

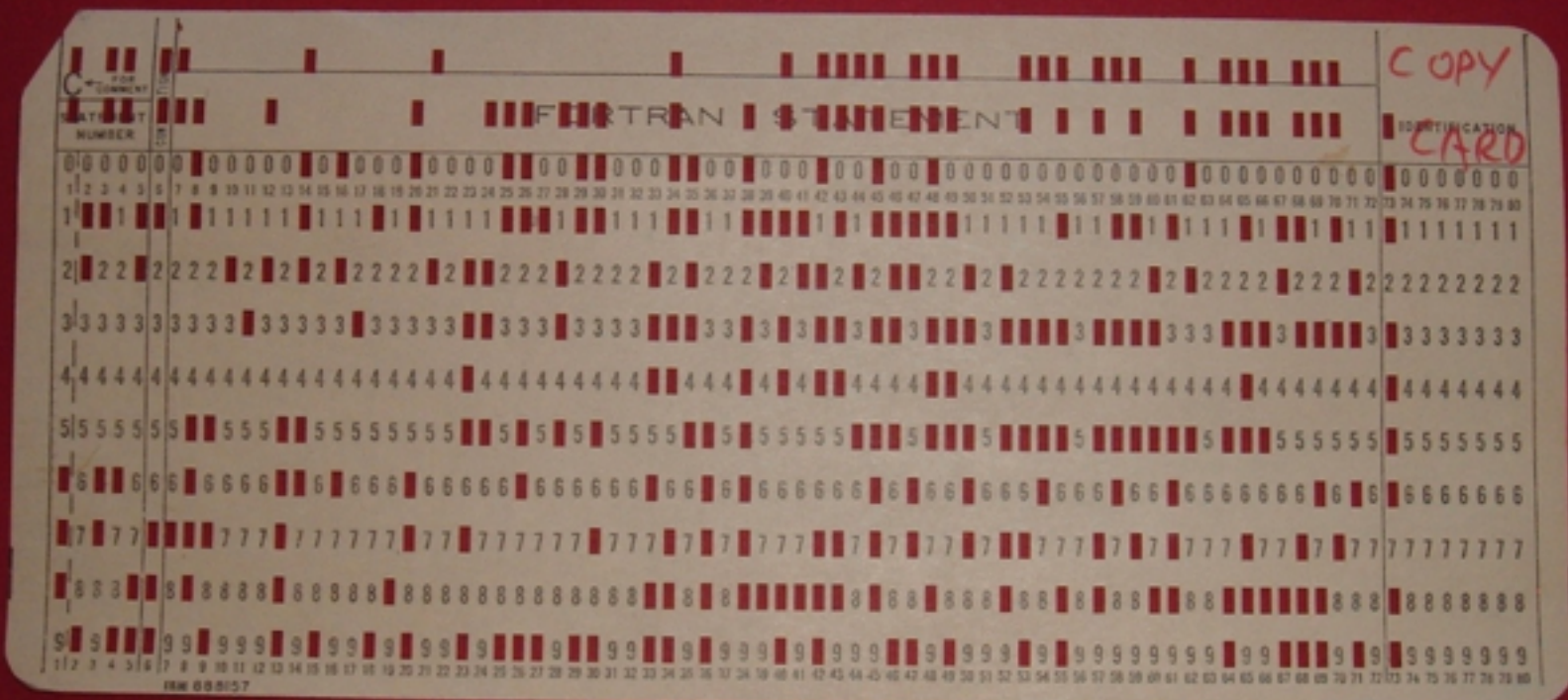
git

Git and GitHub: A Friendly Introduction

Bill Laboon

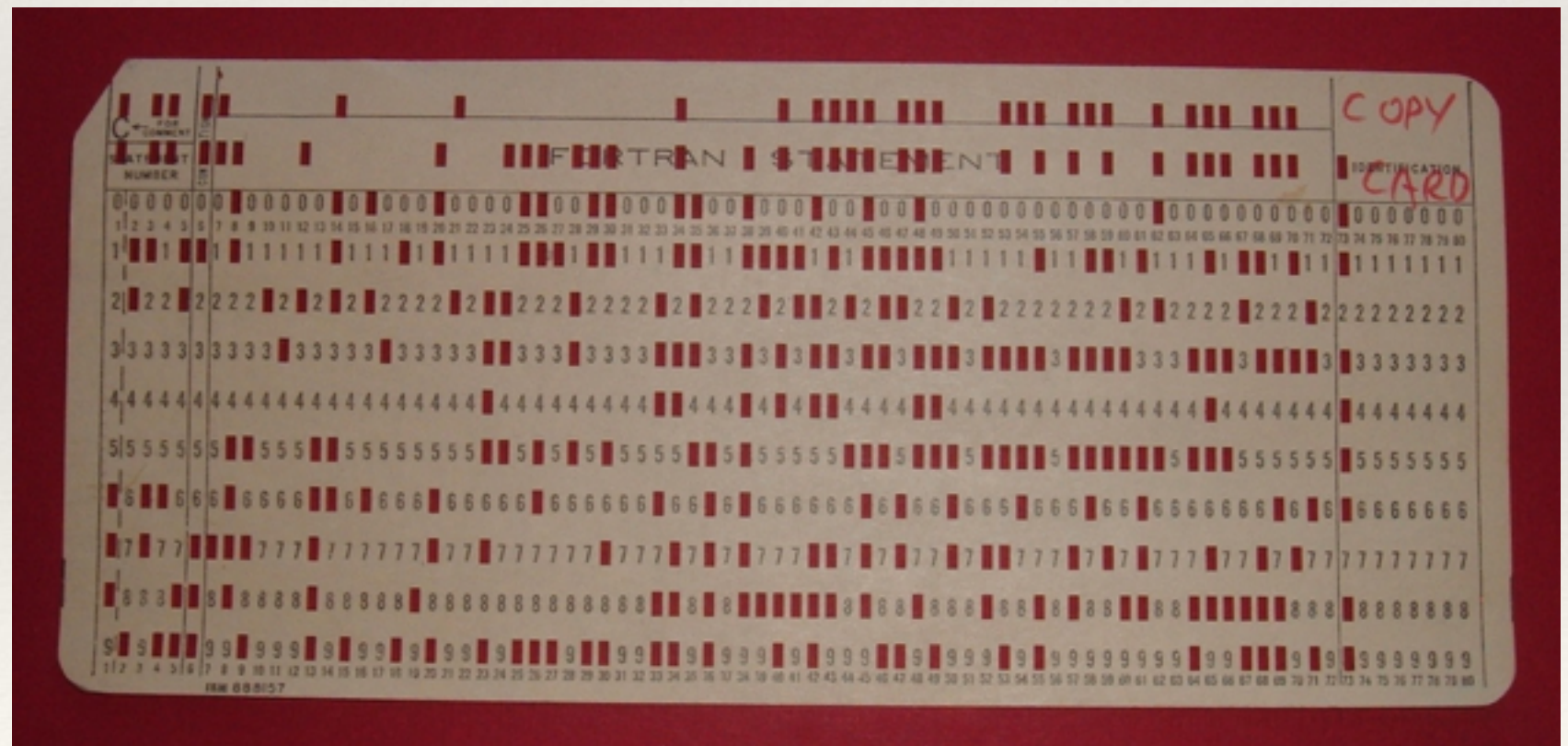
Version Control: A Brief History

- ❖ In the old days, you could make a copy of your code at a certain point, and release it
- ❖ You could then continue working on your code, adding features, fixing bugs, etc.
- ❖ But this had several problems!



VERSION 1

VERSION 2



Version Control: A Brief History

- ❖ Working with others was difficult - if you both modified the same file, it could be very difficult to fix!
- ❖ Reviewing changes from “Release n ” to “Release $n + 1$ ” could be very time-consuming, if not impossible
- ❖ Modifying code locally meant that a crash could take out much of your work

Version Control: A Brief History

- ❖ So now we have version control - a way to manage our source code in a regular way.
- ❖ We can tag releases without making a copy
- ❖ We can have numerous “save points” in case our modifications need to be unwound
- ❖ We can easily distribute our code across multiple machines
- ❖ We can easily merge work from different people to the same codebase

Version Control

- ❖ There are many kinds of version control out there:
 - ❖ BitKeeper, Perforce, Subversion, Visual SourceSafe, Mercurial, IBM ClearCase, AccuRev, AutoDesk Vault, Team Concert, Vesta, CVSNT, OpenCVS, Aegis, ArX, Darcs, Fossil, GNU Arch, BitKeeper, Code Co-Op, Plastic, StarTeam, MKS Integrity, Team Foundation Server, PVCS, DCVS, StarTeam, Veracity, Razor, Sun TeamWare, Code Co-Op, SVK, Fossil, Codeville, Bazaar....
