Source Control (Git and GitHub)

CREATE BEST 2018

Presented by: Miguel Garzon

Overview

- Overview:
  - Key Concepts
  - Installation and Setup
  - GitHub Introduction
  - Basic Git WorkFlow
  - Utility Commands

THE BASICS

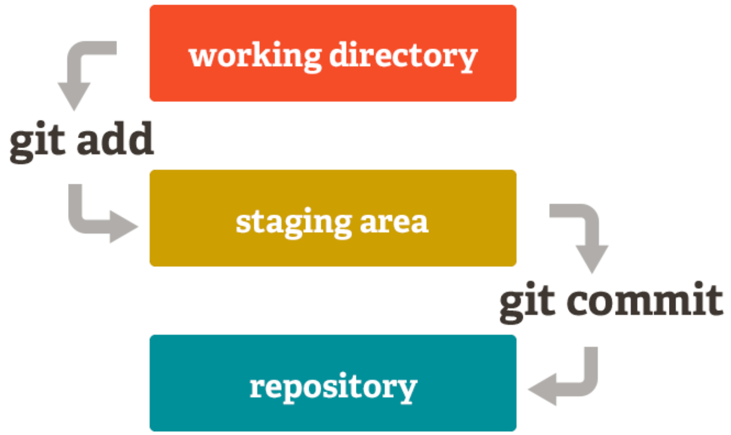
Source Control? Version Control?

- Management tools used to manage changes to files
- Isolates developers from one another while they work
- Allows for selective and predictable integration of different portions of a project
- Can be directly integrated into development platforms like Android Studio
- Common Version Control Systems (VCS):
  - SVN: Very Simple, single repository shared by all users.
  - Mercurial: Medium Complexity, distributed repository system.
  - Git: Medium-high Complexity, distributed repository system.

Why Git?

- Distributed source control system
  - Most operations in Git are local
  - Only a few commands required a network connection
- Massively scales
- Open Source
- Developed for Linux project requirements
  - 15 million lines of code
  - 12000 worldwide developers contributing to project
  - Dozens of active branches
- Fast, FREE and Open Source!
- Very active community!

Core Concepts

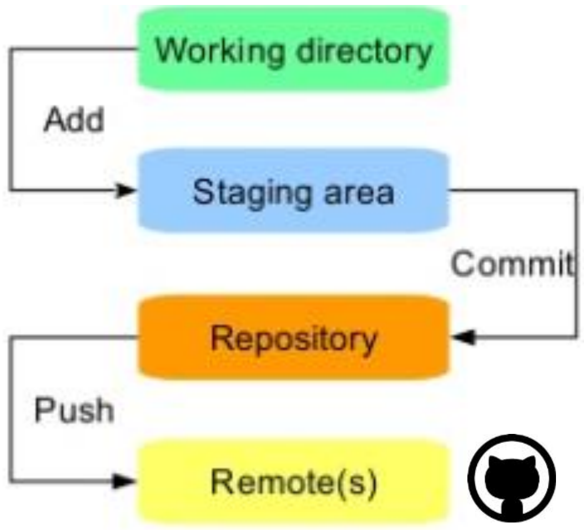


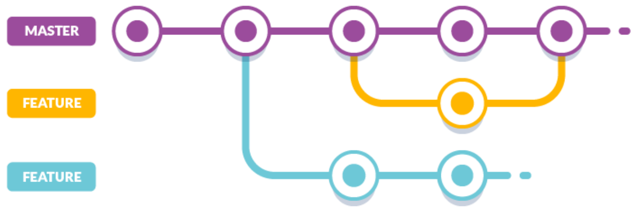
- **Repository contains:**
  - Files
  - History
  - Special configurations

- **Three (local) States of Git**
  - **Working directory** is the directory or folder on your computer that holds all the project or application files. *It's a copy of the remote repository in which you add/remove/edit your files*
  - **Staging Area** (holding area or Git Index): Queues up all changes for the next commit
    - Files can be put into the staging area if they have been modified
    - Files in the staging area can be moved in and out
  - **Repository** – Git Repository (.git folder)

Core Concepts (cont'd)

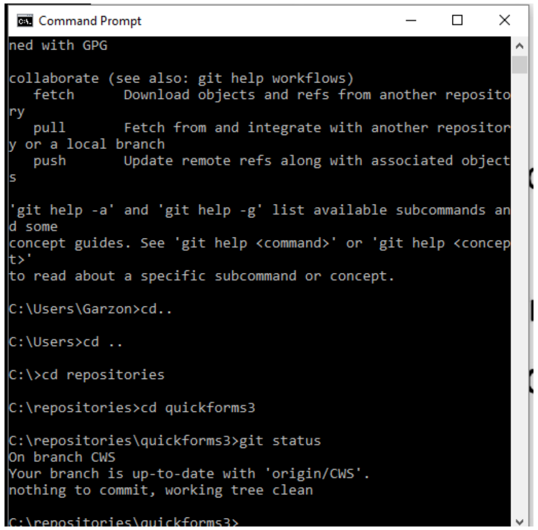
- **Remote Repository:**
  - The remote repository is just another repository with its own three states
  - Last step in the basic Git Workflow
- **Master Branch**
  - Default branch that contain your changes
  - Git provides a default branch named **master**





Git - Command Line

- Git was originally designed as a command line tool, and graphical support came later.
- Features make it onto the command line before they are integrated into a graphical client
- Documentation
- More power!
- **Consistency**:
  - **Terminal** on Mac/Linux
  - **Git Bash** on Windows



```
ned with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another reposito
ry
   pull       Fetch from and integrate with another repositor
y or a local branch
   push       Update remote refs along with associated object
s

'git help -a' and 'git help -g' list available subcommands an
d some
concept guides. See 'git help <command>' or 'git help <concep
t>'
to read about a specific subcommand or concept.

C:\Users\Garzon>cd..

C:\Users>cd ..

C:\>cd repositories

C:\repositories>cd quickforms3

C:\repositories\quickforms3>git status
On branch CWS
Your branch is up-to-date with 'origin/CWS'.
nothing to commit, working tree clean

C:\repositories\quickforms3>
```

