Sustainable Engineering Design

CSE 5408 – California State University, San Bernardino

Version control systems

Lab Schedule

Activities

- **This Week**
  - Lab3: Git Basics
  - Lab4: Git Advanced-Distributed VCS

Wednesday Class will be Open Session:

You need to watch this Tutorials: https://www.youtube.com/watch?v=3RjQznt-8kE&list=PL4cUxeGkcC9goXbgTDQ0n_4TBzOO0ocPR

Assignments Due

- **Lab 2**
  - **Due by Feb. 15th 11:45 am**
- **Lab 3**
  - **Due by Feb. 22nd 11:45 am**
- **HW1**
  - **Due by Feb. 22nd 11:45 am**

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_v1.doc>
12. Report-3-FINAL_v2.doc>
13. Report-3-FINAL_forsure.doc>
14. Report-3-FINAL_forsure_v1.doc>

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
  - 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
    - Git keeps track of it using the concept of *change sets*
    - What changed? where? Moved where in the file?

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

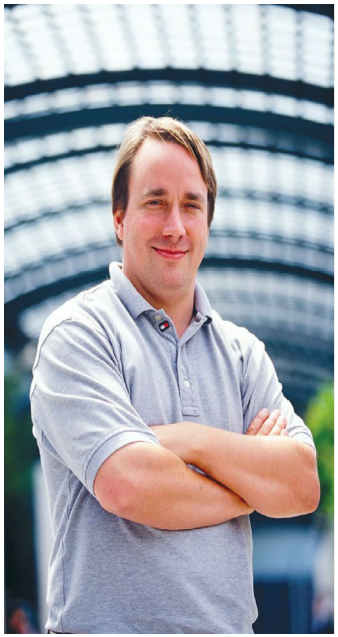
- What kind of files should I keep in version control?
  - Program source code *(obviously)*
  - VHDL / Verilog files (from digital design class)
  - Matlab scripts (from DSP and Image Processing)
  - HTML files
  - Server configuration files
    - Imagine you work at CSUSB ITS, and your job is to manage 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?
  - *These aren't "rules", so much as "guidelines"…*
  - **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~`)

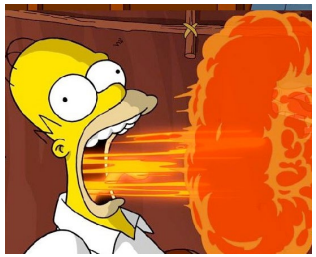
What is GitHub

- GitHub:
  - is a code hosting platform for version control and collaboration.
  - It lets you and others work together on projects from anywhere.

