



Computer Systems and Networks

ECPE 170 – Instructor Dr. Pallipuram– University of the Pacific

Version Control

These slides are credited to Dr. Jeff Shafer

Lab Schedule

➤ Today

➤ **Lab 2 – Version Control**

➤ Next Week

➤ Intro to C (for C++ programmers)

➤ **Lab 3 – C Programming / Build Tools**

➤ Deadlines

➤ **Lab 1 Report – Sep 6, 2021 by 11:59 PM**

➤ Submit via Canvas

➤ **Lab 2 Report – Sep 8, 2021 by 11:59 PM**

➤ Submit via version control

Before Version Control

1. <Report.doc>
2. <Report.doc.bak>
3. <Report-1.doc>
4. Email off to partner...
5. <Report-2.doc>
6. Partner responds with doc
(that is missing the changes you just made)
7. <Report-2a.doc>
8. <Report-2a-WITH-REFERENCES.doc>
9. Email off to partner...
Partner responds with new doc
<Report-3.doc>
10. <Report-3-FINAL.doc>
11. <Report-3-FINAL-OOPS-FIXED-TYPO-FINAL.doc>

Version Control Features

- Project history tracking
- Concurrent file editing (merges)
- Non-linear program history (branches)
- Naming scheme for program releases (tags)

Motivation for Version Control

- **Why would a single programmer (working alone) use version control?**
 - Backup files
 - Roll-back to earlier (working) version
 - See changes made between current (broken) code and earlier (working) code
 - Maintain multiple versions of a single product
 - Experiment with a new feature
 - Try a risky change in a “sandbox”
 - If it works, you can merge it into the regular code. If it fails, you can throw it away.

Motivation for Version Control

- **Why would a small group of developers use version control?**
 - All the reasons a single programmer would, plus...
 - Merging different changes made by different developers into the same file
 - Add a new function at the bottom? Safe to automatically merge in
 - Re-write a function at the same time another developer is also editing it? Version control will catch this and ask you to decide which edits should “win”
 - Blame – who wrote this buggy code?!?

Motivation for Version Control

- Why would a large group of developers use version control?
- Different question: Could you develop the Linux kernel, Adobe Photoshop, Google Chrome, etc... using:
 - A single shared “folder of code”?
 - Emailing code snippets between developers?
 - Everyone sits around and shares one keyboard?