- ❖ But we will discuss git and its most popular repository hosting service, GitHub

What is git?

- ❖ Developed by Linus Torvalds
- ❖ Strong support for distributed development
- ❖ Very fast
- ❖ Very efficient
- ❖ Very resistant against data corruption
- ❖ Makes branching and merging easy
- ❖ Can run over various protocols

Git and GitHub

- ❖ git != GitHub
- ❖ git is the software itself - GitHub is just a place to store it, and some web-based tools to help with development. Consider http vs a specific web site.
- ❖ There are other git hosting services:
 - ❖ BitBucket, GitLab, GNU Savannah, Sourceforce, etc.
 - ❖ You can even easily make your own!

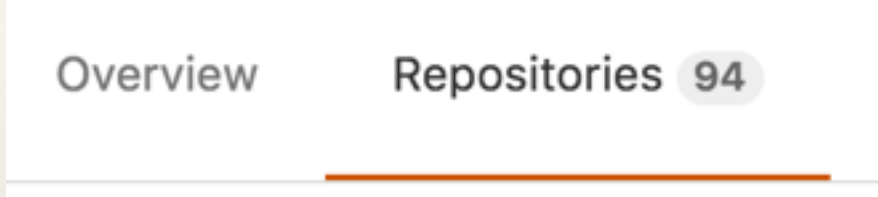
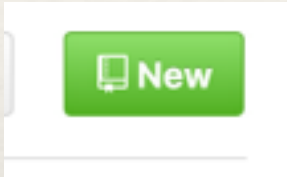
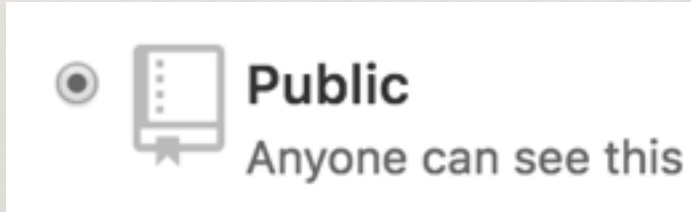
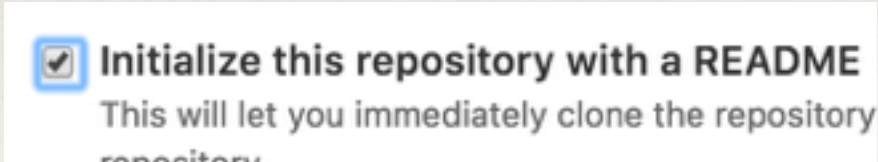
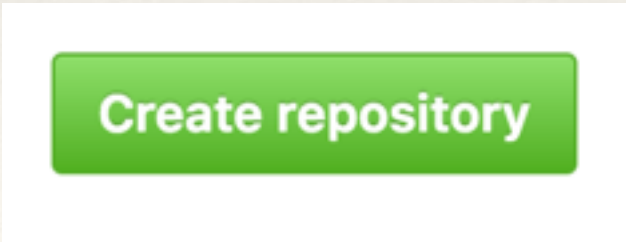
git Terminology

- ❖ A logical grouping of files into a project, controlled by git, is called a *repository* (often shortened to *repo*)
- ❖ This will map to a directory on your local machine
- ❖ But note that a file in the directory may not be controlled by git!

GitHub Basics

- ❖ You can easily create a new repository with GitHub's web interface
- ❖ You can then use your command line to “connect” a new directory / local repository to the repository
- ❖ Remember, this is not a different repository. It's a different computer accessing the same repository - git is distributed
- ❖ Theoretically you are both looking at the same repo, although like any distributed system, there are some practical concerns to worry about!