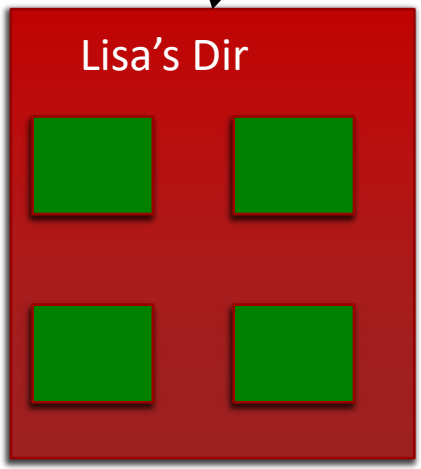


created by Linus Torvalds in 2005 to develop Linux Kernel

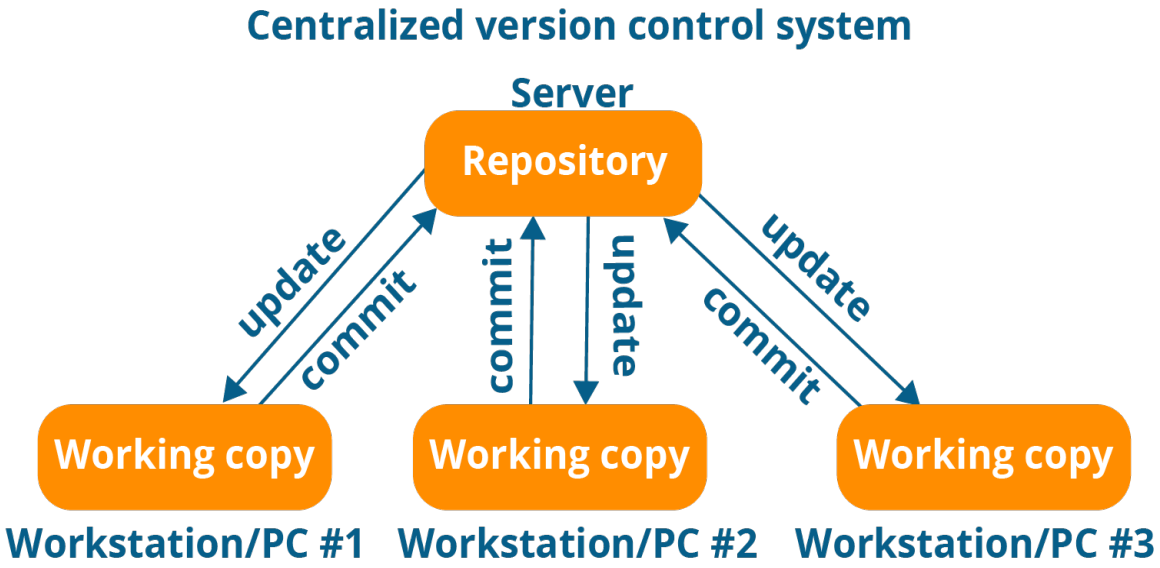
Universe 1: Centralized Version Control SVN



svn commit

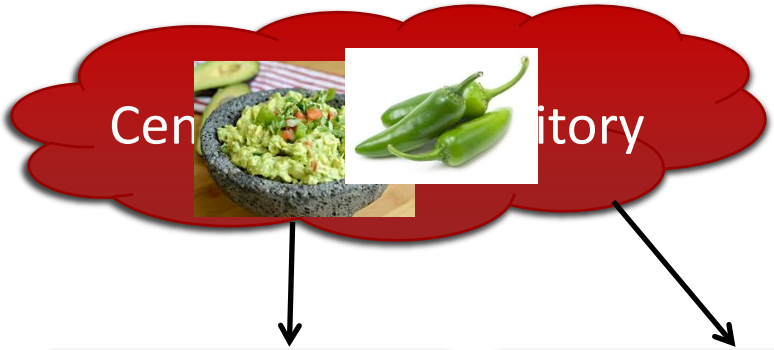


Centralized VCS: Single Repository



- *Repo is not locally available;*
- *Everything is centralized, if the central server getting crashed or corrupted will result in losing the entire data of the project.*