Git Installation

- Windows
  - https://git-scm.com/download/windows
- Mac OS X
  - https://git-scm.com/download/mac
  - Or type git version on the terminal
- Linux
  - Use your package manager to install git, e.g.: "apt-get install git"

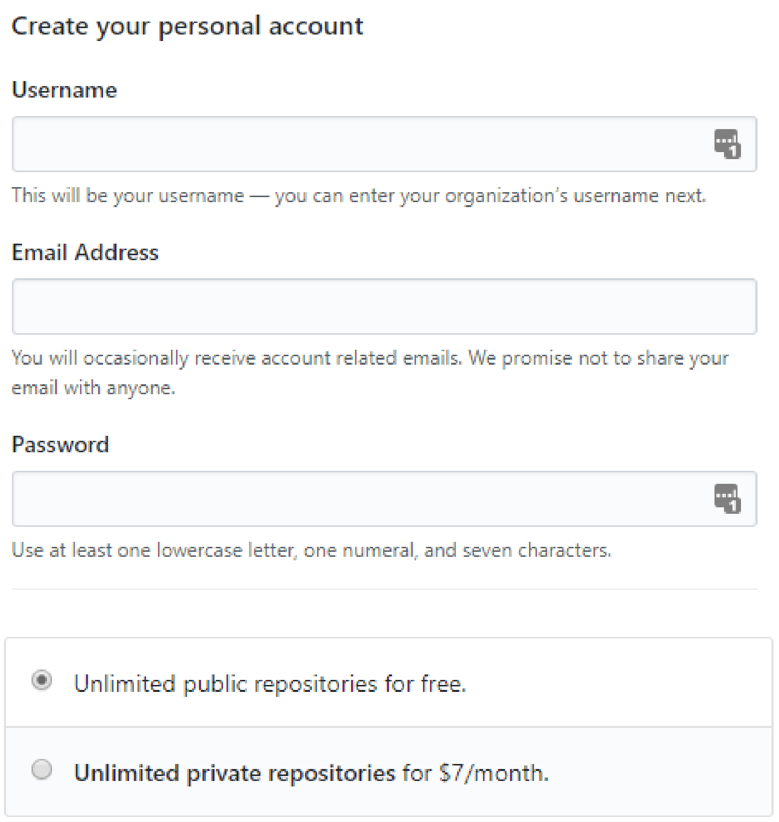
Git Commands are the same!

Test your installation by typing: **git version**

GitHub – First Repository

- Navigate to GitHub.com
  - You can sign in or sign up.
  - Select the ***'Unlimited public repositories'*** free plan if you sign up.

- Create a public repository:
  - **Name**: **my_first_github**
  - **Description**: A simple demo to show the basic Git Workflow;
  - Select the option **'Initialize** this repository with a README'



Create your personal account

Username

This will be your username — you can enter your organization's username next.

Email Address

You will occasionally receive account related emails. We promise not to share your email with anyone.

Password

Use at least one lowercase letter, one numeral, and seven characters.

[✓ Unlimited public repositories for free.]

Unlimited private repositories for $7/month.

Prepare our local system

- Create a folder called **projects** using the command line
  - Place yourself under c:\
  - Type: mkdir **projects**

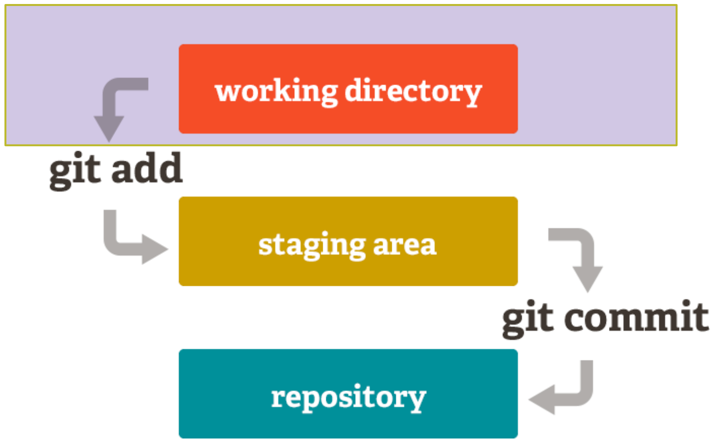
- Git requires two pieces of information before we proceed
  - **Name**: git config --global user.name "Miguel Garzón"
  - **Email**: git config --global user.email mgarzon@uottawa.ca
  - This information will be associated with every commit made by the user

Cloning a Repository

- Using the command line, place yourself inside the folder projects (c:/projects)
- Type: "git clone <name of the repository>"
  - **e.g.: git clone https://github.com/mgarzon/createbest-github.git**
- Confirm that a folder named has been created and that you have the README file (for the example above, the folder will be named 'createbest-github')
  - You can add a creation folder name after the repository name to specify the name of the folder you'd like to create
- Type: "git status" and pay attention to the information displayed.
  - If you cloned the repository correctly, you should have all commits and be in the repository head.

