Web Development Immersive

Day 1, Week 1
New York, NY

Overview - Today

- Introductions
- Culture & Expectations
- Software Development Process
- *nix and Sublime Basics
- Git & Github

Preview - Tomorrow

- Ruby Basics
- Functions
- First lab exercise

Preview - Week

Monday (today)	Tuesday	Wednesday	Thursday	Friday
Workflow, *nix, Sublime, Git	Ruby basics and functions	Debugging, arrays, blocks	OOP, Modules	File I/O, TDD

**This is all subject to minor change as we go faster/slower*

Introductions - Instructors

- David Fisher
- Chris Clearfield
- Phil Lamplugh

Introductions - Students

- Your name
- Previous job
- Why are you here?
- Guilty Pleasure

Rough Course Timeline

W1	W2	W3	W4	W5	W6
Ruby	Web/ Sinatra	Rails, Databases	Rails	Testing	Javascript
W7	W8	W9	W10	W11	W12
Ajax	External APIs	MV* Frameworks	Rails	Computer Science	Deployment, Optimization, Security

Course Expectations 1

- Mixed daily workflow of lecture, workshops/labs, homework, breaks
- After class reading, coding and study (homework)
- Working on final projects/portfolio

Course Expectations 2

- 1:1Meetings
- Progress tracking
- Reflection time
- Blogging/tweeting
- Collaboration/teamwork

Graduation Criteria

We want to be able to make *excellent recommendations* for students

In order to graduate from General Assembly's Web Development Immersive, students must:

- Make an honest attempt on ALL course assignments (homework and labs), submitting and completing a minimum of 80%. Students will receive timely feedback from instructors on their assignments throughout the course.

- Complete and submit the course final project, earning a satisfactory evaluation by completing all functional and technical requirements on the project rubric, including delivering a presentation.

Students who do not meet these criteria will not quality for WDI exclusive opportunities and events (e.g. Apprentice Program, hiring events, etc.).

In addition, students who consistently display a lack of commitment may be asked to leave the program.

Culture Discussion

aka- making sure you have what you need to learn

How to get help

- Think about what your question is. What is working, and what isn't?
- Identify the problem as specifically as possible. What is the background?
- Check your notes and slides
- Ask another student (Hipchat or in person)
- Google it!
- Ask on Stack Overflow, check IRC, or Github Issues
- Ask a team member
- Ask Kristin regarding class, student or instructor issues

Note Taking and Learning

- Not much note taking needed (although you're welcome to)
- Much more important to learn <u>how</u> to find something than to memorize it. *Concepts* > *Details*
- Alternating between lecture and exercise will reinforce learning
- Code examples will be annotated and shared on Github , along with links to resources and PDF copies of slides on Schoology

"*You don't become a great programmer by reading about programming.*
You become a great programmer by programming"

Our Tools and Resources

- **Schoology** - Central class communication and tracking tool. Grades, assignments
- **Hipchat** - Reach out to students during and after class for help
- **Github** - Central class code repository, both for examples and your work (covering how to in a bit)
- **Email** - Get in touch directly with the instructional team

Having stupid problems doesn’t mean you’re stupid

We're all in this together

And sometimes the documentation is poor

Software Development Process

Roles in Software Development

- Client/Management
- Project Managers
- UI/UX designers and art teams
- Software Developers

Agile vs Waterfall

Agile	Waterfall
Short term planning	Long term planning
Frequent client/dev interaction	Infrequent client/dev interaction
Short time to implement	Long time to implement
Easier to respond to changes	Harder to respond to changes
Short cycles of improvement	Long cycles of improvement

For more on these: www.unterstein.net/su/docs/CathBaz.pdf

Scrum agile development

Project Management Questions

- What is a feature?
- What is a bug?
- What is an enhancement?
- Example of a small feature
- Example of a large feature
- Breaking larger features into smaller ones

*nix basics

(there are several unix-like operating systems)

About Unix

- Unix created in 1969 by employees at Bell Labs
- Multitasking, multiuser operating system
- Linux* is a free open source unix-like OS released in 1991 by Linus Torvalds
- Ubuntu, Debian, Gentoo and Redhat are all examples of Linux *distributions*. Linux itself is really just the kernel, and a distribution is a collection of all the software you use with it.
- 90% of the 500 fastest computers in the world run Linux, all top 10 do
- Excellent for server deployments, but also on laptops, embedded and mobile devices

* if you want to be proper and not upset Richard Stallman, you should really say GNU/Linux

bash shell basics

- Commands all issued to a prompt
- Each command is a program itself
- Most commands take arguments
- We will focus on commands for the bash shell
- Standard filesystem is arranged in a tree
- Learn details about any command by typing ‘man’ and then the command - `man pwd`

bash commands overview

- `ls` - list contents of directory
- `cd` - change directory
- `pwd` - list current path
- `mkdir` - create directory
- `cp` - copy (file or directory)
- `mv` - move (file or directory)
- Examples at: https://gist.github.com/tibbon/5794257