Universe 2: Distributed Version Control Git



git push

Ivan's Repo

Lisa's Repo

Kevin's Repo

Dorothy's Repo

git commit

Ivan's Dir

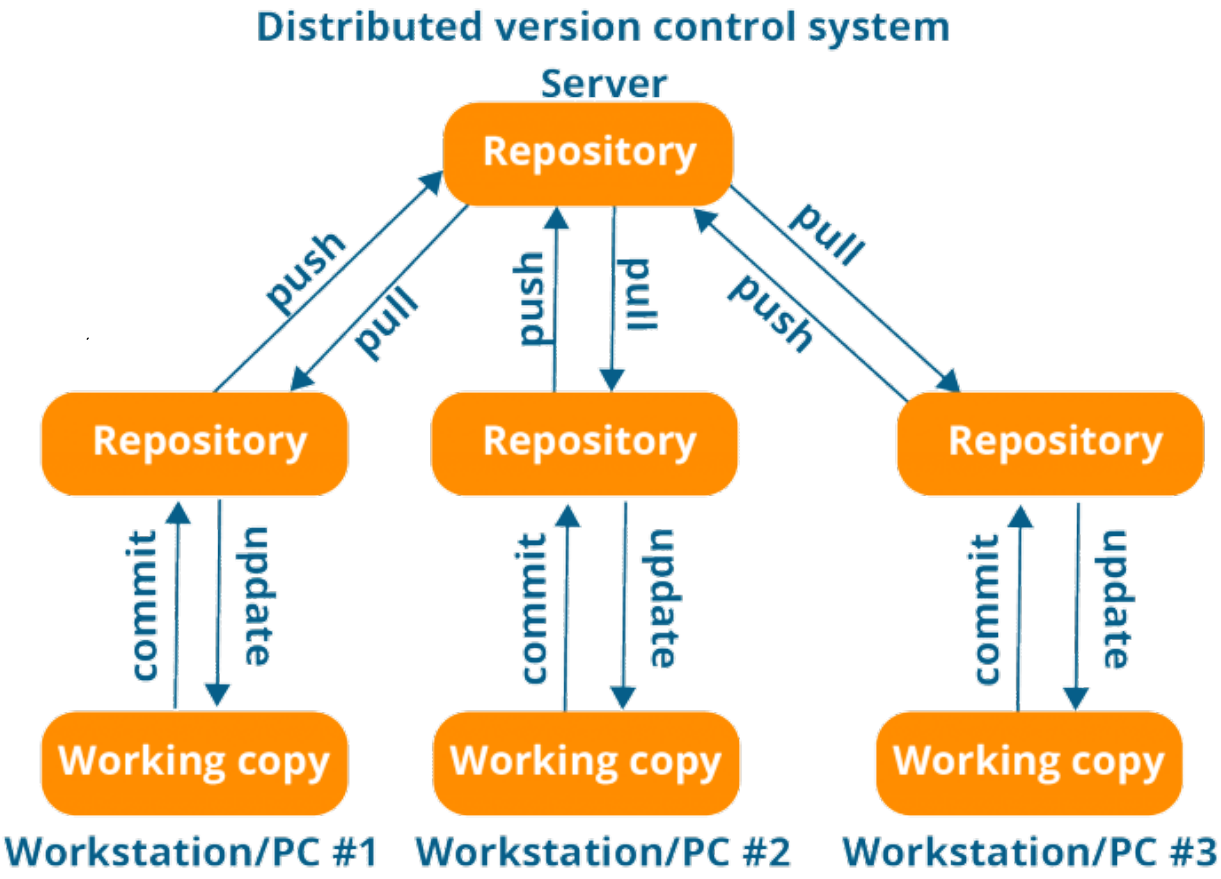
Lisa's Dir

Kevin's Dir

Dorothy's Dir

Distributed VCS

- *every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive.*
- *They can commit and update their local repository without any interference*.
- **Pull** and **Push**

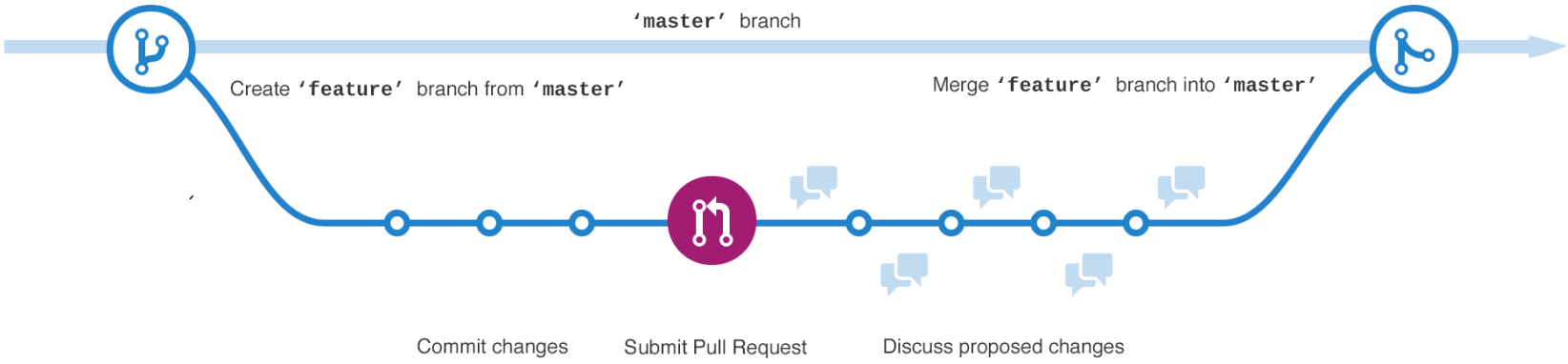


VCS concepts

- **Repository (repo)**: The database storing the files.
- **Server**: The computer storing the repo.
- **Client**: The computer connecting to the repo.
- **Working Set/Working Copy**: Your local directory of files, where you make changes.
- **Branch**: Create a separate copy of a file/folder for private use (bug fixing, testing, etc). Branch is both a verb (“branch the code”) and a noun (“Which branch is it in?”).
- **Merge** : Apply the changes from one file to another, to bring it up-to-date. For example, you can merge features from one branch into another.

