to an SSH server, in our case, running at GitHub. **Secure** because it uses public-key cryptography to authenticate server/client. It’s more complicated and interesting that this so I encourage you to read about it in your free time. To use SSH to talk to GitHub, we need a few things:

1. A private key, which we would use to derive
2. A public key, which we’d give to
3. GitHub.

Let’s go ahead and do that now by following the steps in this guide: <https://help.github.com/articles/generating-ssh-keys/>.

* **Via HTTPS**

You’ll have to enter your GitHub username/password combination every time you want to update the repository. Or you could use a credentials helper to store your credentials.

How do you choose between SSH and HTTPS? It’s by how you specify the remote repository’s URL.

To Use SSH:

**$** git remote add origin **git**@github.com:username/repository.git (4)

To Use HTTPS:

**$** git remote add origin **https**://github.com/username/repository.git

where username refers to your GitHub username and repository refers to the newly created repository. **origin** is historical; it could be anything as long as it makes more sense. Now let’s **push** our code files into our remote repository. If anyone wants a copy they can **clone** it, and later **pull** new updates.

**Push**, **clone**, and **pull** are emphasized because while they complete the sentence exactly, they’re technical. We clone remote repositories we don’t have already, we request updates from the remote repository by pulling, and we push new changes we’ve made on our local repository to the remote repository.

And now, let’s get crazy.

**Resources**

* Git Book [Git](https://git-scm.com/)
* Installing Git <https://git-scm.com/book/en/v1/Getting-Started-Installing-Git>
* Commit messages [tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html](http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html)
* Great tutorial on popular commands <http://gitref.org>
* Practice Git in your browser <https://try.github.io>