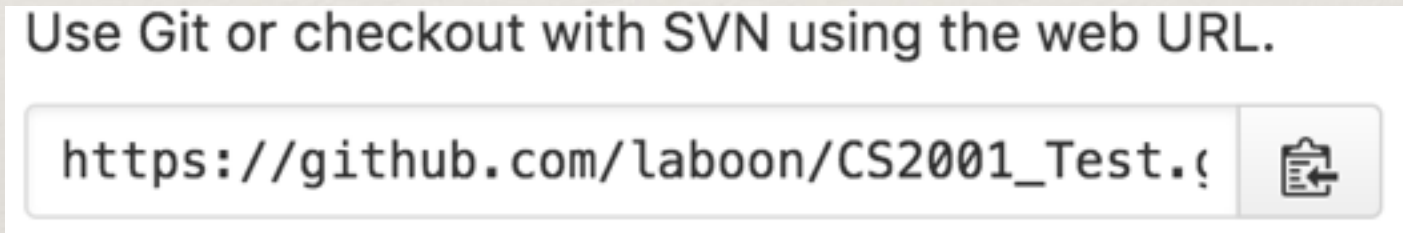
Let's Make A New Repo

- ❖ Go to <https://github.com> and log in
- ❖ Click on “Repositories”
- ❖ Click on “New”
- ❖ Give it the name “CS2001_Test” and the description “test repo for CS 2001”
- ❖ Ensure that the “public” radio button is selected
- ❖ Check the box to create a repository with a README.md file
- ❖ Click “Create Repository”

Let's Make A New Repo

- ❖ You should now see a web version of your repository
- ❖ There's not much there now, but we will add some more interesting things soon!

Let's Make A New Repo

- ❖ This repository doesn't help us much - we can edit it via the web, but we probably want to develop our code locally instead of via a textbox
- ❖ Click on the green "Clone or Download" button 
- ❖ Select the text (should be something like: https://github.com/laboon/CS2001_Test)

- ❖ Open up a terminal and go to the directory you want to put the Git repo in (note that the git repo will be a subdirectory of the current directory)
- ❖ Type "git clone https://github.com/laboon/CS2001_Test" (or whatever the text you selected is)

We Now Have a Local Repo Sync'd To The Repo On GitHub

- ❖ Note that this is not just a copy - it also includes the data necessary to synchronize with the *origin* (the original location of the repo)
- ❖ The origin does not have to be on GitHub - it can be anywhere. But most projects use GitHub or another hosting site.
- ❖ The origin is considered the “first among equals” in the distributed system that is git
- ❖ But the fact that it's distributed means work can be done even without access to the origin (e.g. on a plane)

What Do We Have?

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17467) $ ls -al | grep -v "\.$"
total 8
drwxr-xr-x  12 laboon  staff   408 Dec  5 10:37 .git
-rw-r--r--   1 laboon  staff    35 Dec  5 10:37 README.md
```

.git directory: Stores all of the git information. You should not need to go here! Note that it is a dot-file (hidden).

README.md: Our single actual file under source control.

git status

- ❖ Type “git status” to see an overview of the status.
- ❖ Right now, we are up-to-date with origin, have not added any files, and have not modified anything.

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17468) $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Status of files in a git repository

- ❖ Files can be in one of several states:
 - ❖ **Untracked:** git is not even looking at this file (e.g. .class files or other generated executables)
 - ❖ **Tracked but Unmodified:** The file is being tracked by git, and no changes have been made to it since then.
 - ❖ **Modified:** This file has been modified, but is not staged for commit (that is, you haven't decided if you want to keep these changes)
 - ❖ **Staged:** You have modified the file, and told git to be prepared to keep these changes
 - ❖ **Committed:** You have decided to keep these changes. This is functionally equivalent to “tracked but unmodified”.

A Commit Is Like A “Save Point”

- ❖ A repository consists of many commits
- ❖ This allows you to slowly build up your software, one commit at a time
- ❖ Also allows you to “travel through time” - see how the code looked before, see and revert changes, etc.

Commits and SHAs

- ❖ Each commit is uniquely tagged with an identifier, which is produced via SHA256 hashing
- ❖ It's commonly called a SHA
- ❖ Right now, we only have one commit

```
commit 24e87628dc570c25af6fd580d248ef958d55264a
Author: Bill Laboon <laboon@users.noreply.github.com>
Date:   Mon Dec 5 10:25:31 2016 -0500
```

Initial commit

Let's Make A New File

- ❖ Using your favorite text editor (or your least-favorite one), create a file `Hello.java` in the same directory as `README.md`
- ❖ Add a simple “Hello, World!” java program
- ❖ Compile it
- ❖ Now type “git status”

Current Status of Repo

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17470) $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    Hello.class
    Hello.java

nothing added to commit but untracked files present (use "git add" to track)
```

- ❖ We now have two files added, not tracked by git
- ❖ We don't want to track Hello.class (it's a generated file, but do want to add Hello.java)

Add A File To A Repo

- ❖ Type “git add Hello.java” to add the file to git’s tracking control
- ❖ Type “git status” to view our current status

```
(17472) $ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
new file:   Hello.java
```

```
Untracked files:
```

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

```
Hello.class
```

Staging and Committing

- ❖ Hello.java is now STAGED, but not committed
- ❖ Now if we commit, Hello.java will be added, but Hello.class will NOT
- ❖ Commit by typing “git commit -a”. This means “commit all staged changes”

Committing

- ❖ A text editor will pop up (usually vim - you can modify this) and you will be asked to describe the commit
- ❖ Type in “Adding Hello.java” and save the file (if you are not used to vim - type i, then the text, then <ESC>:wq to write and quit)

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17473) $ git commit -a
[master b696069] Adding Hello.java
1 file changed, 6 insertions(+)
create mode 100644 Hello.java
```

Committing

- ❖ We can now check our status (via “git status”)
- ❖ We are now “ahead” of the branch on GitHub by one commit (the repo here does not automatically update the one on GitHub)
- ❖ Note also that it is yelling at us that the class file is not included! This can get annoying with large numbers of files we don’t want to commit.

.gitignore

- ❖ In the root directory, make a .gitignore file and add the following text:
*.class
- ❖ Now git will “ignore” any files with that name
- ❖ You can also specify subdirectories, or more complicated regular expressions
- ❖ You can set a default .gitignore file (e.g. for Java, or TeX, or whatever) when generating a repo via GitHub

git status and .gitignore

- ❖ Let's add the .gitignore file and commit it to the repo, so this won't happen for anyone else who uses the repo

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17484) $ git add .gitignore
```