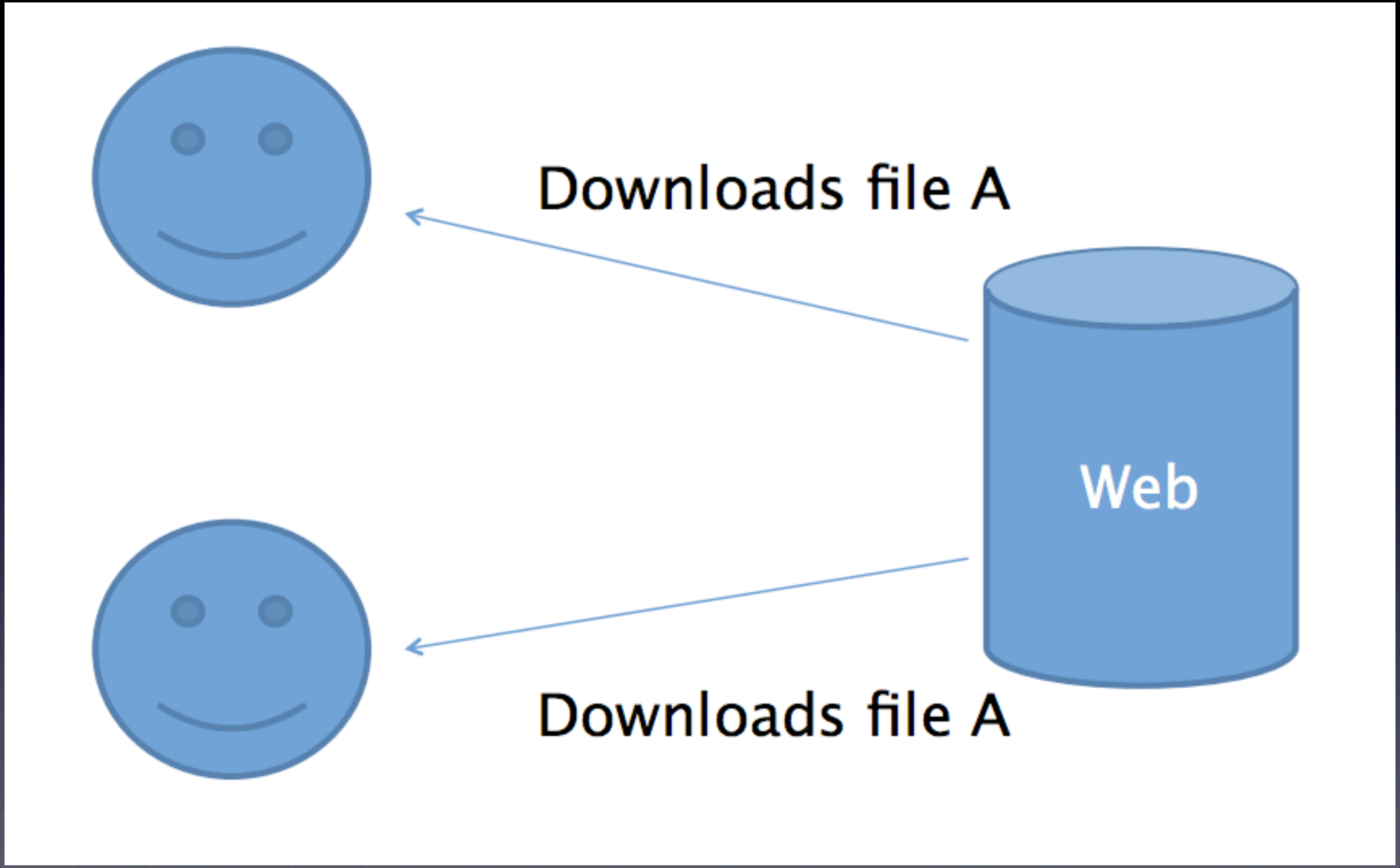
Learn more at: http://linuxcommand.org/lc3_learning_the_shell.php

