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.

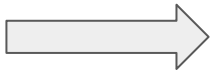
- Save multiple versions of the file and try to remember which is which.
- Run a text comparison tool to see what changed between versions.
- Work with system engineer to get your changes merged with production version.

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
- GitBash → linux commands and editors available.

Mac or Linux

- Terminal

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.
  - git init for local repo.
  - Create in Github for remote repo.
- You may clone a local repository from an existing remote repository.
- The git repository keeps your version history in a hidden “.git” folder.
  - To view on Windows, in your project folder in Windows 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.

This should be your more common scenario since you should intend to capture all changes on a remote system for redundancy.

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 a name for your new repo.
- 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.

Creating a new project locally then linking to Github

In this scenario, you will first create a local repo, then link it to a repo in Github.

This scenario may occur if you start a project on your local machine and then later decide you want to track your changes in Github.

Practice: Linking local repo to Github repo

- Make a new directory named gitdemo2 in your IoT directory.
- Add a file to the directory called firstfile.txt.
- Open the file in notepad or a text editor and add a couple of lines of text.
- Save the file.
- git init → to create a local repo. You can see a .git directory now exists.
- git add and commit the changes to your repo.
- git push → what happened?
- Create a new repo called gitdemo2 in Github. Get the repo link.
- git remote add origin <your repo link here>
- git branch --set-upstream-to=origin/master
- git pull
- git push

Thoughts on creating local before remote

There are more steps to remember in order to link a local repo to a remote Github repo.

It is easier to create the remote first, clone it, then start developing.

Recommend creating remote repo first to save steps.

Practice: Committing and pushing changes

- Go back to your gitdemo directory on your computer.
- Create and add another file.
- 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

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 “develop”
  - Git branch develop → create the new branch
  - Git checkout develop → 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 develop - to merge your develop 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.

Collaboration in github

- Collaboration allows multiple people (github accounts) to work on the same project.
- For your capstones, you will need to add classmates as collaborators to your repo.
- Applying rules to the github workflow helps avoid merge conflicts and problem code.
    - Have team members work on different files.
    - If team members are working on the same files, then communicate those changes to reduce possibility of conflicts.
    - Enforce peer reviews on “pull requests” to reduce chance of committing bad code.

Practice: Collaborating

- Add your teammate as a collaborator to your repo.
  - Settings > Manage Access > Invite Collaborator
- Approve the invitation to collaborate in your email.
- Confirm that you can now see your teammate's repo.
- Make a directory in your IoT folder for your teammate's repo.
- Clone your teammate's repo.
- Checkout the develop branch.
- Create a new branch off the develop branch on your teammate's repo;
  - git branch yourname-feature
  - git checkout yourname-feature
- Make a change to a file on your feature branch.
- Commit and push your change.
- Have your teammate confirm that they can see your new branch and change in their repo.

Instructor note: git config --global user.email <your cnm email>

Pull Requests

A pull request is a safety feature in Github that enforces code review before merging code from a feature branch into an upstream branch.

A pull request is initiated in the Github GUI.

Note: to do this on the command line, you would need to install a 3rd party product that supports this functionality.

To enforce pull requests before merge,

- Go to Settings > Branches > Add Rule.
- Choose “Require pull request reviews before merging.”
- Click “Create” to save the rule.

Caveat! You need a pro account to totally prevent merges

Note: even though you require collaborator approval on a pull request, the developer can still merge to another branch and bypass the pull request process.

So, be a team player and utilize pull requests in Github to notify your teammates that your code is ready for review and merge.

Practice: Approving and merging pull requests

- Create a pull request for your feature branch in Github
- Have your teammate review and approve your pull request
- Have your teammate merge your pull request into the develop branch of your repo.
- View the result in Github.

Git and Github Part Deux

Let’s Review

- What is Git? What is Github?
- How do you create a new repo?
- How often do you commit and push your code?
- What is the difference between commit and push?
- List the steps to send your code to Github.