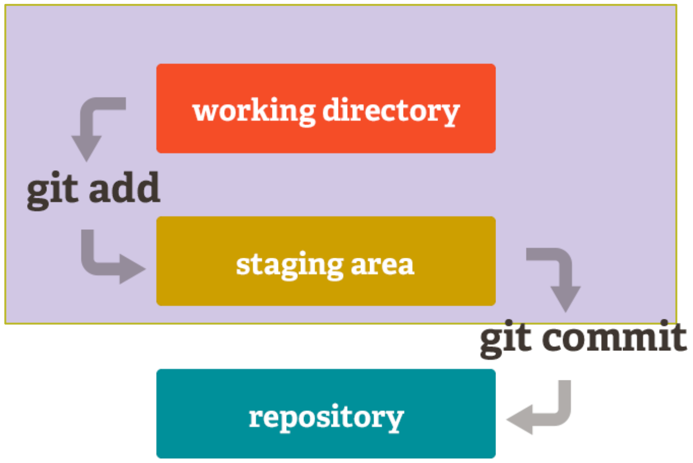
Our First commit

- Add a new file to your working repository directory
  - (you can create a new text file)
- Type ***git status*** in the command line. What do you see?
  - The file you created should be listed as "untracked"



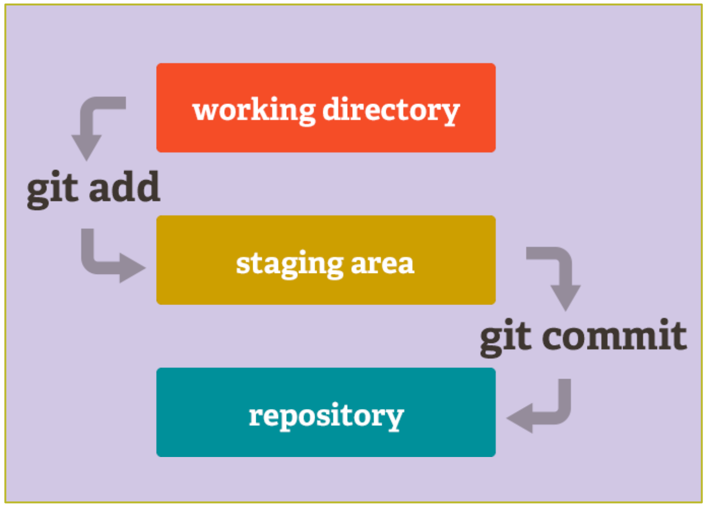
Our First commit (cont'd)

- Use the **add** command to add your file to the staging area.
  - E.g.: "**git add game.js**"
- Then type **git status** again. What do you see now?
  - The file you created should be listed in commit list



Our First commit (cont'd)

- Use the **commit** command to add the file you created
  - E.g.: " **git commit –m "first commit"** "
- Then type git status again. What do you see now?
  - You should now be in the master branch

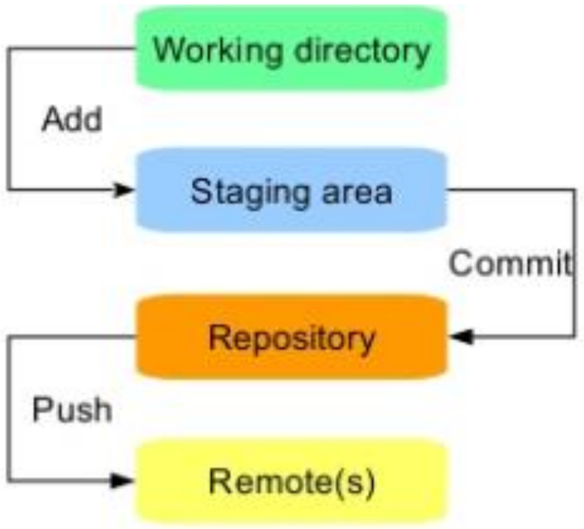


Publishing changes to GitHub

- Your previous commit is still a **local** command
- Use the **push** command to send your changes to the remote server
  - Type ***git push origin master***
  - **origin**: refers to the GitHub copy of our repository
  - **master**: refers to our default and only branch of the repository

- The git push command will prompt you for you GitHub **username** and **password.**

- **Check** your results on Github.



Make Modifications to Files

1. Modify the Readme.md by adding the following text: "Welcome to my Project ".

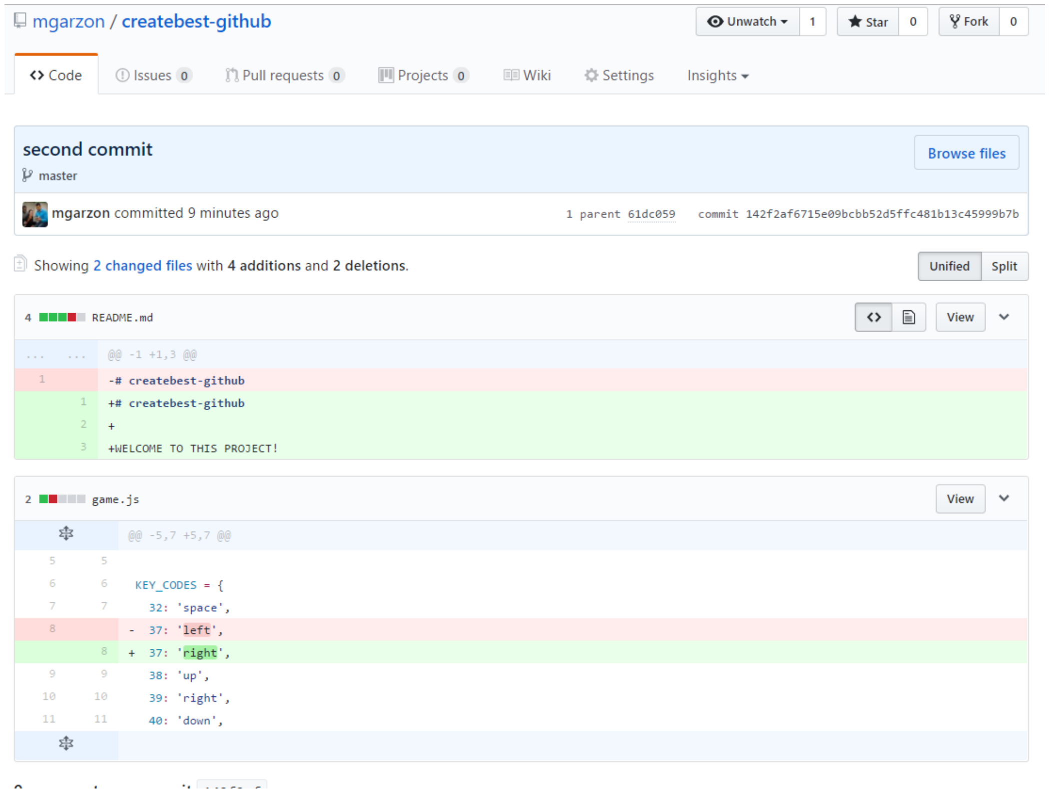
2. Modify the file you created, add some text to it.