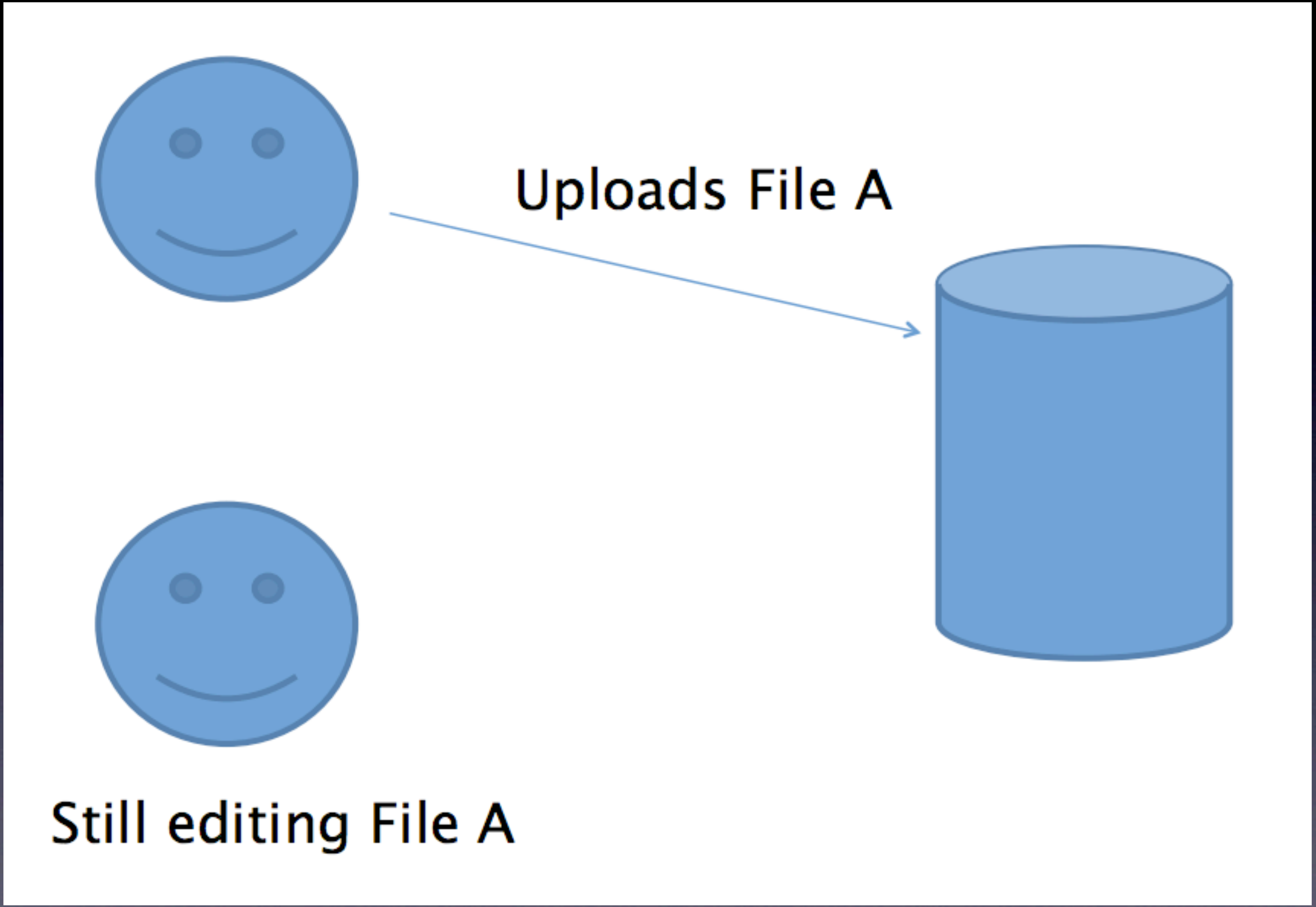
Version control with Git

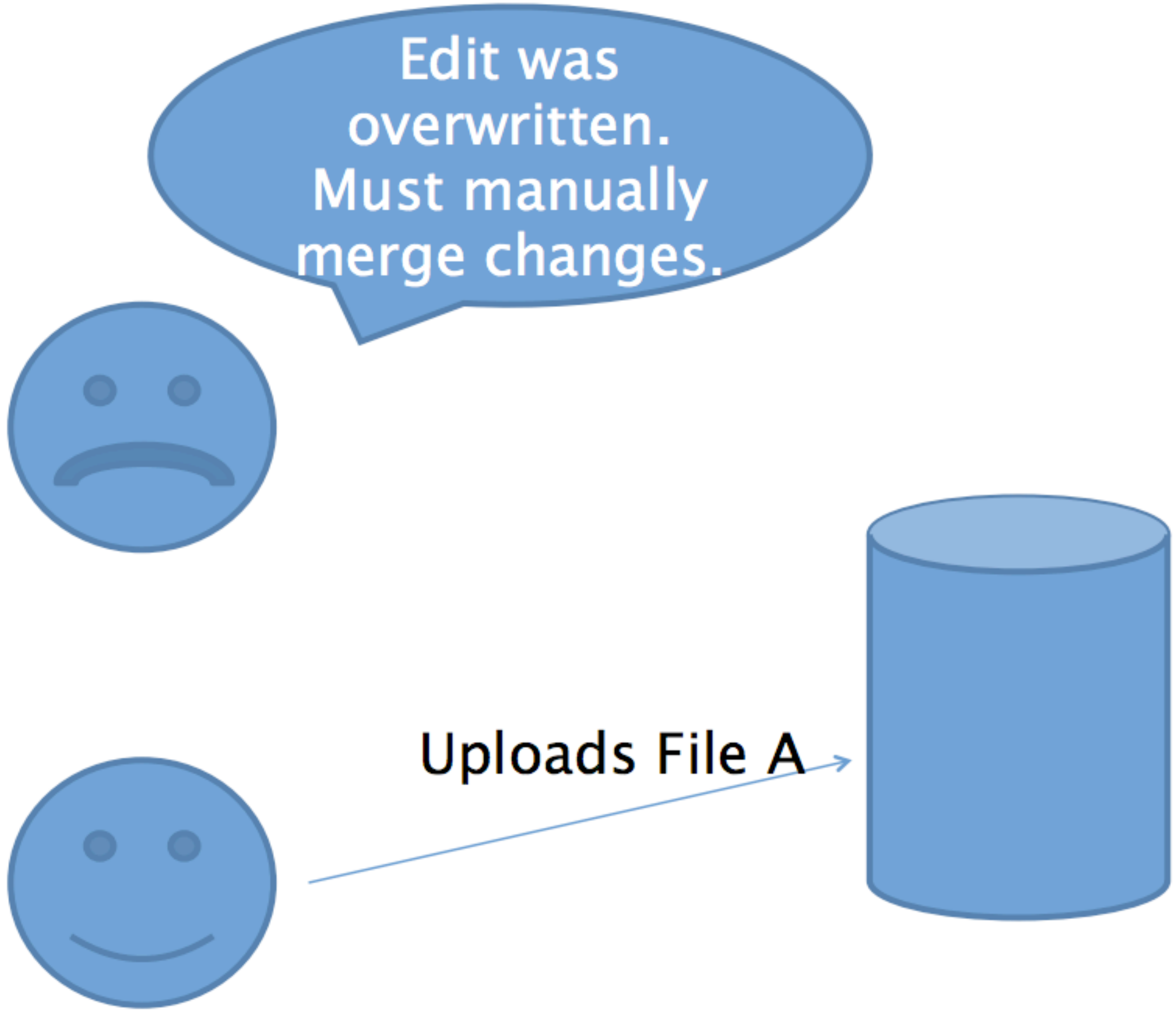
The following slides were inspired (with art also from) ones by Paul Synnott (@psynnott)
Originals at: http://www.tosbourn.com/wp-content/uploads/2011/03/GIT.pdf?eb7cc0

The Problem

(What we were doing before)