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17485) $ git commit -am "add gitignore"
[master f29894a] add gitignore
1 file changed, 1 insertion(+)
create mode 100644 .gitignore
```

git status and .gitignore

- ❖ Now git doesn't talk to you about .class files!
- ❖ Also useful for things like local config files, IDE files, backup files like #FILE.TXT# or File.txt~

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17486) $ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

Pushing and Pulling

- ❖ Now we want to show off our beautiful repo to the world, but the GitHub page still shows nothing but the original commit (just the README.md file)!
- ❖ We need to “push” our changes
- ❖ If we want to get changes that have occurred, we would “pull”
- ❖ Think of each of these as “one-way synchronization”

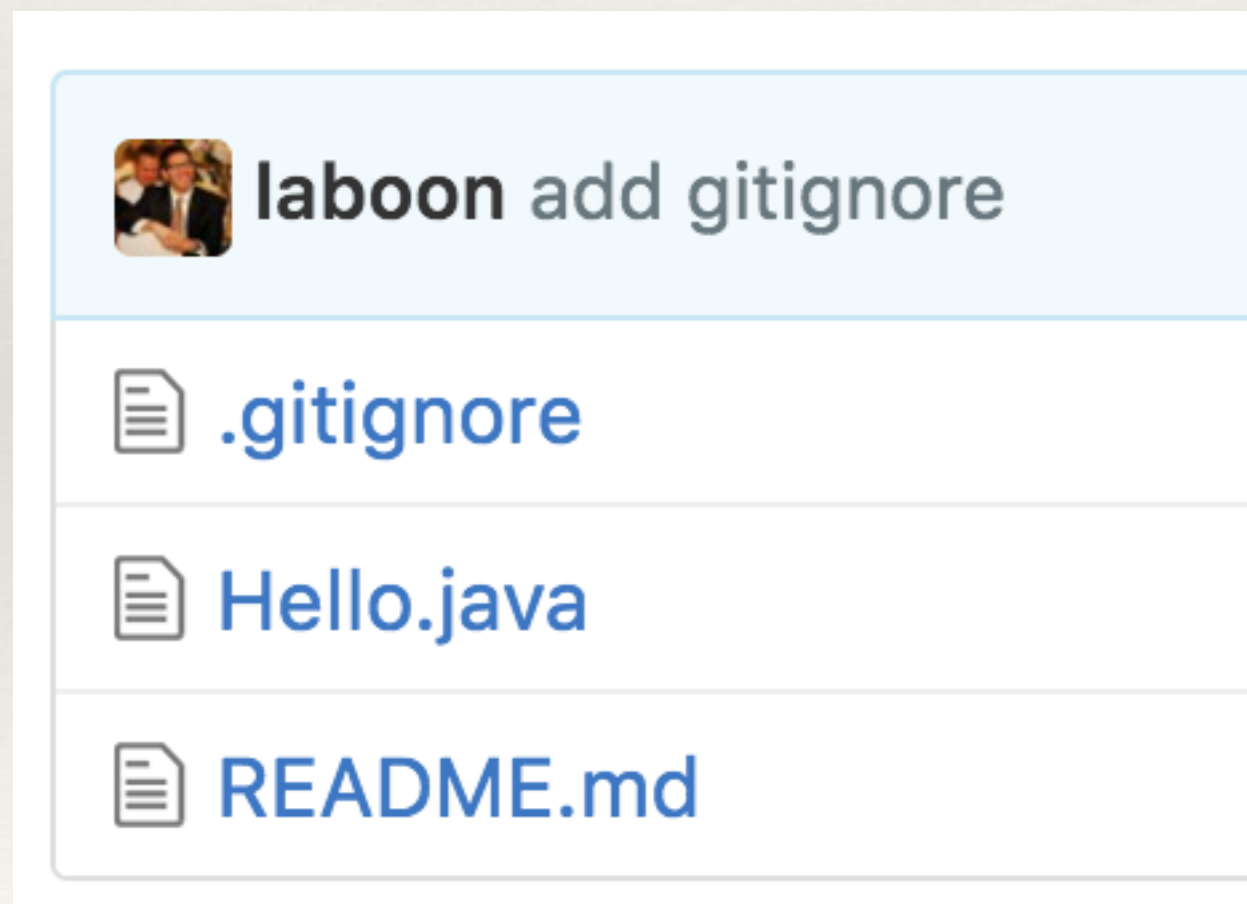
Push

- ❖ Type “git push origin master”
- ❖ This will “push” our changes back to origin (in our case, this is GitHub, but remember can be other things)

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17487) $ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 653 bytes | 0 bytes/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/laboon/CS2001_Test.git
    24e8762..f29894a  master -> master
```

Push

- ❖ Let's see if that code is up there... yes! Beautiful!

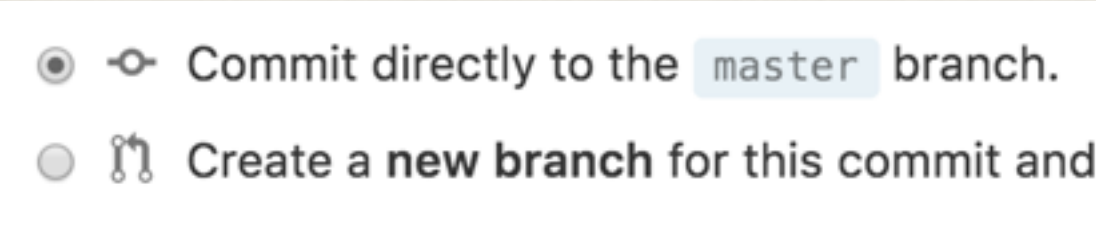



Pulling

- ❖ What if somebody else made a change and we want it to show up on our local machine?
- ❖ Let's add a file via the web interface...

Modifying Code Via The Web

- ❖ Click on “Create New File”
- ❖ Give it the name “Foo.txt”
- ❖ In the main textbox, type “This is a new file”
- ❖ At the bottom of the page, give it a commit description “Adding Foo.txt”. You do not need to add an extended description.
- ❖ Ensure the radio button for “commit directly to master” is selected
- ❖ Click on “Commit New File”

A light gray rectangular button with rounded corners and a thin border, containing the text "Create new file" in a bold, black, sans-serif font.A breadcrumb showing the current file path. It consists of the text "CS2001_Test" in blue, followed by a slash "/" and a text input field containing "Foo.txt". The input field has a blue border and a small circular icon to its right.Two radio button options for committing. The first option is selected, indicated by a filled circle. It features a branch icon (a circle with a horizontal line) and the text "Commit directly to the master branch." where "master" is highlighted in a light blue box. The second option is unselected, indicated by an empty circle, and features a branch icon and the text "Create a new branch for this commit and".