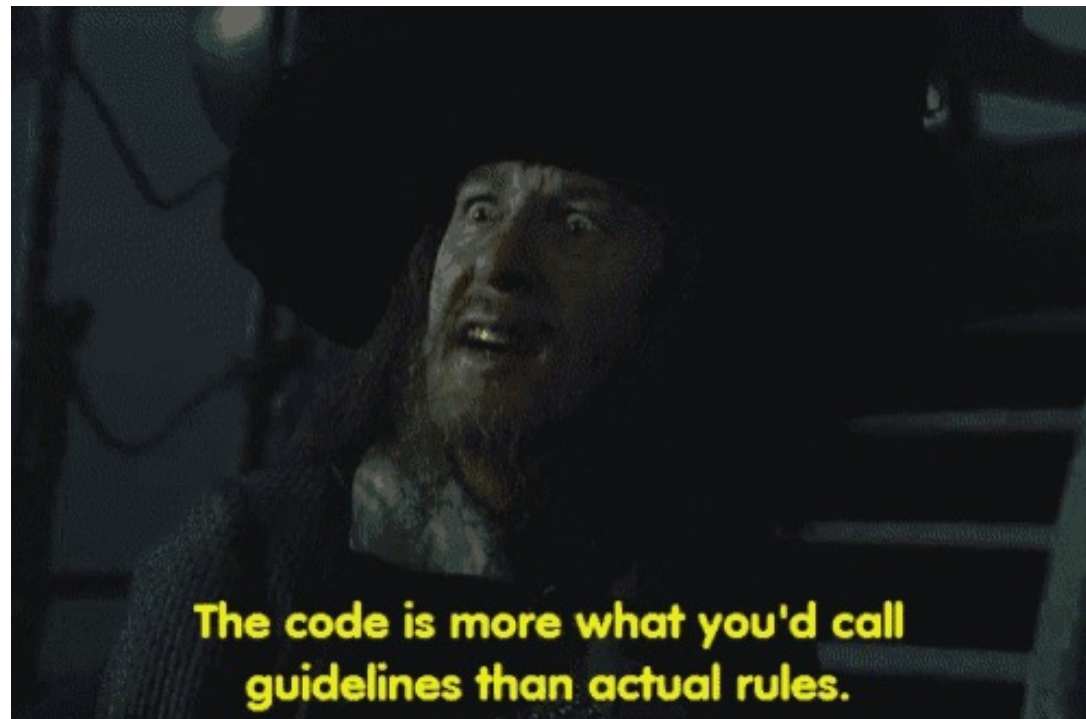
Version Control Basics

➤ What kind of files should I keep in version control?

- Program source code (*obviously*)
- VHDL / Verilog files (from digital design class)
- Matlab scripts
- HTML files
- Server configuration files
 - Imagine you work at Livermore National Labs, and your job is to manage Linux cluster computers with 100,000+ machines (nodes)...
- **Anything that is plain text!**

Version Control Basics

- What kind of files should I not keep in version control?



Version Control Basics

- **What kind of files should I not keep in version control?**
 - *These are more what you'd call "guidelines" than actual "rules"...*
 - **Binary data**
 - How do you *merge* two different binary files together? No general-purpose way to do this
 - **Anything auto-generated by the compiler**
 - Object files or executable file
 - Wastes space on useless junk that can be re-created automatically
 - **Text editor temp files (e.g. `main.c~`)**

Version Control Basics

- **Big risk in putting the executable in version control**
 - If you forget to compile before a commit, the executable may not be **in sync** with the attached source code!
 - **Big headache if you ever roll back to this version!**
- **In ECPE 170, all our executable files can be produced in under 5 seconds with one command. There's no need to include them in your repository**