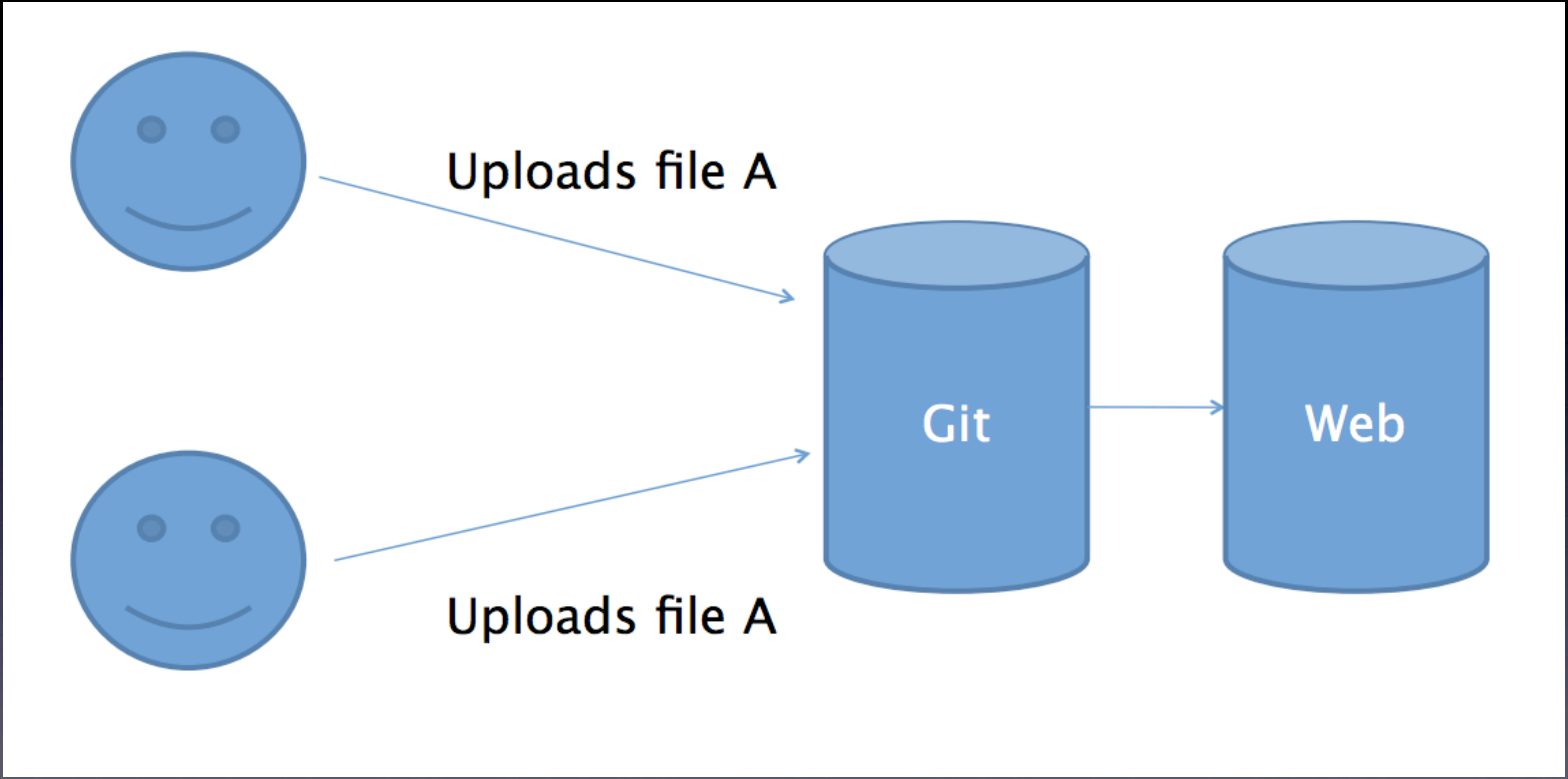

Downloads file A
Web
Downloads file A

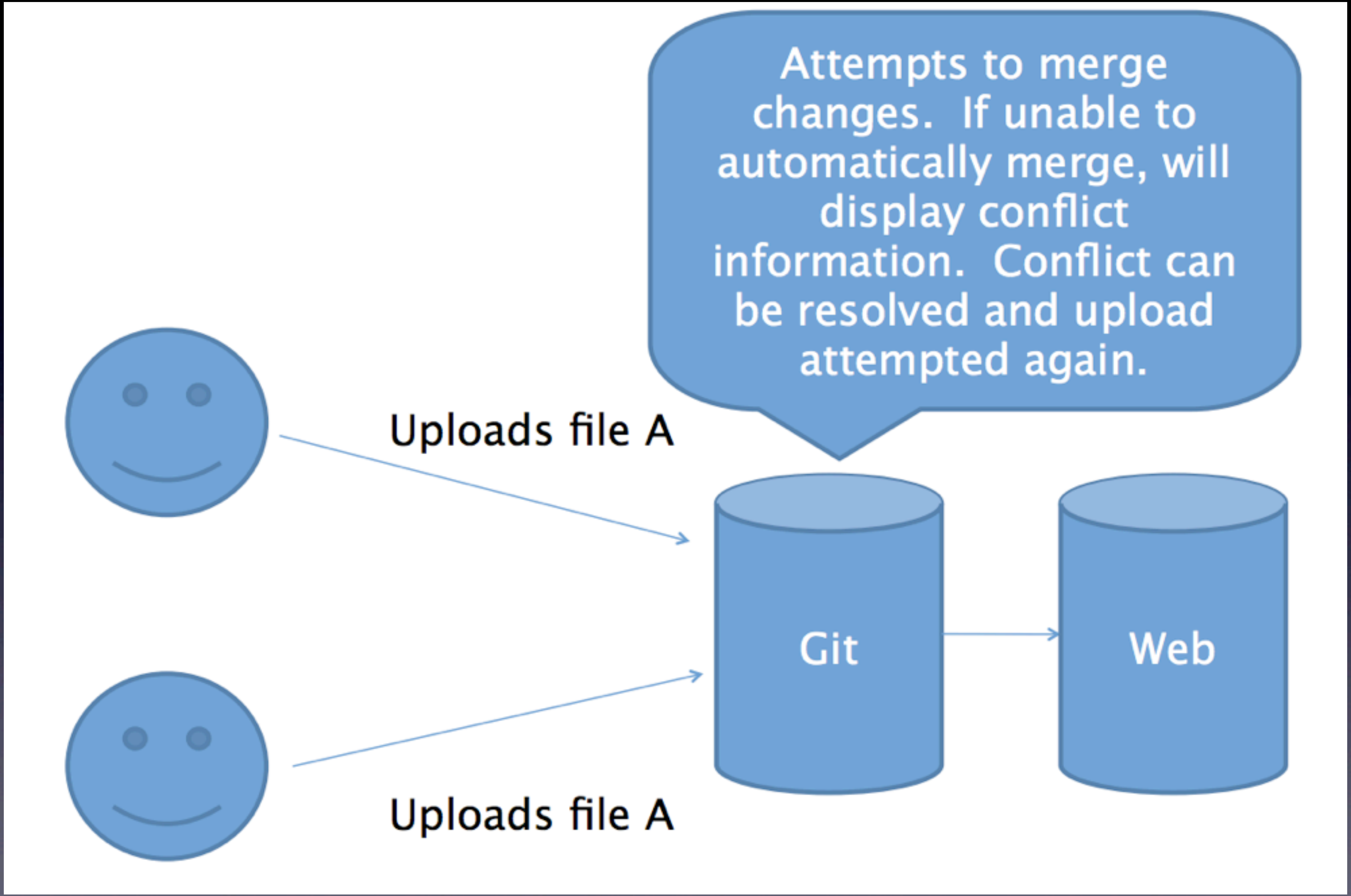

Uploads File A
Still editing File A


Edit was overwritten. Must manually merge changes.
Uploads File A

The Solution

Version Control (Git)