☒  Commit directly to the master branch.

☐  Create a new branch for this commit and

A bright green rectangular button with rounded corners and a thin border, containing the text "Commit new file" in a bold, white, sans-serif font.

Now We Have Another Commit, Visible From The Web



laboon committed on **GitHub** Adding Foo.txt



[.gitignore](#)



[Foo.txt](#)



[Hello.java](#)



[README.md](#)

The Drawbacks of “git status”

- ❖ Let's look locally. We have not “pulled”, so we should not see the new file Foo.txt

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17494) $ ls -l
total 24
-rw-r--r--  1 laboon  staff  417 Dec  5 11:12 Hello.class
-rw-r--r--  1 laboon  staff  113 Dec  5 11:12 Hello.java
-rw-r--r--  1 laboon  staff   35 Dec  5 10:37 README.md
```

The Drawbacks of “git status”

- ❖ So git status should tell us we are one commit behind origin, right?

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17488) $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```



git remote update

- ❖ Remember that git is distributed and will not check origin unless you ask for it! We need to tell our local system to go check with origin and see if there has been an update
- ❖ Use “git remote update” to do so

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17492) $ git remote update
Fetching origin
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/laboon/CS2001_Test
   f29894a..381c2a0  master    -> origin/master
```

git remote update

- ❖ Now try git status and you can see that locally we are behind by one commit

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17493) $ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
nothing to commit, working directory clean
```

git pull

- ❖ Use git pull to do the “opposite” of git push, and synchronize to what’s on origin

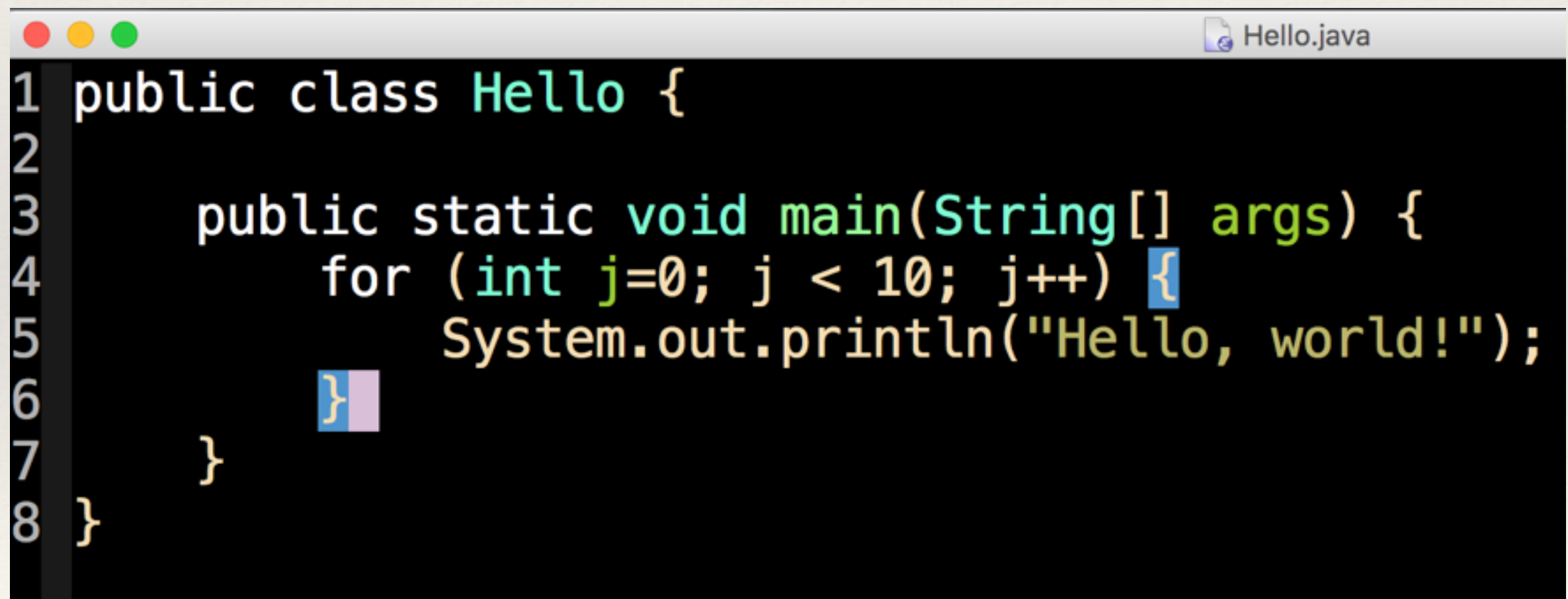
```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17495) $ git pull
Updating f29894a..381c2a0
Fast-forward
 Foo.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 Foo.txt
```

Modifying Files

- ❖ Now let's say that we don't want to have our "Hello, world!" program say "Hello, world!"
- ❖ We want to make it say it 10 times!

Modifying Files

- ❖ Add a for loop to do this in your Hello.java file



```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         for (int j=0; j < 10; j++) {  
5             System.out.println("Hello, world!");  
6         }  
7     }  
8 }
```

git status

- ❖ git status tells us once again that we have some changes which will not be committed.
- ❖ Also, a backup file which is not being tracked! Sidenote: how could we get rid of this annoying message?