Using git inside an IDE - Visual Studio Code

You can run git commands from the Visual Studio Code terminal. Advanced IDEs usually integrate with a VCS like git.

- Open a project folder in VS Code
- Open the terminal at the bottom of VS Code
- Choose you preferred git command tool
  - Windows Command
  - Windows Powershell
  - Git Bash -> like Linux
- Now type your git and file/directory commands in the terminal
- This is easier than Powershell because you are already in your project folder so VS Code can show your git status in the IDE.

Review: Git Commands on your Particle Project

Git log - shows commit history for a repo

Git show <commit number> - shows some details for a specified commit

Git diff - shows differences between file versions

Git status - shows what is staged for commit

- VERY helpful to know what will be committed BEFORE you commit it.

Git branch - to create a new branch and see what branch you are currently in

Git checkout - to rollback files to the current git version

Git switch <branch name> - to change branches

Other git features

If you have a need to use other features, you can view the documentation. Search for “git man page”.

https://git-scm.com/docs/git#_git_commands

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 L14_03_SubscribePublish project in VS Code.
- In your src folder, add a file called secrets.h
- Move your two lines of code for user name and api key into the secrets.h file and save.
- Type git status. What do you see?
- In your src folder, add a file called .gitignore.
- In the .gitignore file, type secrets.h and save.
- Type git status. What do you see?
- Compile your code.
- Commit and push to github.

But what if I commit my sensitive data accidentally?

Suppose you accidentally commit the secrets.txt file anyway?

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

You can find many posts online about how to undo a commit.

Your Capstones - Collaboration in Github

One person will be the keeper of the Github repo for your Capstone.

That person will add other Capstone teammates to the Github repo.

As a team, you will decide who will be allowed to merge changes into the master branch.

As a team, you will decide who works on what functionality. This is where header files will be useful.

Collaboration - Avoiding problems

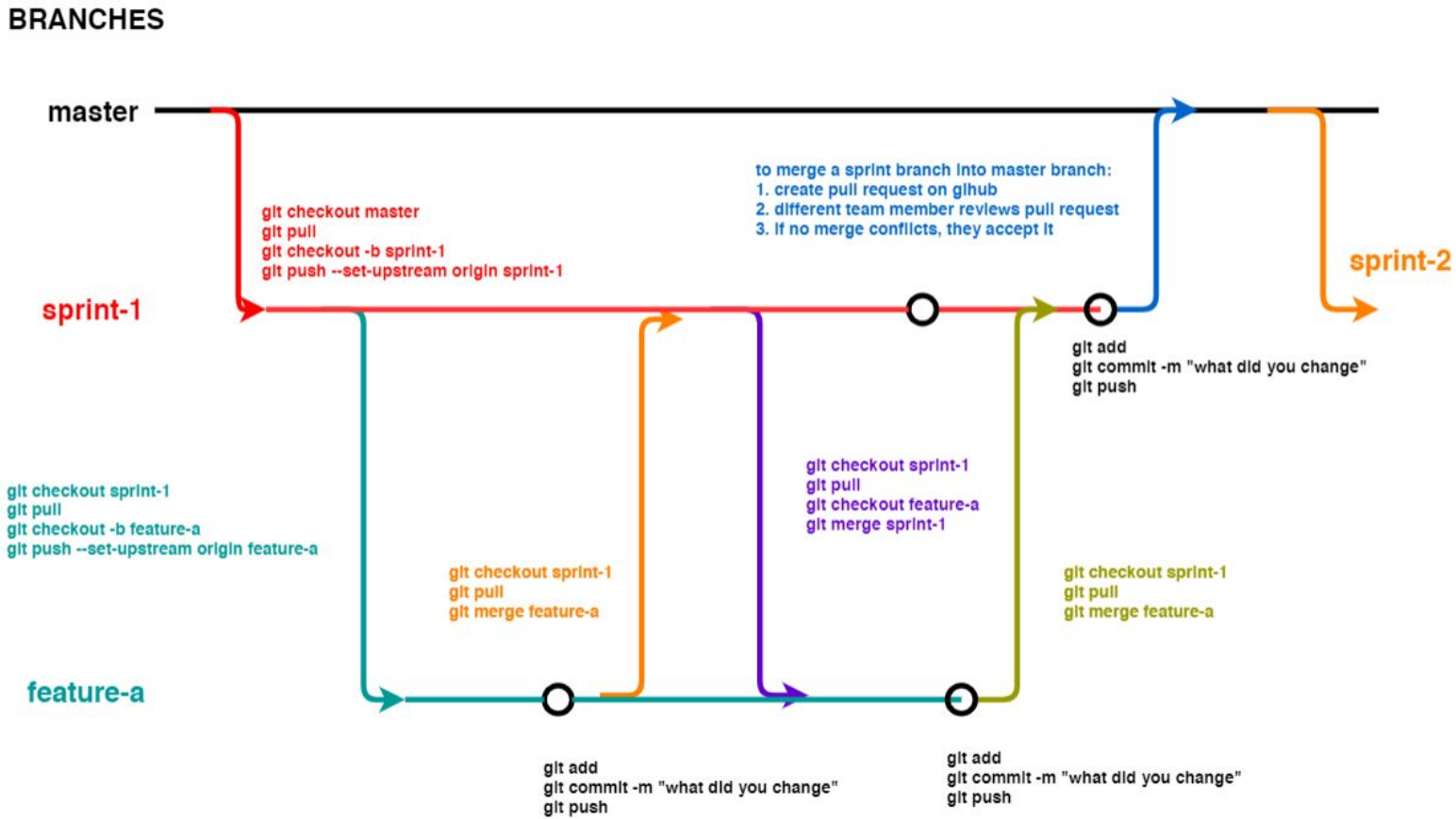
- Communicate with other team members about changes you are making
- Segregate assignments if possible to avoid changing the same code at the same time.
- **Always do a git pull from a branch** before merging to avoid having your branch ahead of a higher level branch.
- Use a separate branch for development to avoid breaking the master branch or your teammates' code.
- Thoroughly test your code before committing and pushing to Github.
- Have a teammate do a code review before merging to a higher level branch.

