CODEFIRST: Professionals

Web Fundamentals - Session 3

Version Control & Using GitHub

We gave you this guide and this guide last week.

Recap (from this guide): **Version Control** (aka Revision Control aka Source Control) lets you track your files over time. Why do you care? So when you mess up you can easily get back to a previous working version.

You've probably cooked up your own version control system. Got any files like this?

- ResumeOct2006.doc
- ResumeMar2007.doc
- logo3.png
- logo4.png
- logo-old.png

It's why we use "Save As". You want the new file without obliterating the old one. It's a common problem, and solutions are usually like this:

- Make a **single backup copy** (Document.old.txt).
- If we're clever, we add a **version number or date**: Document_V1.txt, DocumentMarch2007.txt
- We may even use a **shared folder** so other people can see and edit files without sending them over email. Hopefully they relabel the file after they save it.

So Why Do We Need A Version Control System (VCS)?

Shared folder/naming system is fine for class projects or one-time papers. But for software projects? Not a chance.

Do you think the Windows source code sits in a shared folder like "Windows2007-Latest-UPDATED!!", for anyone to edit? That every programmer just works in a different subfolder? No way.

Large, fast-changing projects with many authors need a **Version Control System** (aka for "**file database**") to track changes and avoid general chaos. A good VCS does the following things:

- **Backup and Restore.** Files are saved as they are edited, and you can jump to any moment in time. Need that file as it was on Feb 23, 2007? No problem.
- **Synchronization.** Lets people share files and stay up-to-date with the latest version.
- **Short-term undo.** Monkeying with a file and messed it up? (That's just like you, isn't it?). Throw away your changes and go back to the "last known good" version in the database.

- **Long-term undo.** Sometimes we mess up bad. Suppose you made a change a year ago, and it had a bug. Jump back to the old version, and see what change was made that day.
- **Track Changes**. As files are updated, you can leave messages explaining why the change happened (stored in the VCS, not the file). This makes it easy to see how a file is evolving over time, and why.
- **Track Ownership.** A VCS tags every change with the name of the person who made it. Helpful for blamestorming giving credit.
- **Sandboxing**, or insurance against yourself. Making a big change? You can make temporary changes in an isolated area, test and work out the kinks before "checking in" your changes.
- **Branching and merging**. A larger sandbox. You can **branch** a copy of your code into a separate area and modify it in isolation (tracking changes separately). Later, you can **merge** your work back into the common area.

Shared folders are quick and simple, but can't beat these features.

The below are some of the main elements of version control, check through them and talk them through with your partner or instructor if you're unclear:

- Repositories
  - A **repository** is the basic unit of GitHub, most commonly a single project. Repositories can contain folders and files, including images – anything your project needs. They **should** include a README and a license
- Branches
  - **Branching** is the way to work on different parts of a repository at one time.
  - When you create a repository, by default it has one branch with the name master.
- Commits
  - On GitHub, saved changes are called **commits**. Commits are pretty glorious, because a bunch of them together read like the history of your project.
  - Each commit has an associated **commit message**, which is a description explaining why a particular change was made. Thanks to these messages, you and others can read through commits and understand what you've done and why.
- Issues
  - An **Issue** is a note on a repository about something that needs attention. It could be a bug, a feature request, a question or lots of other things.
- Pull Requests
  - Pull Requests are the heart of collaboration on GitHub. When you make a **pull request**, you're proposing your changes and requesting that someone pull in your contribution - aka merge them into their branch.

Task 1 : Work through the exercise on "Using Git & the Github Desktop App" in the attached PDF with your partner.

Putting your site online

We're going to use GitHub Pages to host our site - it will provide the server where the files for our site is stored. This will be in the Homework.

GitHub provides a developer pack from students as well, here, including discounts on DNS management, a GitHub account upgrade (for free), Microsoft Visual Studio, and one year free on a domain name from namecheap.

The Command Line.

The command-line interface, sometimes referred to as the CLI, is a **tool** into which you can type text commands to perform specific tasks—in contrast to the mouse's pointing and clicking on menus and buttons.

Since you can directly control the computer by typing, many tasks can be performed more quickly, and some tasks can be automated with special commands that loop through and perform the same action on many files—saving you, potentially, loads of time in the process.

The command line is simply a place where you **type commands directly to the computer**.

The computer will take these commands at face value, and will attempt to carry out any command that it understands. Unfortunately, the computer does not speak in any language spoken by humans (although it has recognizable elements). The people who created the operating systems you work on have thus created a standard set of commands that are built into the computer without having to be processed or compiled.

NOTE: The command line, as with all power, has its risks. You have the capability to instruct the computer to do anything it has the capability of doing. If you instruct the computer to erase all of your data, it will cheerfully proceed to do so. **Do not run a command just to see what it does.** Make sure you understand what the command is supposed to do first, especially if the command involves changing or removing files.

The application or user interface that *accepts your typed responses and displays the data on the screen* is called a **shell**, and there are many different varieties that you can choose from, but the most common these days is the **Bash** shell, which is the default on **Linux** and **Mac** systems in the **Terminal** application. By default, **Windows** systems

only include the anemic Command Prompt application, which has nowhere near the power of Bash, so for the purposes of this article we're going to suggest you use the open source **Cygwin** tool as your Windows command line, since it's quite a bit more powerful.

Resources

This Guide , This Guide was given last week

A Mac-Specific Guide

Git

We learned about Version Control Systems earlier in the course, and have already been using GitHub. So technically, you do already know how to use version control and git, but you've only been doing it via a GUI - the GitHub Desktop Client.

Git is a very powerful and popular version control system, and the only place you can run **all** Git commands is from the command line.

Most GUIs only use a subset of all functionalities for simplicity, so if you know how to run the command line version, you can probably figure out any GUI versions. Command-line tools are also available across operating systems, whilst GUIs are subjective for different companies or teams. So it's best to understand it for the future.

Task:
Try Git here.
We recommend that you follow this guide to try it yourself in your own command line, the GitHub guide for this is here.

More Guides

A Git Cheatsheet from GitHub
A Cheatsheet & Reference from Git
A Non-Programmer's Guide to Git
Pro Git
Think Like a Git
GitHub List of References
A Visual Git Guide

CODEFIRST: Professionals

Web Fundamentals - Session 3

Homework

Putting your site online

Tasks:

1. Complete the guides at the top of the notes and finish them if you haven't already.
2. Use your GitHub account to follow the guides here and here.
3. Make some **changes** to your first_site based on what you've learnt. Update the online version of your site, by re-uploading to GitHub Pages using the GitHub Desktop Client.

Your site will currently show up with a the url of [your-github-username].github.io. In order to use your own domain name, you have to buy one from a domain name registrar, as we covered in the previous section.

Resources

How-To Geek explains GitHub , GitHub Guides

An introduction to Version Control

Another (Visual) Introduction to Version Control