3. Add, Commit and Push your changes. Commit message should be ‘Second commit’.

4. Commit multiple files at the same time:
   – git add .
   – git commit –m “second commit”
   – git push origin master

Comparing two commits

- Portions removed are marked in red and those added are marked in green.



second commit
master
mgarzon committed 9 minutes ago 1 parent 61dc059 commit 142f2af6715e09bcbb52d5ffc481b13c45999b7b

Showing 2 changed files with 4 additions and 2 deletions.

4 README.md

```
@@ -1 +1,3 @@
-# createbest-github
+# createbest-github
+
+WELCOME TO THIS PROJECT!
```

2 game.js

```
@@ -5,7 +5,7 @@

  KEY_CODES = {
    32: 'space',
-   37: 'left',
+   37: 'right',
    38: 'up',
    39: 'right',
    40: 'down',
```

Pulling Remote Updates

- If someone else has made changes to your project and has sent them to the remote server, in order to be able to synchronize, you need to run a **pull** command
- Trying to push commits when the remote contains changes not visible locally will cause the push to fail. You need to integrate the changes before submitting your own.
- If there are not conflicts, using "***git pull***" will download the outstanding changes without issue. Then, you will be able to **push** your own changes.
- *Cases where multiple team members change the same files will be treated following, during discussion of "merging".*

INTERMEDIATE CONCEPTS

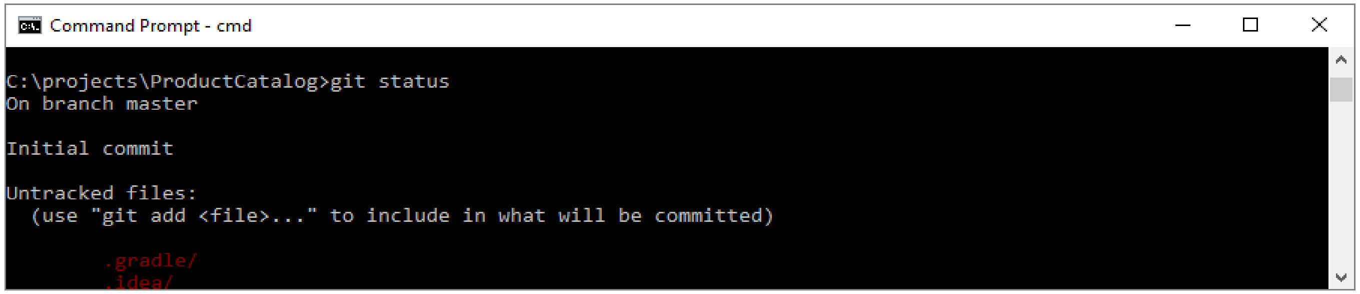
Starting With an Existing Project

1. Consider a project where you already have source code. If you have a project in your computer you want to use, make a copy of it in a separate folder. You can also create a new folder, create a test text file for this example.

2. To place this folder under version control with Git, we use the **init** command.
   git init .
   – At this point, Git has initialized an empty Git repository in the current folder.

3. Verify that a .git folder exists in the current folder.
   – Type **"dir /a"** in windows, or "**ls –la**" in linux/mac

4. Type ***git status*** to check the current state of the (local) repository.



```
C:\projects\ProductCatalog>git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gradle/
        .idea/
```

Commit Untracked Files

- Before committing our changes let's create a file called **LICENSE.MD**

Commit all files at the same time, remember to add a message:

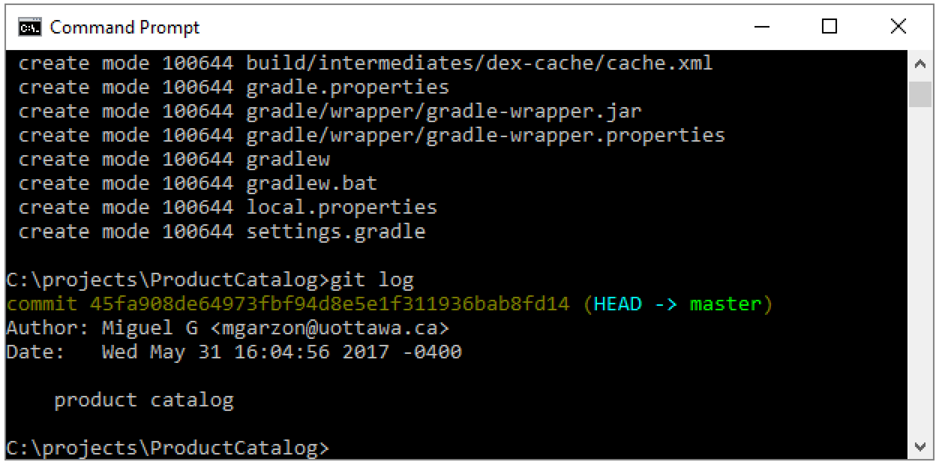
- "***git add*** *.*"
- *"**git commit –m "Created a repository from an existing project"*** "

<u>Don't push it! We still want to play with the staging area!</u>

Logs!

- Use the ***git log*** to see the list of commits made locally. The log contains:
  - A long identifier (SHA-1 identifier)
  - Author
  - Timestamp
  - Commit message

- ***git show*** gives you more details, give it a try.



