

Version Control



Siddharth Kaza
Towson University, Computer and Information Sciences

What is version control? Why should you care?

- ▶ Video

- ▶ <http://git-scm.com/video/what-is-version-control>

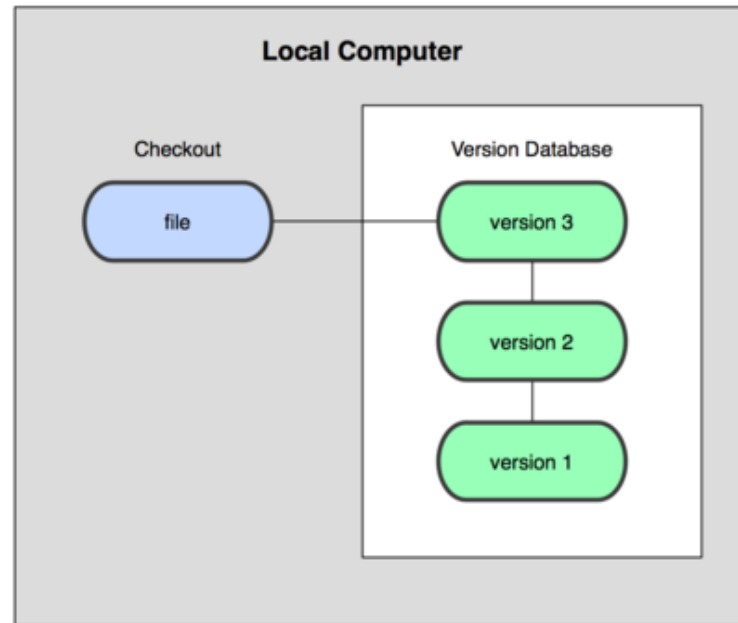


What is version control? Why should you care?

- ▶ Version control is a system that records changes to a file over time so that you can recall specific versions later.
- ▶ It allows you to:
 - ▶ revert files back to a previous state,
 - ▶ revert the entire project back to a previous state,
 - ▶ compare changes over time,
 - ▶ see who last modified something that might be causing a problem,
 - ▶ who introduced an issue and when, and more.
- ▶ Using a VCS also generally means that if you break things or lose files, you can easily recover.

Local version control system (VCS)

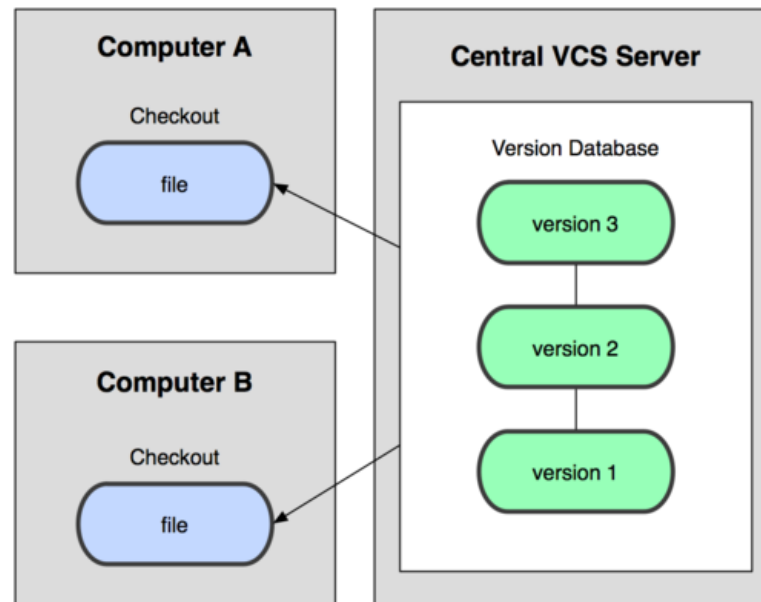
- ▶ Local VCSs had simple databases that kept all the changes to files under revision control.



- ▶ E.g., 'rcs'. But, you need a centralized version control if you have to share files

Centralized Version Control Systems

- ▶ These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- ▶ e.g. CVS, Subversion, and Perforce



Centralized VCSs (cont.)

