GETTING STARTED WITH GIT

AARON HOOVER & BRAD MINCH

JANUARY 17, 2017

Much of this document was blatantly cribbed from Allen Downey's fork of the book Pro Git called Am Git.

1 *Why use a version control system?*

In this class you will be working in teams. For the miniprojects, they may be small teams of two, and for the final project, they may be teams as large as five. You will also be writing a lot of code - C code for the PIC microcontroller and Python (or a language of your choice) on your laptops. If you've ever tried to manage collaborative work on a project using tools like Dropbox, folders on Public, or email + attachments, then you know what a headache it can be to keep track of files, share revisions, and generally ensure that things don't get mucked up. A version control system provides the following features and advantages that make it helpful in these situations:

- The history of files is tracked and readily viewed so that you can know who edited what when.
- It is possible to revert a file to an older version in the event that change causes problems.
- Files are typically stored both locally and remotely in a repository on a professionally maintained server, protecting against catastrophic loss of data.
- Copying and sharing a repository is easy, making collaboration easy.
- The repository can help people on a team coordinate their work.
- Some version control systems include tools with added functionality like the ability to track and assign issues.

2 *Installing Git*

Git is the name of the version control system we will use in this class. **Github** is a website that hosts remote Git repositories and provides different interfaces and tools for those repositories.

Windows

There are a few options for using git on Windows. Probably the easiest to help you get up and running is GitHub Windows available at http://windows.github.com. GitHub Windows includes a simple GUI as well a command line tool (that uses PowerShell by default) for using the git commands.

Mac

It is probably easiest to install git on OSX from a terminal using Homebrew by running

```
>brew install git
```

You can install homebrew by following the instructions at http://brew.sh/.

3 *First Time Setup*

Identity - You will need to configure Git with your username and email address because these pieces of information are appended to all of your commits. The first time you open GitHub windows, it prompts you for this information, otherwise you can enter it manually from the command line as follows:

```
>git config --global user.name "Aaron Hoover"
>git config --global user.email "aaron.hoover@olin.edu"
```

Editor - Many a flame war has erupted over which text editor is the best for writing code. We will not fuel the fire here. However, sometimes Git has occasion to open a text editor (to add comments to a commit, for example). To get it to automatically open your favorite editor you can execute the following from the command line:

```
>git config --global core.editor my_favorite_editor
```

Check Your Settings - You can inspect your current Git settings with the following:

```
>git config --list
```

Getting Help - From time to time, you may need to be reminded of what Git commands do or even what they are. You can use any of the following:

```
>git help git
>git help <verb>
```

For example,

```
>git help push
```

will show you the manual page for the `push` command.

4 *Create a Github Account*

Go to Github and sign up for an account.

5 Forking and Cloning a Repository

5.1 Forking

We have set up a repository for the class on Github. The first operation you need to perform is to "fork" this repository. This will create a copy of the repository that you own under your Github user account. Any changes you make to the code and commit will show up in your copy of the repository, not in the one that belongs to the OlinElecanisms organization. To fork the repository, navigate to the URL and click the "Fork" button in the upper right-hand corner of the page.

5.2 Cloning

Now, you will need to "clone" the repository to a location on your local file system. Cloning creates a local repository on your machine. You can clone within GitHub Windows by clicking on the little "+" sign in the upper left and selecting the "Clone" option on the resulting screen.

To clone the repository from the command line, change to the directory where you'd like to store the code and execute:

```
>git clone https://github.com/<your username>/elecanisms
```

You may name the repository something other than "elecanisms" by appending an optional argument to the end of the `git clone` command.

6 Editing, committing, and pushing changes

6.1 Editing

Open the `blink` subdirectory and open the file `blink.c` in the editor of your choice. Edit and save the file. In GitHub Windows, you'll notice a heading appears that says "Uncommitted changes," meaning you have made changes that are not yet committed to your repository. Expanding the heading shows all the files with changes and highlights the changes for you.

You can also check for uncommitted changes from the command line:

```
>git status
```

The output should look something like this:

```
# On branch master
```

```
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified:   blink.c
#
no changes added to commit (use "git add" and/or "git commit -a")
```

6.2 *Committing*

Before you can commit them to the repository, you need to "add" your changes to the commit. In the GUI, this is accomplished simply by checking the checkbox next to the name of each file whose changes belong in the commit.
To do it using the command line (with blink.c) we execute:

```
>git add blink.c
```

Now, running `git status` displays:

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:   blink.c
#
```

To actually commit from the GUI, we need to enter a description of the changes in the "Summary" text box at the top of the "Uncommitted changes" area. Once a summary has been entered, a button with a check mark on it next to the text, "Commit to master" is enabled. Clicking that button commits the changes to our local resposiitory

From the command line, we can run:

```
>git commit -m 'Sample commit message'
```

Your commit messages should be concise and informative so that the changes made as part of a commit are apparent immediately to anyone reading the message.

6.3 *Pushing*

We are almost finished. We have committed the changes to our local repository, but we still need to update the repository stored on Github because this is the repository that is shared by our teammates.

In the GUI, simply click the "Sync" icon in the upper right corner of the application.

To do that from the command line, run:

```
>git push origin master
```

The `push` command takes a destination and a source as its two arguments. `origin` refers to the remote repository from which the local repository was cloned. `master` is the name of the branch we committed our changes to. We haven't discussed branches yet, but in our case we have only one branch which, by default, is called `master`.

7 *Typical Workflow*

The most typical workflow after you have cloned a repository is shown below.

7.1 *GUI*

1. Click the "Sync" button to get all the latest changes from the remote repository since your last sync.
2. Modify files as needed.
3. Show uncommitted files and select the changed files you intend to commit.
4. Enter a commit message in the "Summary" field.
5. Click "Commit to master" to commit to your local repository.
6. Click "Sync" to push your local changes to the remote repository.

7.2 *Command line*

1. `git pull`: Gets changes made to the remote repository since your last `push`
2. Modify some files
3. `git add <files>`: Stages changes to be committed
4. `git commit`: Commits changes to the local repository
5. `git push origin master`: Copies committed changes to the remote repository