```
(17496) $ git status
On branch master
Your branch is up-to-date with 'origin/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:   Hello.java

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

        Hello.java~
```

Stage the changes

- ❖ git add the file in order to stage its changes for commit (you can also use git add -p to stage only SOME changes for commit)

```
[laboon@ekaterina ~/pitt/CS2001_Test]  
(17497) $ git add Hello.java
```

```
[laboon@ekaterina ~/pitt/CS2001_Test]  
(17498) $ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
modified:   Hello.java
```

Now commit

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17499) $ git commit -am "10x java"
[master f52b890] 10x java
1 file changed, 3 insertions(+), 1 deletion(-)
```

git log

- ❖ Let's look at the work that we've done with git log
- ❖ This will show us all of the commits that we've done, along with their SHAs (IDs) and commit messages
- ❖ Note that your SHAs will be different than mine!

git log

```
[laboon@ekaterina ~/pitt/CS2001_Test]
```

```
(17500) $ git log
```

```
commit f52b890a04a1ce4f5aa332c4ff90a9b67e31c01c
```

```
Author: laboon <laboon@gmail.com>
```

```
Date: Mon Dec 5 12:08:18 2016 -0500
```

```
10x java
```

```
commit 381c2a0b733e495b0d27bd9b7570a1dbaba46cf1
```

```
Author: Bill Laboon <laboon@users.noreply.github.com>
```

```
Date: Mon Dec 5 11:49:16 2016 -0500
```

```
Adding Foo.txt
```

```
commit f29894a8307e3ceafa0cc1e194a0f674d3af2581
```

```
Author: laboon <laboon@gmail.com>
```

```
Date: Mon Dec 5 11:23:02 2016 -0500
```

```
add gitignore
```

git diff

- ❖ Let's see what has changed since two commits ago
- ❖ git diff will show you - you just need to give it the SHA of the commit you want to compare the current one to
- ❖ + means line added
- ❖ - means line deleted
- ❖ Note that lines are never changed - something you think is “modified” is actually a line that was deleted and a similar one added (at least deep down in git's core - some tools will try to “guess” which lines have been modified)

git diff

```
(17505) $ git diff f29894a8307e3ceafa0cc1e194a0f674d3af2581
diff --git a/Foo.txt b/Foo.txt
new file mode 100644
index 0000000..6dfa057
--- /dev/null
+++ b/Foo.txt
@@ -0,0 +1 @@
+This is a new file
diff --git a/Hello.java b/Hello.java
index b45e38b..6922bbb 100644
--- a/Hello.java
+++ b/Hello.java
@@ -1,6 +1,8 @@
 public class Hello {

     public static void main(String[] args) {
-        System.out.println("Hello, world!");
+        for (int j=0; j < 10; j++) {
+            System.out.println("Hello, world!");
+        }
     }
 }
```

Branching

- ❖ One of the powerful features of git is easy and cheap branching
- ❖ A “branch” is like a parallel universe of code, which you can modify without messing with your main branch (usually called “master” by default)
 - ❖ You can try something out on a new branch
 - ❖ You can work on your own new feature or bugfix
- ❖ So far, we have done all work in the same “branch”

Branches

- ❖ Usually, work is never done on the master branch
- ❖ Rather, it's done on a branch until it's ready, then we “merge” the changes back in
- ❖ This also allows us to clean up our commits and whatnot before adding it to master

Creating a new branch

- ❖ Use `git checkout -b NEW_BRANCH_NAME`
- ❖ This will create a new “parallel universe” (branch)
- ❖ I like to put my initials in as the prefix (in case other people are working on this repo, they know who is responsible) and a short description

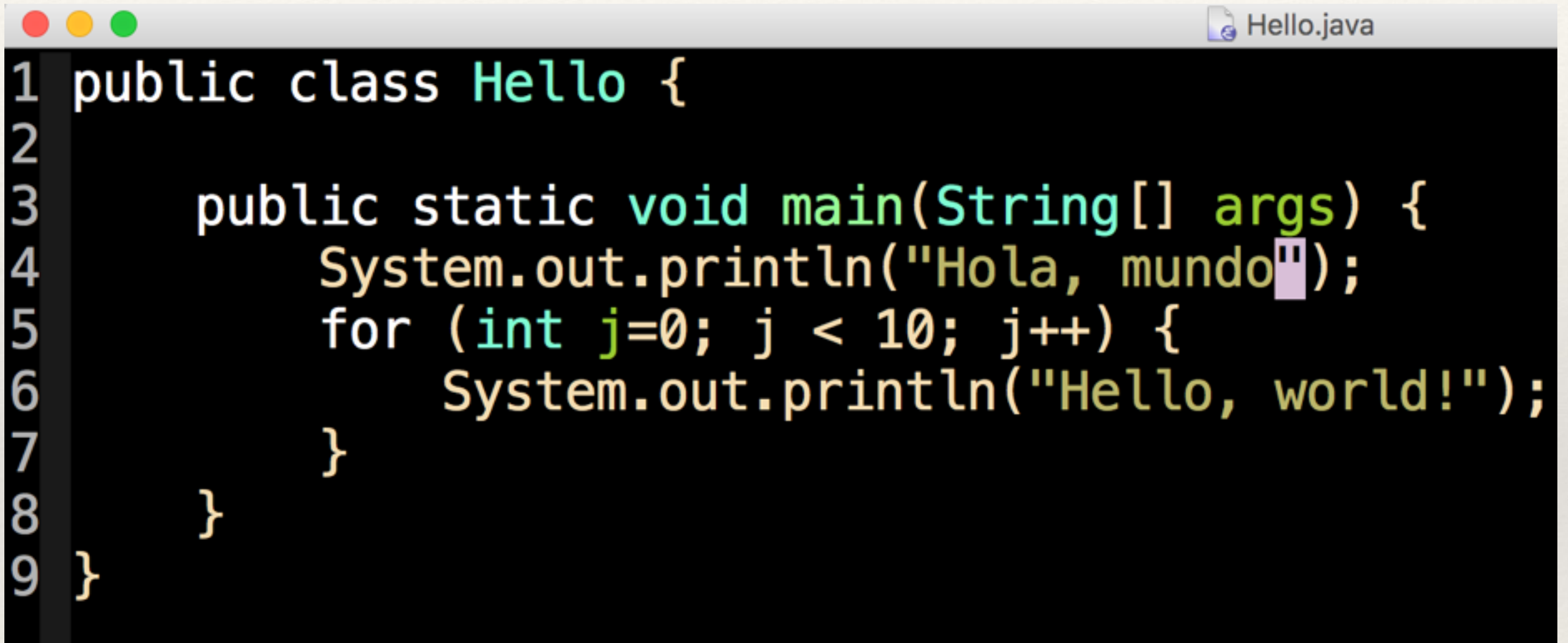
Seeing Branches

- ❖ `git branch` will show you all local branches
- ❖ `git branch -a` will show you all branches on local and origin

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17508) $ git branch
master
* wjl_spanish
```

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17509) $ git branch -a
master
* wjl_spanish
remotes/origin/HEAD -> origin/master
remotes/origin/master
```


Let's Make Some Changes!



```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.println("Hola, mundo");  
5         for (int j=0; j < 10; j++) {  
6             System.out.println("Hello, world!");  
7         }  
8     }  
9 }
```

And commit them...