▶ Advantages

- ▶ Everyone know what others are working on (collaboration)
- ▶ Administrators have control over who can do what (control)
- ▶ Knowledge of who did what (auditing)

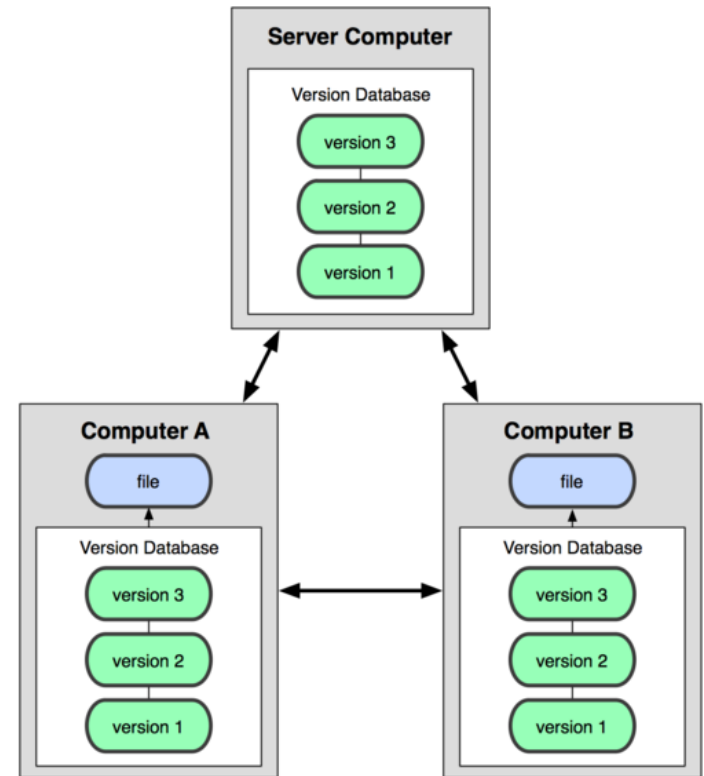
▶ Disadvantages

- ▶ Single point of failure
- ▶ Need to be online



Distributed Version Control Systems

- ▶ In DVCS - clients don't just check out the latest snapshot of the files: they fully mirror the repository.
- ▶ Every checkout is really a full backup of all the data.
- ▶ Many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways in the same project.
- ▶ Eg. Git, mercurial, bazaar, darcs



Using Subversion this semester

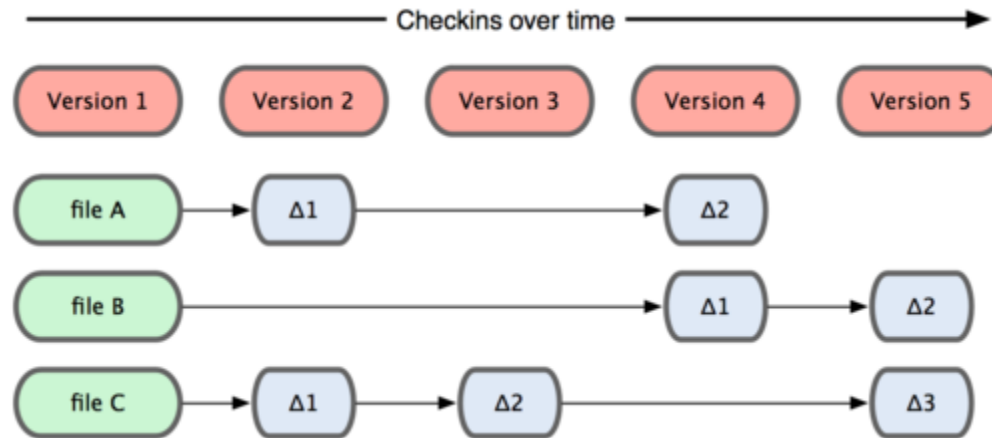
▶ History

- ▶ Subversion is an open source version control system.
- ▶ Founded in 2000 by CollabNet, Inc



Subversion basics

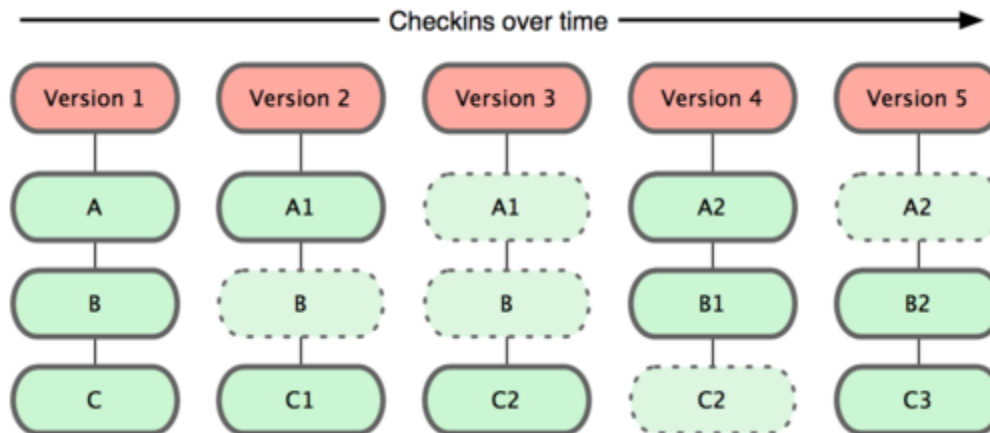
- ▶ Many VCS (subversion, CVS, Bazaar) consider information they keep as a set of files and the changes made to each file over time.