Uploads file A
Git
Web
Uploads file A


Attempts to merge changes. If unable to automatically merge, will display conflict information. Conflict can be resolved and upload attempted again.
Uploads file A
Git
Web
Uploads file A

This only scratches the surface

Notes about Git

- Was created to manage the Linux kernel version control (hundreds of developers)
- Every commit (save) is stored with a unique ID and message
- You can go backward in time, and see who did what
- You can *fork* the code into separate branches, *merge* them back together later and resolve *conflicts* in the merge
- Commit nonstop

Github

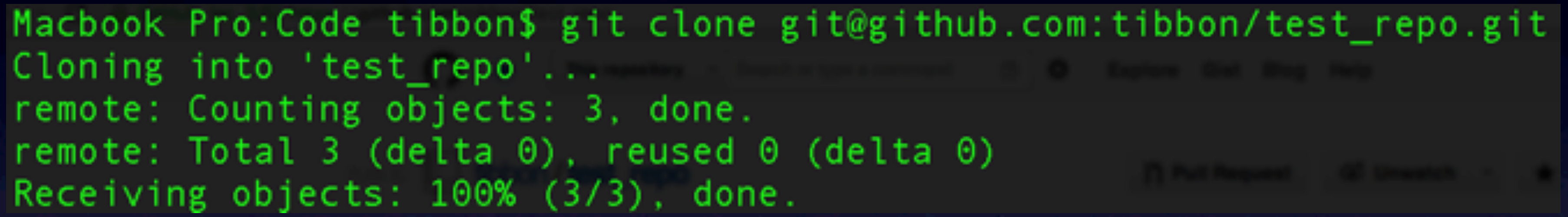
- Git hosting platform with free and paid accounts
- Great place to keep a portfolio of your code
- Incomplete/broken projects are ok to put out there!
- Be careful to not check in sensitive data like passwords, databases or API keys

Git Concepts

- Keeps track of changes in files that are *tracked* in the repo
- Every cloned repo contains a record of all changes
- Files can be ignored with a `.gitignore` file
- Local configuration is stored in a hidden `.git` directory
- Commits can be *pushed* to Github and other servers

Most Basic Git Workflow

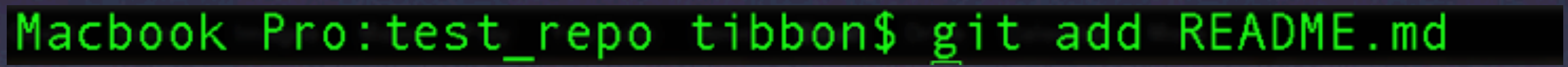
1. Clone a repo from Github



```
Macbook Pro:Code tibbon$ git clone git@github.com:tibbon/test_repo.git
Cloning into 'test_repo'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.
```

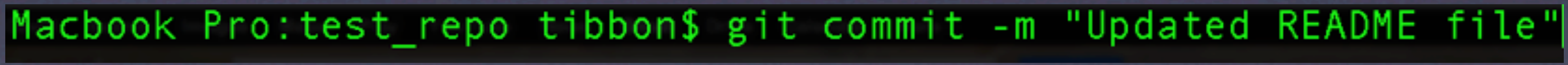
2. Make changes to files

3. Add files to a commit



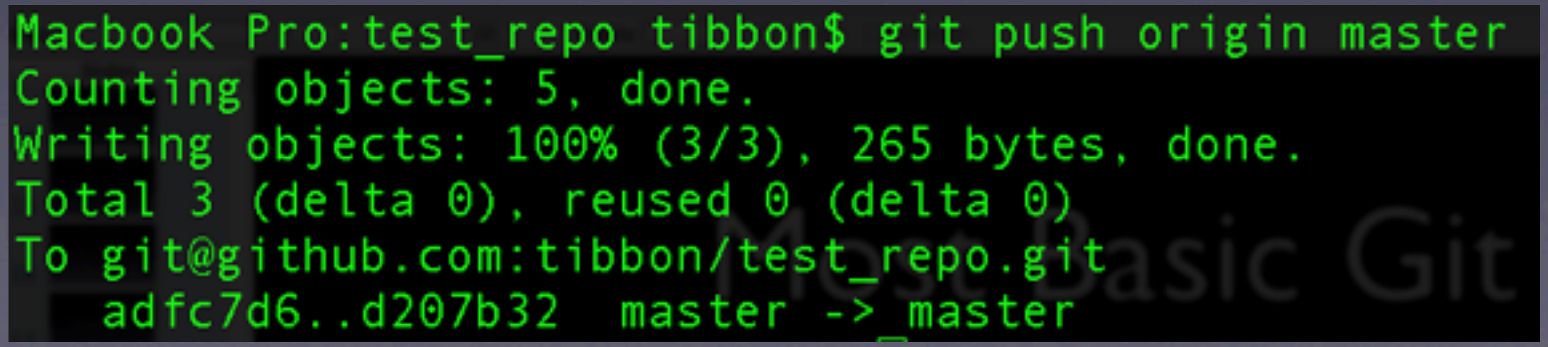
```
Macbook Pro:test_repo tibbon$ git add README.md
```