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17511) $ git commit -am "add Spanish version"
[wjl_spanish b8acbf6] add Spanish version
1 file changed, 1 insertion(+)
```

Parallel Universe Travel

- ❖ So now we have the master branch and the wjl_spanish branch
- ❖ We can “hop” from one to the other, and see that master does not have any of the changes we made
- ❖ We use `git checkout BRANCH_NAME` to do so

Parallel Universe Travel

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17513) $ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
    (use "git push" to publish your local commits)
```

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17514) $ more Hello.java
public class Hello {

    public static void main(String[] args) {
        for (int j=0; j < 10; j++) {
            System.out.println("Hello, world!");
        }
    }
}
```

Go Back To wjl_spanish Universe

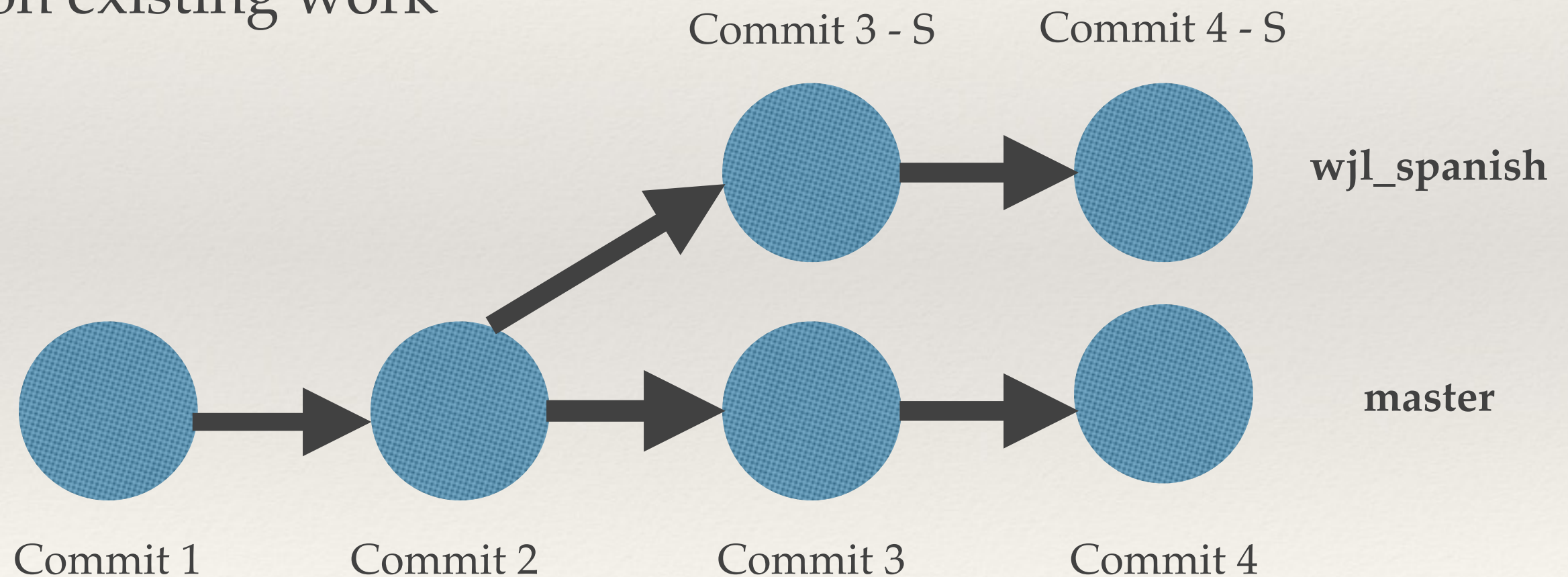
```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17515) $ git checkout wjl_spanish
Switched to branch 'wjl_spanish'
```

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17516) $ more Hello.java
public class Hello {

    public static void main(String[] args) {
        System.out.println("Hola, mundo");
        for (int j=0; j < 10; j++) {
            System.out.println("Hello, world!");
        }
    }
}
```


This Branching Is A Key Idea in git

- ❖ And really any version control!
- ❖ It allows us to work in an isolated environment, but based on existing work



Merging

- ❖ Eventually, though, we want to come back to reality from our parallel universe
- ❖ We can bring the code in via *merging*
- ❖ This is done by going to the branch we want to merge into, and typing `git merge BRANCH_TO_MERGE`

Merging

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17517) $ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
    (use "git push" to publish your local commits)
```

```
[laboon@ekaterina ~/pitt/CS2001_Test]
(17518) $ git merge wjl_spanish
Updating 00faead..b8acbf6
Fast-forward
 Hello.java | 1 +
1 file changed, 1 insertion(+)
```

Now The Parallel Universe Is Part of Ours

```
[laboon@ekaterina ~/pitt/CS2001_Test]
```

```
(17519) $ git branch
```

```
* master
```

```
wjl_spanish
```

```
[laboon@ekaterina ~/pitt/CS2001_Test]
```

```
(17520) $ more Hello.java
```

```
public class Hello {
```

```
    public static void main(String[] args) {
```

```
        System.out.println("Hola, mundo");
```

```
        for (int j=0; j < 10; j++) {
```

```
            System.out.println("Hello, world!");
```

```
        }
```

```
    }
```

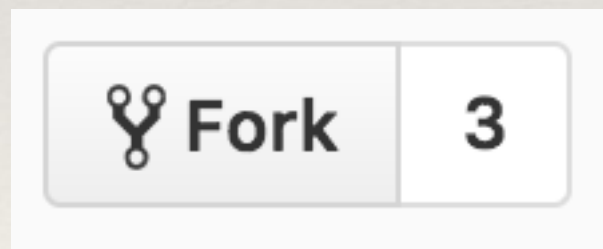
```
}
```

Forking

- ❖ You can make a copy of a repository that already exists by *forking* it
- ❖ Note that this is not cloning - we do not intend to be directly synced with this repository
- ❖ It will have its own origin - it will just copy the entire repo over
- ❖ This is useful when working on projects that are owned by others!
- ❖ We can add the modifications we did by issuing a pull request

Forking Example

- ❖ Let's say I want to make some changes to Carol Nichols' pr-practice repo (note that this repo is set aside just for practice forks and pull requests!)
- ❖ I go to <https://github.com/carols10cents/pr-practice> and click on "Fork"



Forking Example

- ❖ Now I have my own copy of pr-practice at <https://github.com/laboon/pr-practice>
- ❖ But I know that it is a copy of the original one:

 **laboon / pr-practice**
forked from **carols10cents/pr-practice**

Pull Requests

- ❖ I can now clone to my computer and make changes just like any other repo I own
- ❖ I make a new branch, make changes, and commit
- ❖ And I can ask Carol to add my changes in via a *pull request* (I request that she pulls my changes in)

Pull Requests

- ❖ I make a new branch, wjl_clowder, and add a line to the cats.txt file
- ❖ I then commit it. Let's see what I've changed.