- ▶ Git does not do it this way

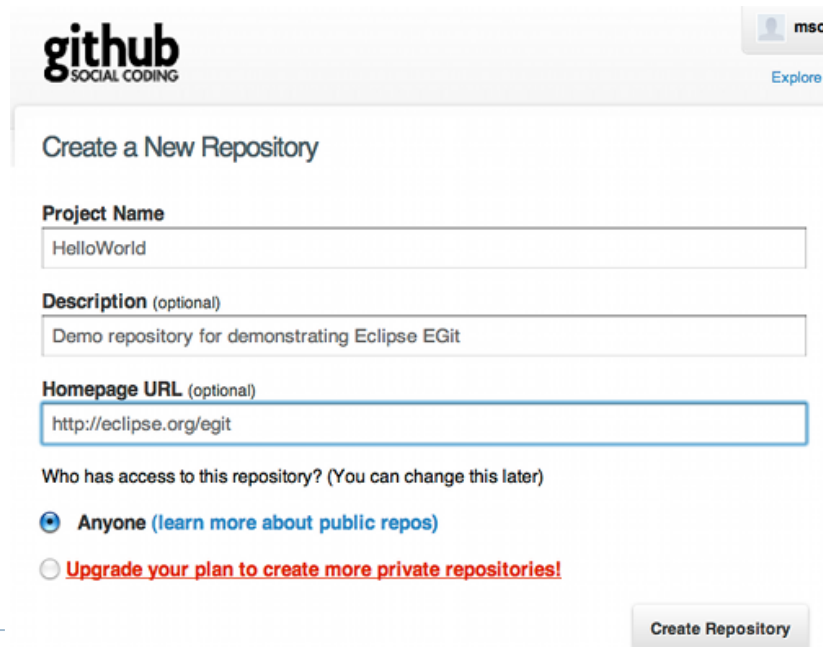
Git basics

- ▶ Git thinks of its data more like a set of snapshots of a mini filesystem.
- ▶ Every time you commit, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.



Creating remote repository

- ▶ For remote repository – use GitHub (Google code is an option but we won't use that)
- ▶ Create a account on GitHub (<https://github.com/>) – I assume you are using the free version.
 - ▶ All team members should create one
- ▶ Create a new repository on Github (search their tutorial)



The screenshot shows the GitHub 'Create a New Repository' page. At the top, the GitHub logo and 'SOCIAL CODING' tagline are visible. A user profile icon for 'mso' and an 'Explore' link are in the top right. The main heading is 'Create a New Repository'. Below this, there are three input fields: 'Project Name' with the value 'HelloWorld', 'Description (optional)' with the value 'Demo repository for demonstrating Eclipse EGit', and 'Homepage URL (optional)' with the value 'http://eclipse.org/egit'. Below the input fields, a question asks 'Who has access to this repository? (You can change this later)'. There are two radio button options: 'Anyone (learn more about public repos)' which is selected, and 'Upgrade your plan to create more private repositories!'. At the bottom right, there is a 'Create Repository' button.

github
SOCIAL CODING

mso
Explore

Create a New Repository

Project Name
HelloWorld

Description (optional)
Demo repository for demonstrating Eclipse EGit

Homepage URL (optional)
http://eclipse.org/egit

Who has access to this repository? (You can change this later)

☒ Anyone (learn more about public repos)

☐ Upgrade your plan to create more private repositories!

Create Repository

Creating remote repository (cont.)