4. Create a commit & message



```
Macbook Pro:test_repo tibbon$ git commit -m "Updated README file"
```

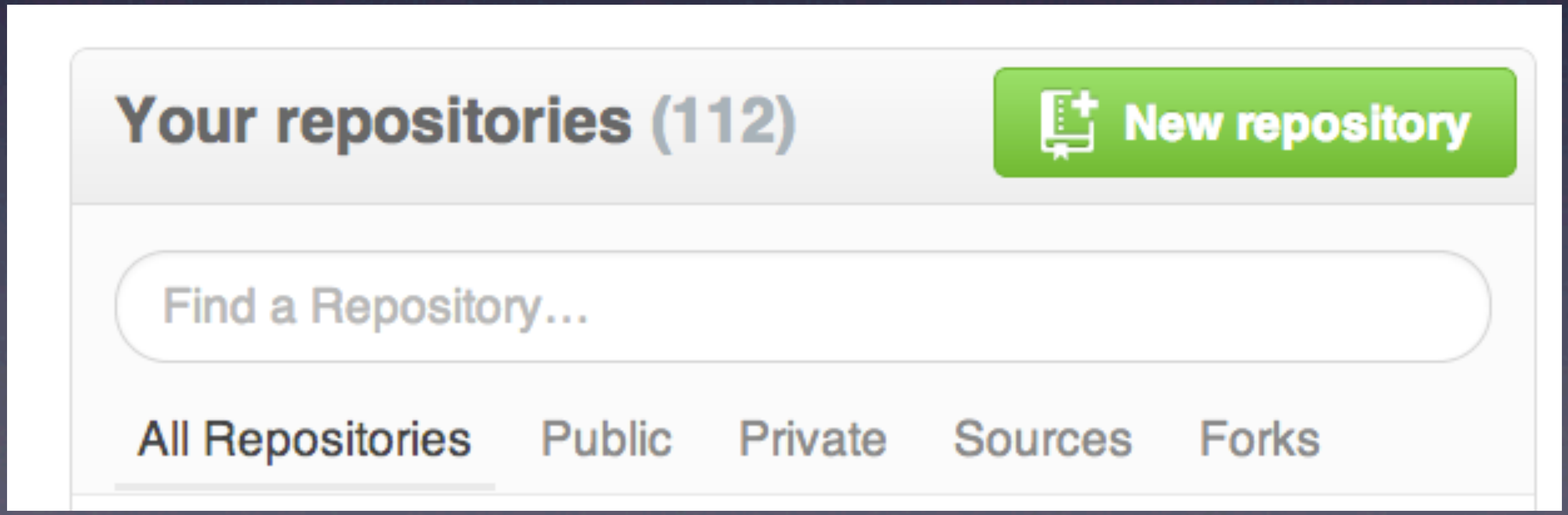
5. Push commits back to Github



```
Macbook Pro:test_repo tibbon$ git push origin master
Counting objects: 5, done.
Writing objects: 100% (3/3), 265 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:tibbon/test_repo.git
   adfc7d6..d207b32  master -> master
```