Problem 1 – Comparison

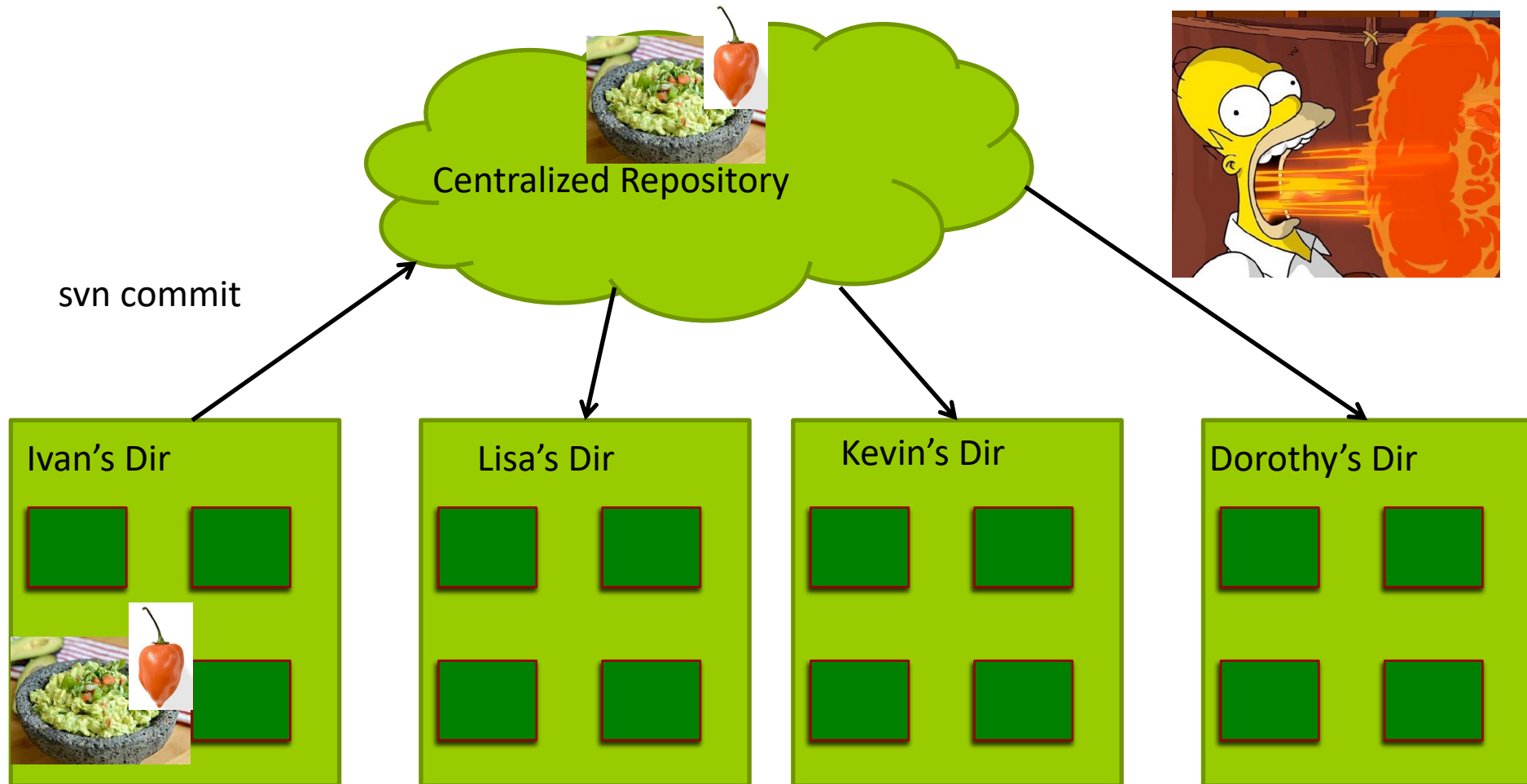
- How are these Version Control Systems different?
 - Git
 - Mercurial
 - SVN