Command Prompt

```
 create mode 100644 build/intermediates/dex-cache/cache.xml
 create mode 100644 gradle.properties
 create mode 100644 gradle/wrapper/gradle-wrapper.jar
 create mode 100644 gradle/wrapper/gradle-wrapper.properties
 create mode 100644 gradlew
 create mode 100644 gradlew.bat
 create mode 100644 local.properties
 create mode 100644 settings.gradle

C:\projects\ProductCatalog>git log
commit 45fa908de64973fbf94d8e5e1f311936bab8fd14 (HEAD -> master)
Author: Miguel G <mgarzon@uottawa.ca>
Date:   Wed May 31 16:04:56 2017 -0400

    product catalog

C:\projects\ProductCatalog>
```

Making changes to Staging area that will be reverted

- **Make** any changes to the **LICENSE.MD** file (add some text to it)
- **Add** the changes to the staging area

  "git add LICENCE.MD"

- **Unstage** your changes ☺ (Unstage the file)

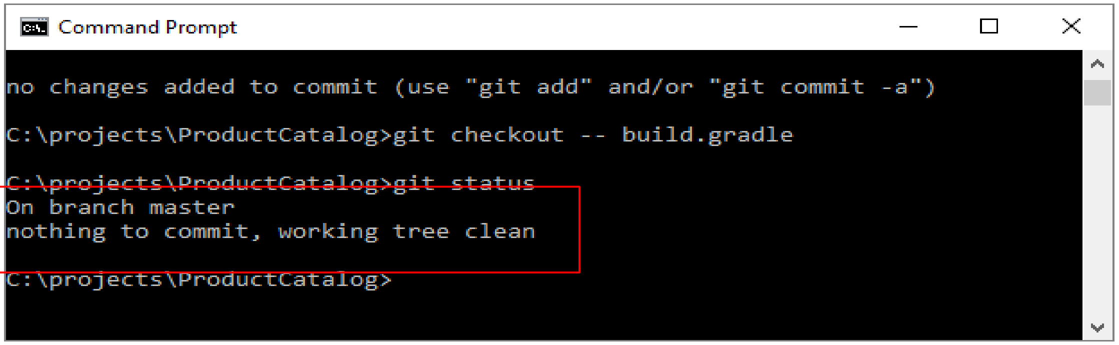
  "git reset HEAD LICENSE.MD"

- **Were the changes reverted?**
  - At this point, your file should go back to being listed as modified but not be in the staging area (run another git status to check)

Fully Reverting Changes (Backing out)

- The changes were **unstaged** but not reverted. To revert:
  - ***checkout – LICENSE.MD***

- Check the contents of the file now. What happened?
  - The file changes should have been reverted to their prior status
  - Your status should read " nothing to commit" if the changes were reverted successfully


Command Prompt

```
no changes added to commit (use "git add" and/or "git commit -a")

C:\projects\ProductCatalog>git checkout -- build.gradle

C:\projects\ProductCatalog>git status
On branch master
nothing to commit, working tree clean

C:\projects\ProductCatalog>
```

Push changes to GitHub

- Before pushing any changes remotely, you need to connect first your local copy with you **GitHub** remote repository.
- Create a repository online for you to send your new local repository.
  - Do not use the repo from the previous example.
- Type:
  - ***git remote add your_url***
  - In my case:

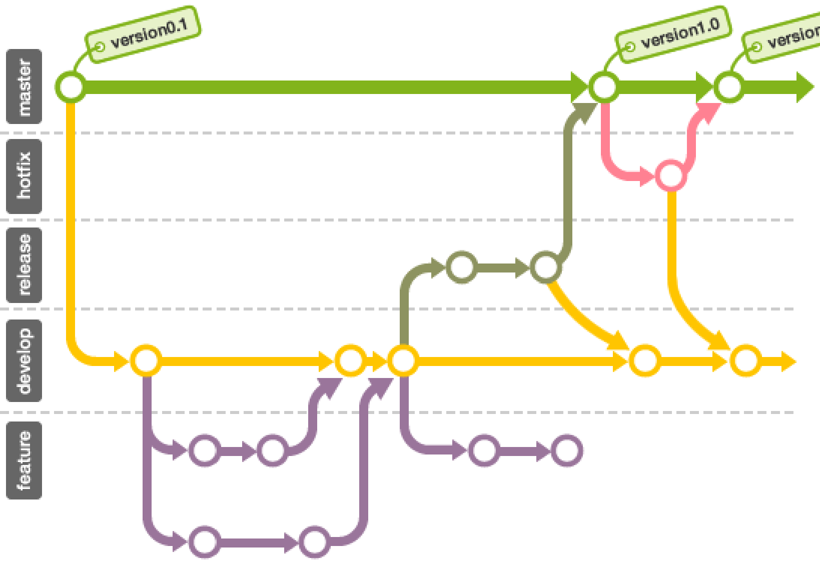
  "git remote add origin https://github.com/mgarzon/sample_02"
- You can now push your changes:

  git push -u origin master

ADVANCED CONCEPTS

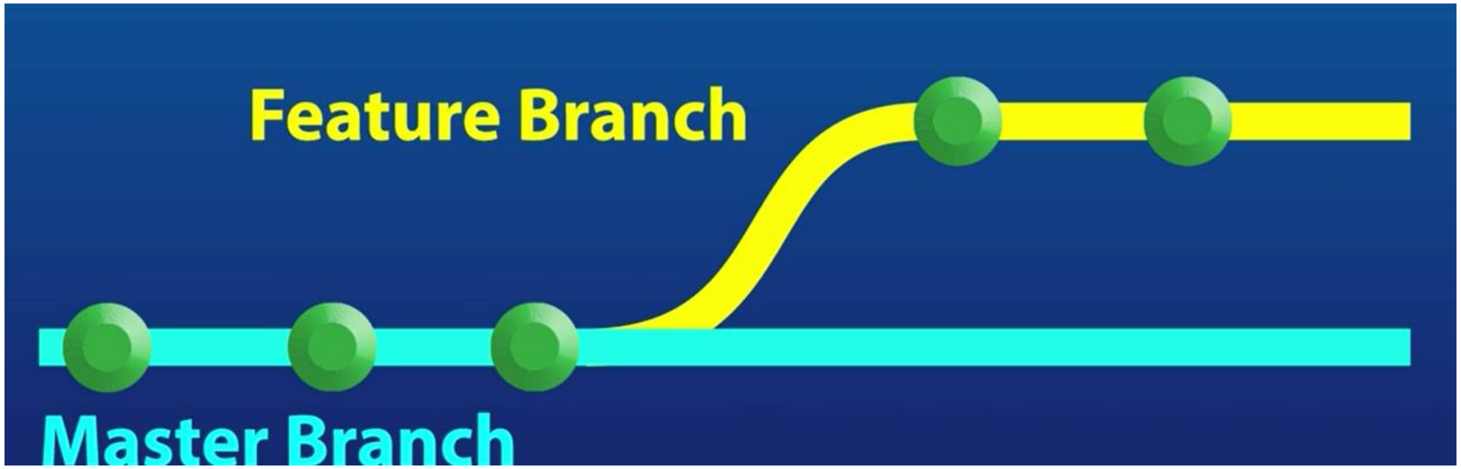
Branches

- A branch is a timeline of commits
- Branch names are labels, they do not carry special meaning other than the one given by the development team
  - Deletion removes label only
- You can use branches to allow for independent development by different team members or for specific development components.



Creating a Branch

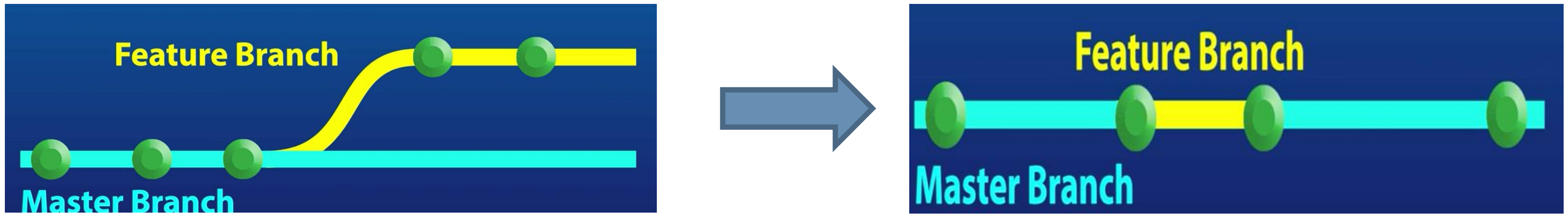
- We will rejoin the Master Branch later.
- Running a "**git branch <new branch name>**" command will create a new branch



Merging Scenarios

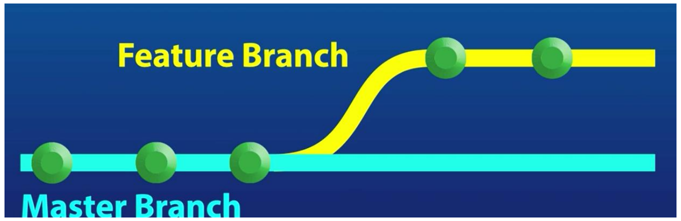
- Running a **"git merge"** command will merge two separate branches

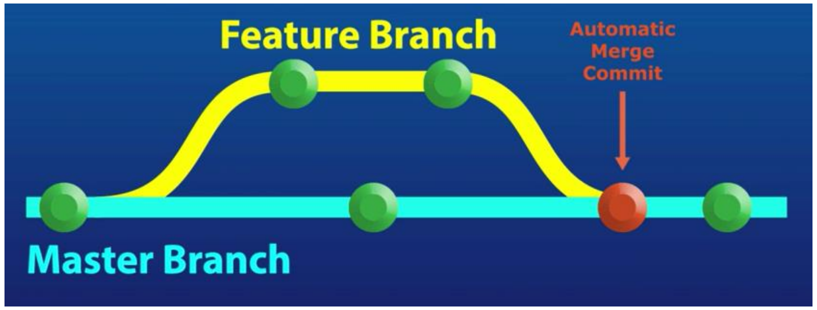
1. **Fast-Forward:**
   - Simplest case
   - No additional work has been detected on the Master (parent) Branch



Merging Scenarios (cont'd)

2. **Automated**
   - Non-Conflicting states
   - Preserves both Timelines

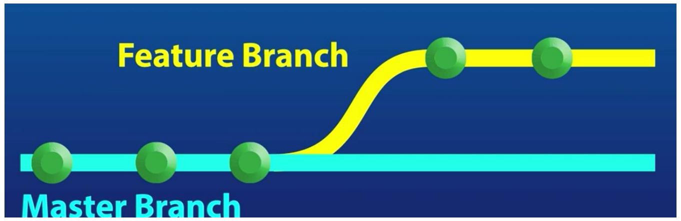


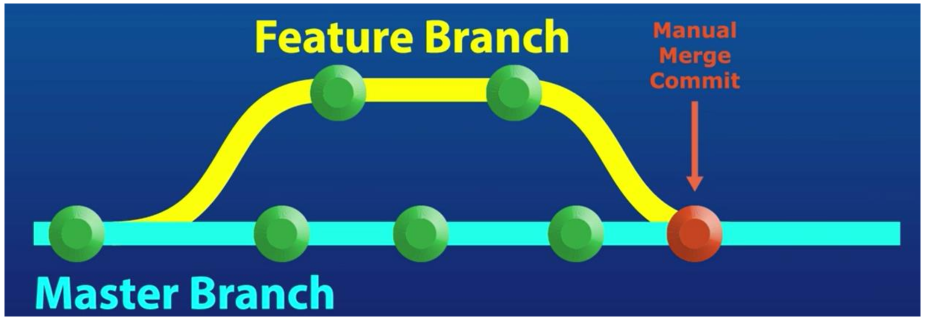


Merging Scenarios (cont'd)