- ▶ **Select the option “initialize new repository with README”**
- ▶ Also add .gitignore and pick the option Rails
 - ▶ .gitignore file defines the files that are not submitted in the repository when you commit
 - ▶ In Rails – this includes log files, database, and several others
 - ▶ You can open the newly created .gitignore to view them



Create SSH Keys for all team members

- ▶ SSH keys will allow all teams members to access the github repository from their machine without using passwords.
- ▶ When your computers' key is added to a project on github then it becomes a 'trusted' computer
- ▶ Follow instructions here:
 - ▶ <https://help.github.com/articles/generating-ssh-keys/>
- ▶ Each team member needs to do this for every computer they will use to code



Pushing skeleton rails proj to remote github

- ▶ **Only one team member does this.**
- ▶ Start new project in Rubymine 'cloning' the git repo
 - ▶ Close all projects – file > close project
 - ▶ In the welcome to rubymine window > check out from VCS
 - ▶ Enter SSH URL given by github
- ▶ Now the project has been cloned (copied over).
 - ▶ Since it is empty on github – all you see is an empty project
- ▶ **If you are using sublime (or another IDE), you can use the git command line.**
 - ▶ **git clone <ssh_url>**



Pushing skeleton rails proj to remote github (cont.)

- ▶ **Only one team member does this.**
 - ▶ Now create a new rails project by:
 - ▶ Opening terminal in project
 - ▶ Entering the command “rails new .” – which creates a new rails project in folder
 - ▶ Don't overwrite .gitignore when prompted
 - ▶ Make sure your skeleton project is running
 - ▶ Stage and Commit the new files
 - ▶ Right click on project > Git > Add (adds to staging area, see slide 10)
 - If using command line: `git add .`
 - ▶ Right click on project > Git > Commit
 - Command line: `git commit`
 - ▶ Use tutorial (<http://www.jetbrains.com/ruby/webhelp/using-git-integration.html>) if needed.
-



Pushing to remote repository

- ▶ Right click > Git > Commit > 'Commit and Push' (button)
- ▶ Give a good message
- ▶ Hit "Push"
 - ▶ Command line: `git push`
- ▶ Check on GitHub to see if everything was pushed.
- ▶ If you are using the VM and you see that the files were committed under my name "Siddharth Kaza". Then you need to follow instructions here to set the username:
 - ▶ <https://help.github.com/articles/setting-your-username-in-git/>



Collaborating using Git – Mock Exercise

- ▶ This is needed when one team member (say, Walter) created the skeleton and now the second team member (Jesse) wants to start contributing.
- ▶ First – Jesse (and any other team members) needs to create a github account
- ▶ Walter needs to add Jesse as a collaborator on the project
 - ▶ This needs to be done on the github website under the “Settings” option. Use Jesse’s username.
 - ▶ Once done, Jesse will see Walter’s project under his Github page



Collaborating using Git cont.

▶ Jesse:

- ▶ Start new project in Rubymine 'cloning' the git repo
- ▶ Now the project has been cloned (copied over).
- ▶ When you change a file or create new ones
 - ▶ notice the decorations changing - the changed file becomes a different color in the file tree
- ▶ Right click on project > git > add and then 'commit'
 - ▶ Give a good comment
 - ▶ Select all files
 - ▶ This commits to your local repo, not the github repo owned by Walter (unless you used the 'commit and push')
 - ▶ Notice the files Jesse changed are no longer a different color



Collaborating using Git cont.

- ▶ Now the Jesse's changes are **committed** to his local repo. He needs to “push” them to the remote repo (owned by Walter)
 - ▶ Cannot push if not committed
- ▶ VCS (on top menu) > git > push
- ▶ Changes go up to github
- ▶ Walter still does not see them
 - ▶ He needs to pull them down and merge them into his own local code
- ▶ VCS > git > pull
 - ▶ Please see the changes you are bringing down before moving on
 - ▶ You select the branch you are merging into your own main branch*
 - ▶ Assuming there are no conflicts code will be merged and show

▶ * Code branches are not being discussed here – study them on own if interested

Resolving conflicts

- ▶ If both Jesse and Walter make changes to the same file (same line) – this causes conflict when merging.
- ▶ If that happens then the conflict has to be resolved manually (by editing code, not by whatever Walter and Jesse did to resolve conflicts*).

Conflicts

- ▶ Resolving conflicts is not easy.
- ▶ See <http://www.jetbrains.com/ruby/webhelp/resolving-conflicts.html>
- ▶ After you are done with resolving conflict
 - ▶ Either manually, or by copying code from one file to another, or merging
 - ▶ Right click on file > git > add
 - ▶ Then commit.



More info on Git

- ▶ <http://www.jetbrains.com/ruby/webhelp/version-control-with-rubymine-2.html>
- ▶ There is more help to be found on the web.

