Git and Github

Your Version Control Friends

Acknowledgements

Links and information taken from Deep Dive Full Stack bootcamp materials that were prepared by:

- Dylan McDonald
- Paul Schulzetenberg
- George Kephart
- Marty Bonacci

What is a version control system?

Version Control Systems (VCS) record changes made to files so that you can

- compare and track changes over time,
- revert single files to a previous state,
- or even revert an entire project to an earlier version.

Git is a VCS, or Version Control System.

Similar to Backup on Windows or Time Machine on the Mac.

Why use a version control system?

The good ‘ol days.

- Save multiple versions of the file and try to remember which is which.
- Run a text comparison tool to see what changed between versions.
- Work with system engineer to get your changes merged with production version.

With Git.

- Commit as you code.
- Versioning and timestamping auto-”magically” happen.
- Easy to see differences between versions using git diff or visually if using Github.
- Easy to merge changes to production or rollback versions.
- Links with ticketing system for better task management and customer support.

Git vs. Github

Git is the version control system. This is on your local system.

⇨ git commit

Github is a GUI cloud-based product that allows you to save your work remotely and facilitates collaboration.

⇨ git push

Commit vs. Push - Muy Importante!

Commit

- File saved to **LOCAL** repository. File is given a timestamp and unique commit number that is the file version.

Push

- File is saved to **REMOTE** repository.

Remember to Push your files in order to save from data loss and to ensure your work is available for collaboration.

When should you commit and push your work?

When to commit? **Often!**

- Any time you finish a task where you want to save or retain a version.

When to push? **Often!**

- Any time you have finished a task, milestone, or significant project.
- At the end of each work session
  - Before you take a break
  - Before a meeting
  - Before lunch
  - Before you go offline for the day

Installing git on Windows

There are two good ways to install Git on Windows.

- The official build is available at git-scm.com/download/win.
- You can also use the GitHub GUI tool at windows.github.com. During the installation of this tool you can choose to install command-line tools as well (i.e. Git's command-line interface).

Installing git on Mac

There are three ways to install Git on Mac.

- A git installer is available at git-scm.com/download/mac.
- You can download the GitHub GUI tool at mac.github.com. During the installation of this tool you can choose to install command-line tools as well (i.e. Git's command-line interface).
- You can install the Xcode Command-Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run git from the Terminal the very first time. If you don’t have it installed already, it will prompt you to install it.

Installing git on Linux

You can use the package management system that comes with your distribution.

On Debian based distributions like Ubuntu, use apt-get:

```
sudo apt-get install git-all
```

Using git - command line or GUI?

Why should I use the command line when I could use a GUI instead?

- It's generally better supported than GUI-based tools.
- It's independent from your IDE.
- It helps you come to a better understanding of the principles of version control.
- It's much more commonly used in the industry and thus is more likely to get you a job.
- It's much more likely to be useful if you find yourself in a sticky merge/rebase situation that you can't seem to fix.
- It’s faster to type the commands than to go through the GUI. These time savings add up.

The command line

Windows

- PowerShell - This is what we are using in class.
- GitBash → linux commands and editors available.

Mac or Linux

- Terminal - This is what our Mac users are using in class.

For basic commands, review IoT slides on “Command Line Interface”

The repository - aka the repo

- The git repository is the location where files are saved/staged.
- The git repository is also called a “repo”.
- You may create a new repository from scratch.
  - Your smart room controller, flower pot, and capstone projects
- You may clone a local repository from an existing remote repository.
  - Your class exercises, a 3rd party library
- The git repository keeps your version history in a hidden “.git” folder.
  - To view on Windows, in your project folder in File Explorer, click View, then check “Hidden Items”
  - To view in Powershell, in your project folder, type dir -Force.
  - To view on Mac, in your project folder in Finder, press Shift + Control + dot

Creating a project from scratch from Github

In this scenario, you will create a repository first in Github, then clone it to your local system for use.

Practice: Create a repo in Github

- Login to your github account. If you do not have a github account, then create one. [Github.com]
- On your github home page, click “New” to create a new repository.
- Type the following name for your new repo.
  - gitdemo_yourname
- Indicate if the repo is public or private.
- Select a readme, .gitignore, license option if applicable.
- Click “Create Repository”
- Your new repository is created.

Practice: Clone the repo to your local machine

- Open your new repository in github if it is not already open.
- Click the “Code” button.
- Copy the https address for your github repo.
  - Note: there is also an SSH option if you choose to link your public SSH key to your github repository. https://docs.github.com/en/github/authenticating-to-github/about-authentication-to-github
- cd into your IoT directory.
- git clone the https github repo to initialize the git repo on your local machine. This creates your project folder and the hidden .git folder.