3. Manual

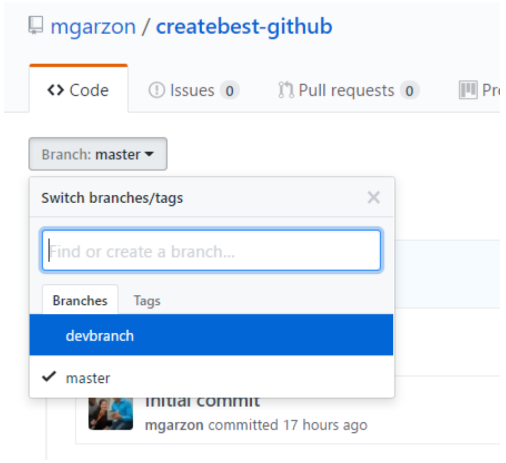
- Conflicting Merge State
- Git is unable to automatically resolve any conflicts
- Merge conflicts must be resolved before doing the commit
- Some tools can assist in solving file conflicts. 3-Way file diff is useful when two people make multiple changes to the same file.





Branching

- Type ***git branch*** to see the list of branches
- To **create** the branch and **switch** to that branch, type:
  - ***git checkout –b <your first name>***
  - ***E.g.: "git checkout –b Miguel"***
- Now, modify the README file and add, commit and push
  - ***git add .***
  - ***git commit –m "committing to another branch"***
  - ***git push origin Miguel***
- You can compare the branches by typing:
  - ***git diff master Miguel***



Merging

- To integrate the changes into the master branch:

1. **Switch** to my parent branch (master): ***git checkout master***
2. **Type *git log –all.*** What do you see?
3. **Merge : "*git merge Miguel*"**
4. **Type *git log –all.*** What do you see now?
   - At this point, the Miguel and master branches should have the same files and should be in the same revision.
5. Now delete the branch
   - ***"git branch –d Miguel"***

Example Conflict File after Merge

Consider the following Scenario where two people have to merge:

- Student 1 writes "Miguel" to their file, commits and pushes it to the server
- Student 2 writes "Felipe" to their file, commits and tries to push.
- Student 2 gets a warning about differences, pulls from remote and gets a conflict warning.
- At this point, the file should look something like:

```
<<<<<<< HEAD
miguel
=======
felipe
>>>>>>> 42708f7a9040d44b00998d8846be0bed0fc5ca3a
```

- After editing the file to fix the conflicts (or using a conflict resolution tool), Student 2 adds the file with conflict to the staging area, makes another commit and successfully pushes his changes to the remote.