GitHub: Master vs Branch - feature

https://guides.github.com/introduction/flow/



Git Command Flow (usually)

1. `git clone` <repository address>

   #get repo on your desktop

2. `git add` <filenames> #always specify a filename to add

   #add new files and make changes

3. `git commit -m` <meaningful commit message>

   #commit to your repo. Also use –a to commit changed files

   Make changes and repeat 3

4. `git push #All done? Let everyone see #push to remote repository`

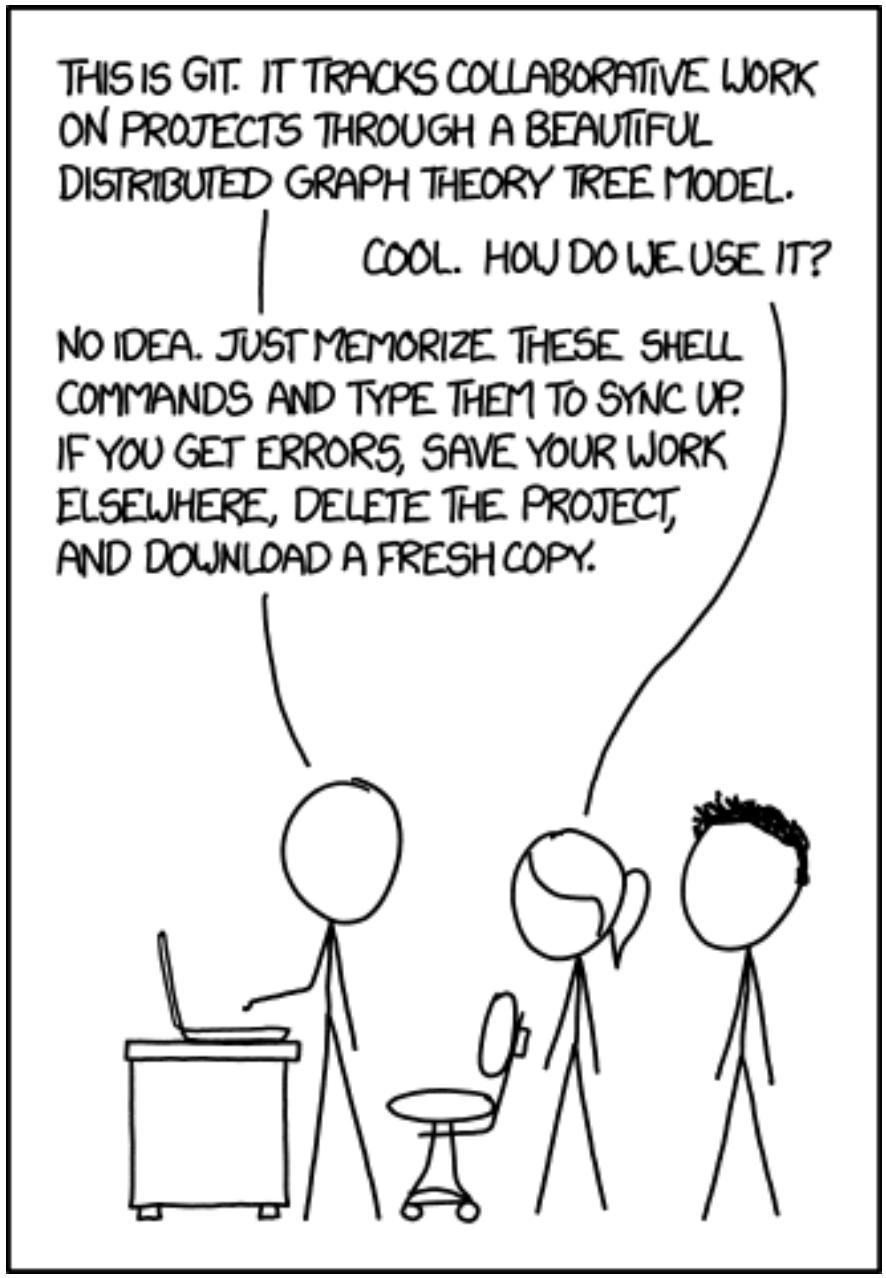
Git Command Flow (usually)

1. `git add` <file> #Add files to index
2. `git init` #initialize local git repository
3. `git status` #Check status of working tree
4. `git branch` <branch_name> #create a new branch
5. `git checkout` <branch_name> #change the working branch

Take home message

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


In case of fire


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.