Practice: Committing and pushing changes

- Go back to your repo directory on your computer.
- Create and add a file.
  - You can use notepad or textedit or create it on the command line.
- Make some changes to the file.
- Commit and push the file.
- Make some more changes to the file.
- git diff [filename] to see the changes that will be committed.
- Commit and push the file.
- Go to github and view “commits”
- See how github displays your changes.

Practice: View changes

- Go to your project directory
- git log - to view version history
- Copy one of the commit numbers to your clipboard
- git show [commit] - to view content changes for that commit

The .gitignore file

There may be times you want to omit directories and/or files from git.

Examples:

- Files with sensitive information such as password/credential files.
- Directories with 3rd party libraries. These will likely already be installed on the deployment server so do not need to go to Github.

Such files and directories are listed in a .gitignore file that usually resides at the root of your project directory.

Practice - creating and using a .gitignore file

- Open your test repo.
- Add a file called credentials.h. Type something into the file and save it.
- Type git status. What do you see?
- Now add a file called .gitignore.
- In the .gitignore file, type credentials.h and save.
- Type git status. What do you see?
- Commit and push to github.

But what if I COMMIT my sensitive data accidentally?

Suppose you accidentally commit the sensitive file anyway?

https://www.atlassian.com/git/tutorials/undoing-changes

Note: there are many different scenarios. So if you run into a situation where you need to revert a commit, check the above documentation and/or check procedures with your company to see how they want to resolve.

But what if I PUSH my sensitive data accidentally?

Suppose you accidentally PUSH the sensitive file anyway?

FIRST CHANGE THE PASSWORD OR INFORMATION IMMEDIATELY.

Then follow steps to remove the file from your git repository. *Note: deleting the file from your repo does not remove it completely*.

- If you are a solo-preneur or hobbyist, then you can decide how to resolve the issue.
- If you are working for a company, then check with your boss to see how to resolve.

https://stackoverflow.com/questions/872565/remove-sensitive-files-and-their-commits-from-git-history

Typical Github workflow and branches

- Master is the good code. This is generally the code that is also available on a production system.
- A production system is a live system that people are actually using.
- You don’t want to push unfinished, untested code to your master branch.

Typical workflow:

Feature → development → staging/testing/release → master/production

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Practice: Branching and merging code

- Create a branch called “dev”
  - Git branch dev→ create the new branch
  - Git checkout dev→ switch to the new branch
  - Git branch → confirm the branch you are now in.
- Make changes to your test file.
- Commit and push your changes.
- git checkout master - to switch back to the master branch.
- git branch - to see what branch you are in.
- git merge dev- to merge your dev branch changes into master.
- git push - to push the merged changes to master
- Look at your commit history to see how the merge was handled.

Why bother with branches? Scenario 1

You and your teammate are working on your capstone project.

Your teammate is writing code for one group of user stories (requirements).

You are writing code for another group of user stories (requirements).

Your teammate checks in their code.

You pull the code.

Aghgh! Your code stops working.

With branches, you can both work on separate pieces of code and then ONLY commit your code to the master branch after it is tested and working. Yay!

Why bother with branches? Scenario 2.

You are presenting your capstone project in an hour. You decide to make some last minute code changes. Then you test your project and discover that it no longer works!

That is ok. You made your last minute changes in your dev branch. Yay!

You switch back to your master branch, pull the code, test your project and it is working again. Double Yay!

You confidently present your working project. The audience is impressed. You are offered an exciting, high-paying job based on your presentation. Triple Yay!

Final words of wisdom

Pay attention to what you are committing and pushing! This is especially important if you are using a public repo.

- Be careful about committing sensitive data files with passwords, personal email addresses, IP addresses, account data, API keys. Use .gitignore.
- When collaborating, keep unfinished code to your personal feature or dev branch.
  - Only push/merge with master when code is tested and working.
- And commit/push your work often!
- Don’t get hacked. Enable 2-factor authentication and set a complex password on your Github account.

Quiz

When should you commit and push your work?

Often.

Regularly.

Several times a day.

Any time you have finished a task.

Additional Resources

- Installing git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- git cheat sheet: https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf
- Github documentation: https://docs.github.com/en/github
- Git workflow: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
- What to do if it goes all wrong: https://ohshitgit.com/
    - or curse free https://dangitgit.com/

Deep Dive Coding Full Stack Git Lecture Notes

- Deep Dive Coding (DDC) Full Stack lecture on git: https://bootcamp-coders.cnm.edu/class-materials/git/

Additional Reading

Abridged history of git:

https://hackernoon.com/how-git-changed-the-history-of-software-version-control-5f2c0a0850df

The End