CLASS EXERCISE

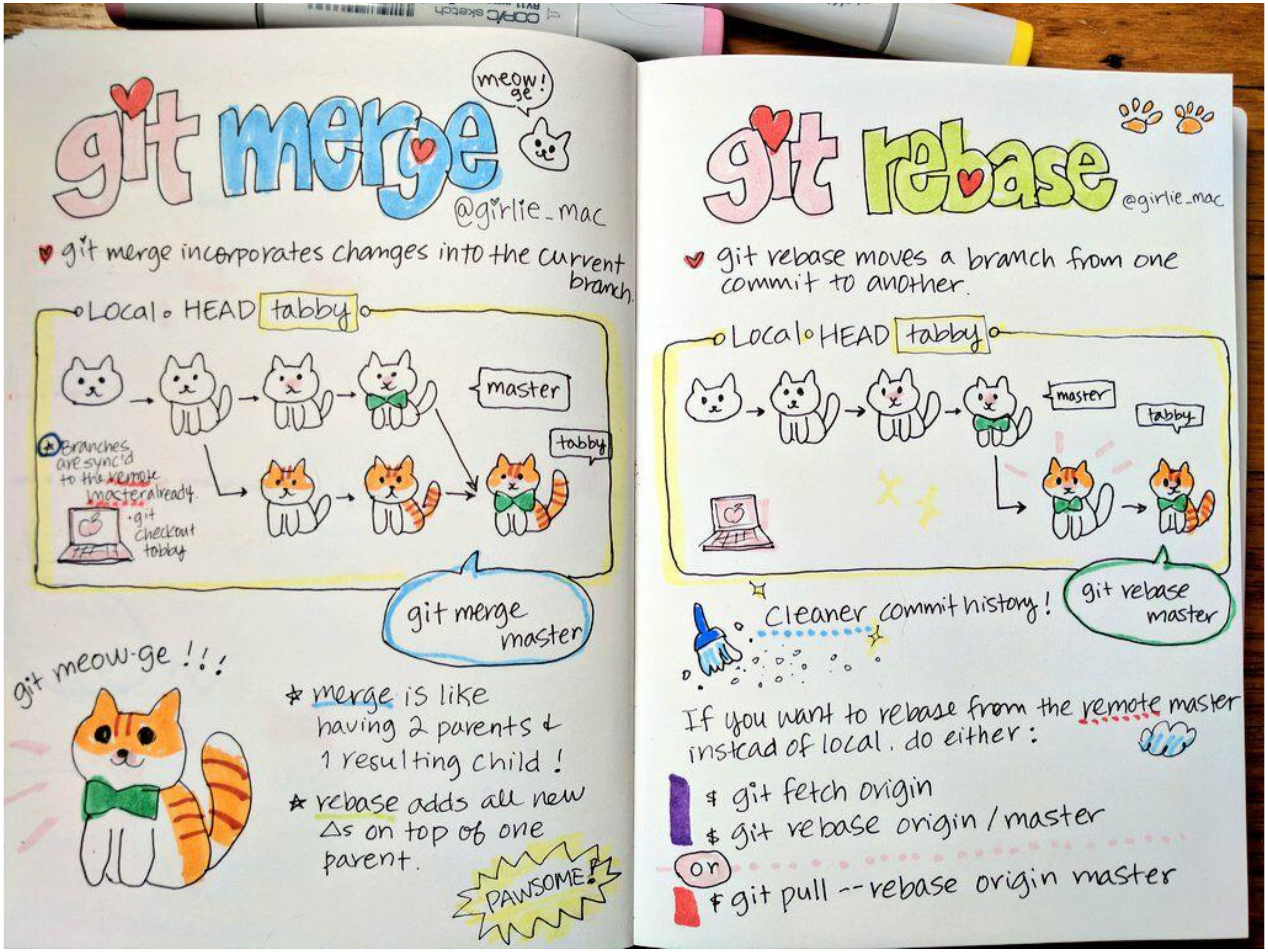
Exercise: Coordinating Access

- Students are to form groups of 4.
- **Part 1**
  - Each group will elect a member to create a repository that all members will use. This person has to add his colleagues as collaborators in Github in the projects settings panel.
  - The members of the group will then clone the repository to their local machines and create a text file with their first name. e.g.: "createbest.txt"
  - Each student must then commit and push their changes to the repository
    - Remember to **pull** changes from the repository before pushing your commit.
  - At the end of **Part 1**, the remote repository should have 4 files, each with the name of a team member. There shouldn't be any conflicts in the repository at this point.

Exercise: Resolving Conflicts

- **Part 2**
  - *Each member of the group will attempt to modify the same file.*
  - Each student must create a file named "team.txt" and write their first name in this file.
  - Each student must commit then commit and attempt to push their file.
    - 3 students wont be able to push without resolving conflict. After pulling remote changes, you will get a "conflict warning". Look into the file and you will see that the changes to the file proposed by the local and remote commits. Fix the file and add it to the staging area.
    - Push your changes (see next slide for example of file with conflict)

Students will be evaluated via their github repository commit history and the status of their files. Remember to write messages that make sense and explain what you did in your commits.



git merge incorporates changes into the current branch

★ merge is like having 2 parents & 1 resulting child!

★ rebase adds all new Δs on top of one parent.

git rebase moves a branch from one commit to another.

If you want to rebase from the remote master instead of local, do either:

```
$ git fetch origin
$ git rebase origin/master
```

or

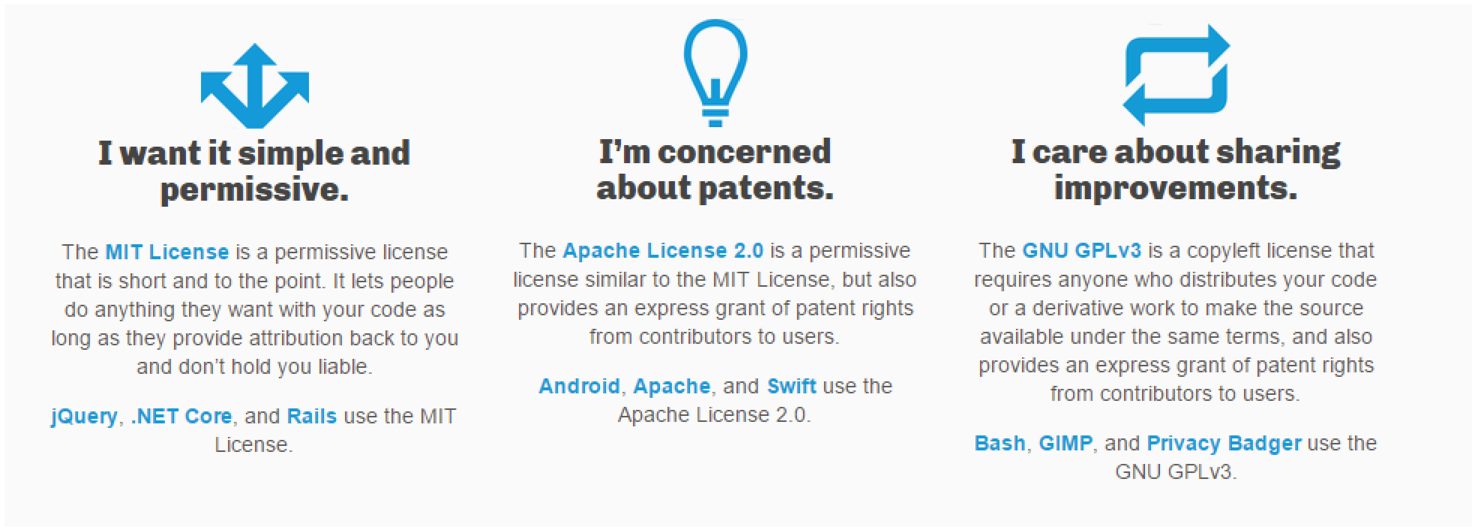
```
$ git pull --rebase origin master
```

Extra Note: Merging vs Rebasing – Graphical Example

Lesson 4: Open Source Licenses

Open Source Licenses

- Open source licenses are licenses that comply with the Open Source Definition — in brief, they allow software to be freely used, modified, and shared.



I want it simple and permissive.

The **MIT License** is a permissive license that is short and to the point. It lets people do anything they want with your code as long as they provide attribution back to you and don't hold you liable.

jQuery, **.NET Core**, and **Rails** use the MIT License.

I'm concerned about patents.

The **Apache License 2.0** is a permissive license similar to the MIT License, but also provides an express grant of patent rights from contributors to users.

Android, **Apache**, and **Swift** use the Apache License 2.0.

I care about sharing improvements.

The **GNU GPLv3** is a copyleft license that requires anyone who distributes your code or a derivative work to make the source available under the same terms, and also provides an express grant of patent rights from contributors to users.

Bash, **GIMP**, and **Privacy Badger** use the GNU GPLv3.

Open source licenses

- MIT: https://choosealicense.com/licenses/mit/
- Apache: https://choosealicense.com/licenses/apache-2.0/
- GNU: https://choosealicense.com/licenses/gpl-3.0/

How to apply the license?

Create a text file (typically named LICENSE or LICENSE.txt) in the root of your source code and copy the text of the license into the file.

More info: https://choosealicense.com/licenses/

THANK YOU!