```
(17539) $ git diff 71c4df72948c16d6eeffb0ba95324e1057e8fc68a
diff --git a/cats.txt b/cats.txt
index 2bf0d35..f90b7e0 100644
--- a/cats.txt
+++ b/cats.txt
@@ -51,3 +51,5 @@ P-I-T-T let's go PITT!
    On average, cats spend 2/3 of every day sleeping. That means
    for only three years of its life.

    The technical term for a cat's hairball is a "bezoar".
+
+A group of cats is called a clowder.
```

Pushing Up My Branch

- ❖ I push my wjl_clowder branch to laboon/pr-practice
- ❖ Note - this is NOT the master branch!

```
[laboon@ekaterina ~/pitt/pr-practice]
(17541) $ git push origin wjl_clowder
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 316 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/laboon/pr-practice.git
* [new branch]      wjl_clowder -> wjl_clowder
```

Now I Will Ask Carol to Accept My Branch

- ❖ I go to <https://github.com/carols10cents/pr-practice/> (*not* my own fork - Carol's original)
- ❖ I click on the “Pull requests” tab
- ❖ I click on New pull request



 Pull requests 0



New pull request

Compare Across Forks

- ❖ I am requesting my change come from another fork, so I click on the link to “compare across forks.”

[compare across forks.](#)

- ❖ I can now see all of the forks that have been made from this repo (GitHub keeps track for you).

Compare changes

Compare changes across branches, commits, tags, and more below. If you need to, you can also [compare across forks](#).



base fork: carols10cents/pr-practice ▼

base: master ▼

...

head fork: carols10cents/pr-practice ▼

compare: master ▼

I Select Source and Destination Branches

- ❖ base is the destination (where you want the changes to go)
- ❖ head is the source (where the changes are coming from)



base fork: carols10cents/pr-practice ▼

base: master ▼

...

head fork: laboon/pr-practice ▼

compare: wjl_clowder ▼

✓ **Able to merge.** These branches can be automatically merged.

Review The Changes

Showing 1 changed file with 2 additions and 0 deletions.

Unified Split

2 cats.txt

View



@@ -51,3 +51,5 @@ P-I-T-T let's go PITT!

51 On average, cats spend 2/3 of every day sleeping. That means
a nine-year-old cat has been awake for only three years of
its life.


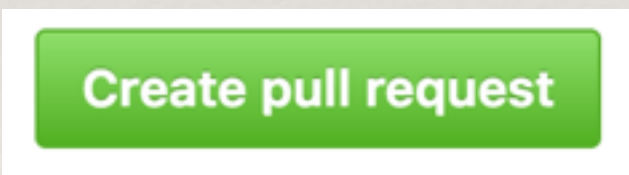
52
53 The technical term for a cat's hairball is a "bezoar".

51 On average, cats spend 2/3 of every day sleeping. That means
a nine-year-old cat has been awake for only three years of
its life.

52
53 The technical term for a cat's hairball is a "bezoar".

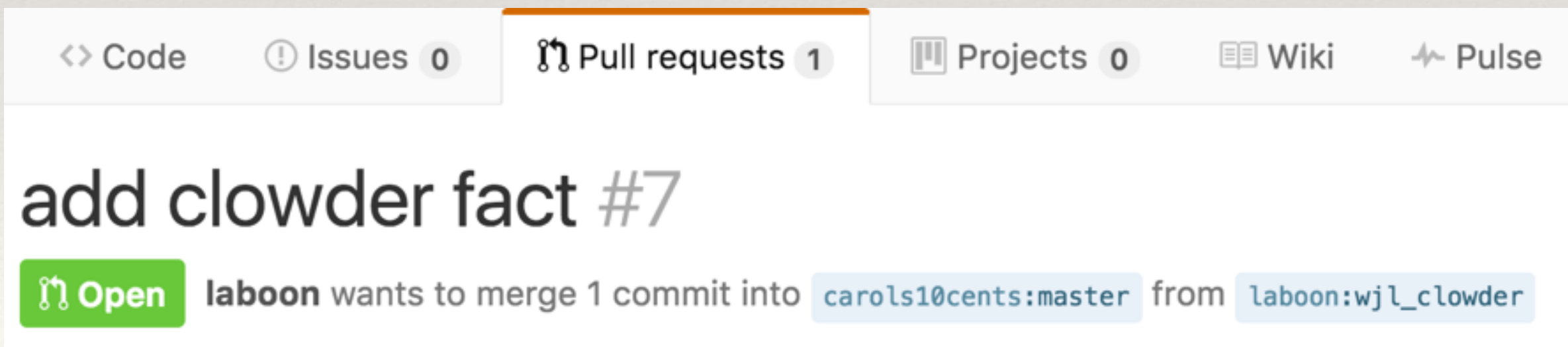
54 +
55 +A group of cats is called a clowder.

Now Create The Pull Request

- ❖ Click on the “Create Pull Request” button 
- ❖ You will have to add a comment for this. Do so, or accept the default, and click “Create Pull Request” again. 

PR added!

- ❖ Carol is notified via GitHub
- ❖ She can now review the code and decide whether or not to bring in my pull request to her codebase
- ❖ She may leave a comment saying that something has to be changed, or if she has questions, etc.
- ❖ From her perspective, all she has to do is click “Merge” and it will be merged to the master branch



Further Reading

- ❖ This was just a taste of what git can do!
- ❖ In-depth git tutorial: <https://git-scm.com/docs/gittutorial>
- ❖ Pro git: A freely-available book on using git: <https://git-scm.com/book/en/v2>
- ❖ The git Reference: when you need a really in-depth look at something: <http://gitref.org/>
- ❖ Detailed pull request guide: <https://help.github.com/articles/about-pull-requests/>
- ❖ Git 911: See the git911.md file in this directory. There are several git exercises here. Answers are in git911-answers.md