Collaboration Workflow

1. Clone the shared repo to your local machine.
2. If you do not have a development or working branch, create one.
3. Switch to the new branch in the repo.
4. Do your development.
5. Commit and push to Github.
6. Notify the repo owner that your code is ready for review and merge to master.

Note: design a workflow that works for your team. Do NOT actively work on the master branch EVER when working collaboratively.

git workflow - advanced merging from feature branch multiple times



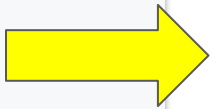
Key points on the collaborative workflow

- Checkout branch you wish to work from.
- **Git pull → VERY IMPORTANT!**
- Now create your working branch off the branch you checked out (if you don’t want to corrupt the branch you are on)
- Do your work on your new branch; commit and push.
- Switch to the parent branch.
- **Git pull → TO GET ANY CHANGES THAT YOUR TEAMMATE MAY HAVE COMMITTED SINCE YOUR LAST PULL**.
- Git merge your branch into the parent branch.

The dreaded merge conflict!

In spite of your best efforts, you will eventually encounter a merge conflict. This occurs when git simply cannot figure out how to merge two files because the changes are too disparate.

```
$ git status
> # On branch branch-b
> # You have unmerged paths.
> #   (fix conflicts and run "git commit")
> #
> # Unmerged paths:
> #   (use "git add ..." to mark resolution)
> #
> # both modified:      styleguide.md
> #
> no changes added to commit (use "git add" and/or "git commit -a")
```



```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```

Fixing a Merge Conflict

- Open the conflicted file in a text editor or VS Code or whatever IDE you are using.
- Notice the merge conflict markers in the file.
- Select the text you want to keep.
- Remove the text you want to delete.
- Delete the conflict markers.
- Save the file.
- Commit and push to github.
- Go to Github and confirm that the develop branch now has the corrected file.

```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```

https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line

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 development 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