P1

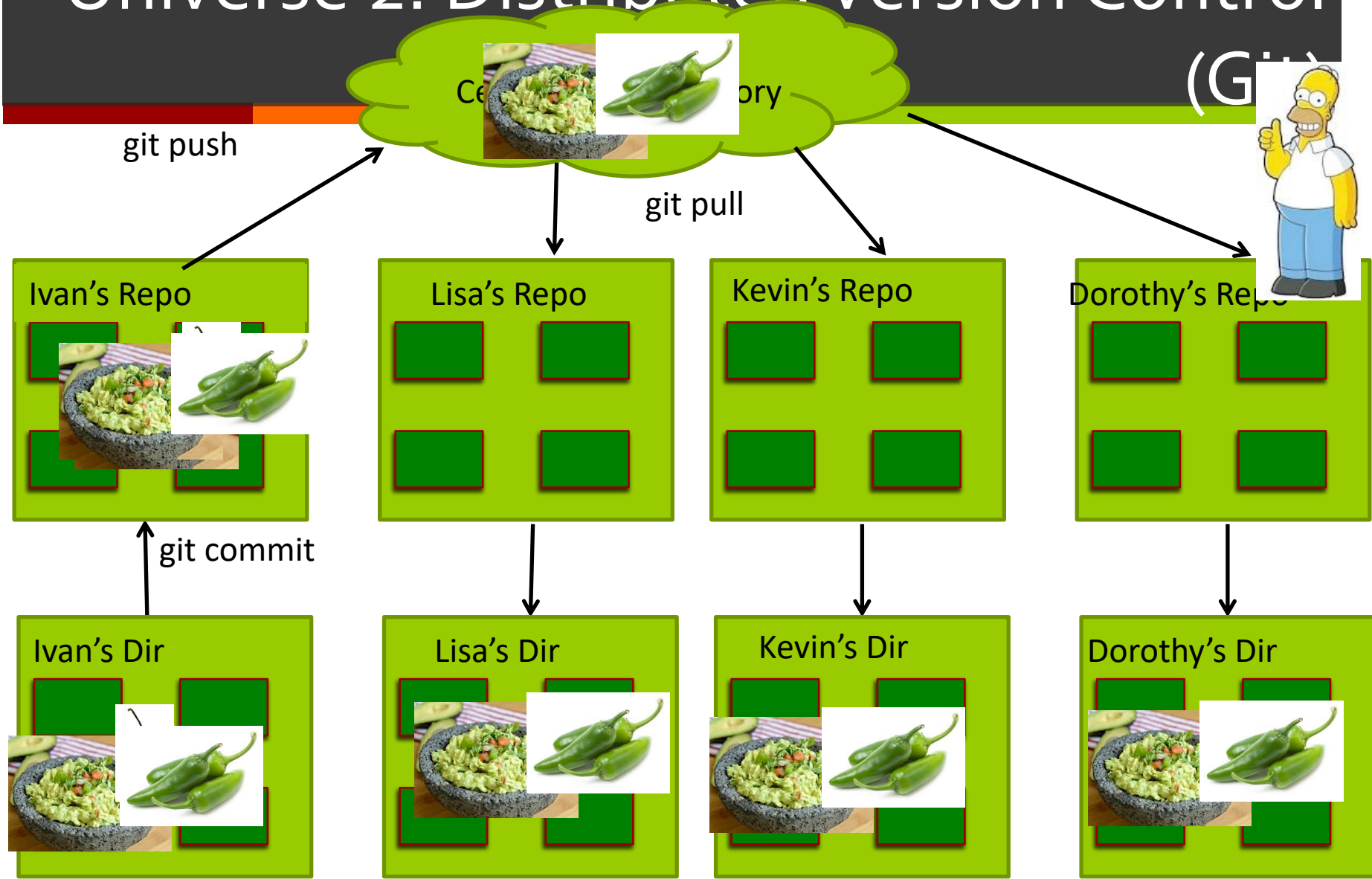
Distributed Version Control

- **Why do they call Git a distributed version control system?**
 - Conventional systems (e.g., Subversion or “svn”) have a centralized server hold the “master” copy
 - Distributed version control – each copy is its own full-fledged master! (But you can still push changes from one person’s copy to another)
 - Allows version control to work offline
 - Allows version control to work with ad-hoc groups

Universe 1: Centralized Version Control (SVN)



Universe 2: Distributed Version Control



Git Command Flow (usually)

1. `git clone <repository address>`
 - a. #get repo on your desktop
2. `git add <filenames>` #always specify a filename to add
 - a. #add new files and make changes
3. `git commit -m <meaningful commit message>`
 - a. #commit to your repo. Also use `-a` to commit changed files
 - b. Make changes and repeat 3
4. `git push` #All done? Let everyone see

Problem 2 – Git Cheat Sheet

➤ Go find a Git cheat sheet (or 2) for future reference



Version Control in ECPE 170




- Version control **required** for this class
 - Used to distribute boilerplate code for labs
 - Used to turn in assignments when finished

Version Control in ECPE 170

- *If you only do one check-in at the very end of your project, you've missed the whole point of version control, and turned a valuable tool into an obstacle to completing the assignment*
- **Check-in code on a regular basis!**

In case of fire



-  1. git commit
-  2. git push
-  3. leave building

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



"If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything."

<http://xkcd.com/1597/>

Problem 3 – Multiple Heads

- Research and answer question 3 on your own, and then begin the lab!