Creating Repos on Github

Unlimited public ones are free



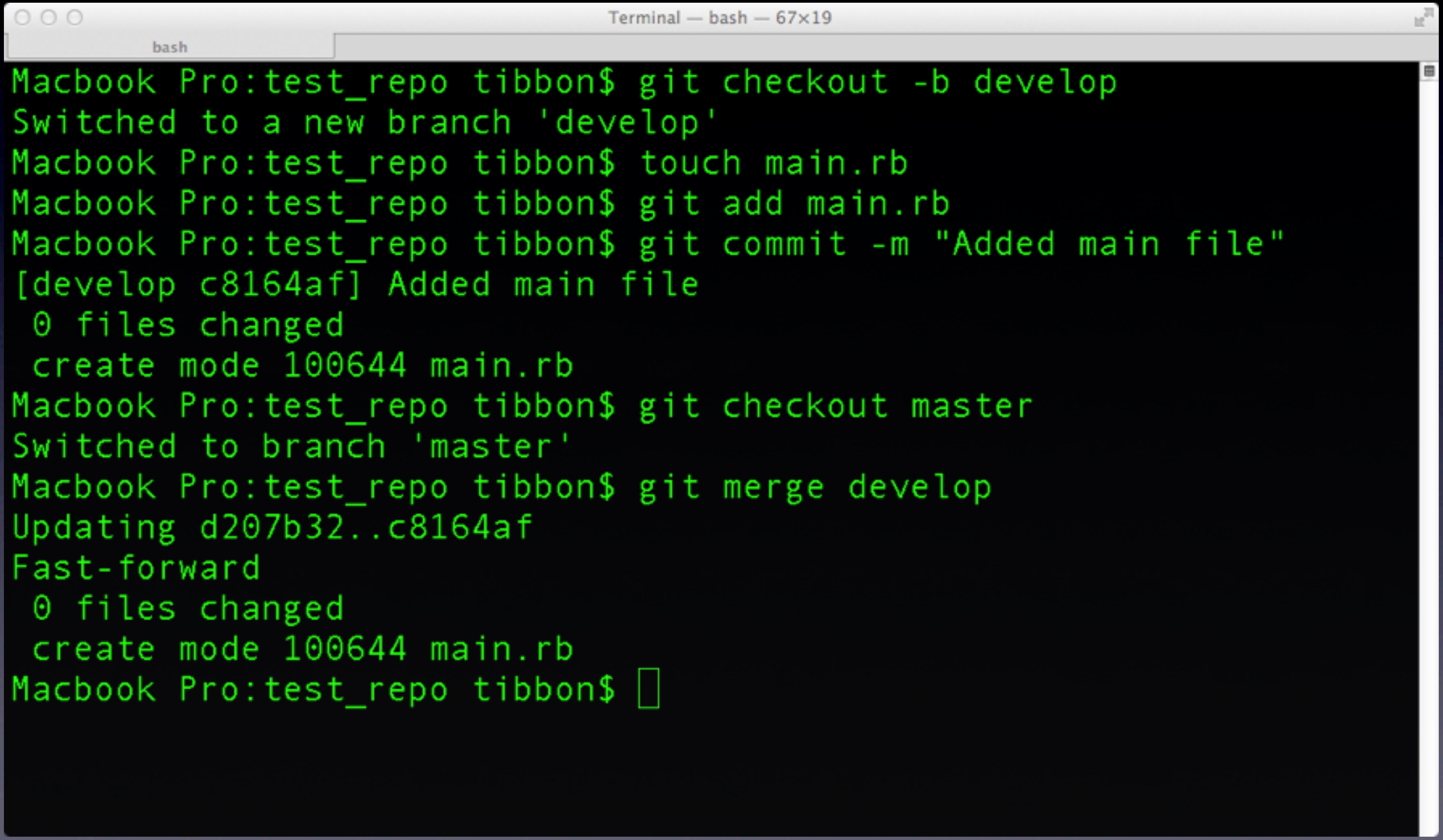
Git Branching and Merging

- Branches can be useful for developing individual features, enhancements or bug fixes
- Each repo can have unlimited branches
- Branches can be branches of branches and so on
- You can merge branches back together
- On Github you can make a fork, which is your own copy of a repo. These can be merged together as well, but its a little more complex

Git Branching Basic Workflow

- Clone or create a repo
- Create a branch
- Make changes and create commits
- Switch to other branch, merge in changes from other branch
- Sometimes you have to resolve conflicts if it can't decide how to merge them best

Git branching example



```
Macbook Pro:test_repo tibbon$ git checkout -b develop
Switched to a new branch 'develop'
Macbook Pro:test_repo tibbon$ touch main.rb
Macbook Pro:test_repo tibbon$ git add main.rb
Macbook Pro:test_repo tibbon$ git commit -m "Added main file"
[develop c8164af] Added main file
 0 files changed
 create mode 100644 main.rb
Macbook Pro:test_repo tibbon$ git checkout master
Switched to branch 'master'
Macbook Pro:test_repo tibbon$ git merge develop
Updating d207b32..c8164af
Fast-forward
 0 files changed
 create mode 100644 main.rb
Macbook Pro:test_repo tibbon$ 
```

Git - When do we use this?

- Creating new feature? Fixing a bug? -> New branch
- New feature working or bug fixed? Merge branch
- Use Git as part of your team's agile toolchain
- More reading:
  - http://nvie.com/posts/a-successful-git-branching-model/
  - https://github.com/nvie/gitflow
  - Exercise: Go through http://try.github.io/

Homework

- Create a text document with feature, bug and enhancement ideas for a piece of software you might use (1 of each type)
- Create a public Github repo called something like "Sublime_ideas" for the software you'd want improved
- Clone your Github repo, *add* this text document to a commit, *commit* it, and *push* it to Github
- Explore Github and find two interesting sounding projects. Star them and look through their code a bit. No need to understand the code yet, but getting used to reading unfamiliar code is important.

Monday Review

- Class expectations, resources and culture
- Software development process
- Sublime and bash shell basics
- Git and Github basics

Tomorrow

- The programming begins!
- Ruby Basics (datatypes, input, output, calculations)
- Functions