

# cs304

# Software Engineering

**TAN, Shin Hwei**

陈馨慧

Southern University of Science and Technology

Slides adapted from cs427 (UIUC) and cs304( SUSTech)

# Questions from previous class

How to join GitHub classroom?

A You do not need to join GitHub classroom now. Invitation link will be available for each assignment/project.

Can the group members for the final project register for different lab sessions?

A It is not recommended for selecting different lab sessions because (1) you may need to discuss with your group members during certain lab sessions, and (2) you will need to present together during final presentation

Can I choose different language?

A You should use Java (projects with mixture of Java and other languages is fine) because most tools introduced in class do not support other languages

# Platforms for teaching

- A few platforms for ensuring the effectiveness of communication for e-learning. These platforms include:
- 1.. **Wechat group:** you will need install wechat work(企业微信) for assessing the wechat group (the QR code is attached). the wechat group consists of (i) students taking the class, and (ii) student helpers who will be helping you during the lab session and grading your project.
- 2. **GitHub classroom:** you will be using GitHub classroom for submitting your assignments. You only need to create a GitHub account (only create if you don't have a GitHub account) and we will linked your GitHub account to your student id during the first assignment (the first assignment will be posted later).
- 3. **Face-to-face lecture:** Thanks to GitHub classroom. Once we have the first face-to-face lecture, students who participate actively in class will get some small gifts (e.g., stickers and booklets) from GitHub. As 10% of your grade comes from class participation, I hope that this will encourage you to actively participate during lecture.

# Selection of issues

- The class project is a group assignment. **Each group could have only 4-6 members.**
- **Detailed instructions have been uploaded to Sakai**
- **Do not choose an open issue that has pull requests**

Wiki Security Insights

Cache.asMap() interface makes it possible to map NULL values into the cache New issue

#5342

Open pontusp opened this issue on 3 Dec 2020 · 2 comments · May be fixed by #5348

**pontusp** commented on 3 Dec 2020

Through the `asMap()` interface it is possible in certain situations to partially map NULL values into the `Cache` which results in an unpredictable state. When using `ConcurrentMap.compute` and `ConcurrentMap.computeIfPresent` it possible to remap a previous value to null rather than removing it, which I think would be the expected behaviour here.

Attached [CacheNullValueTest.java.zip](#) contains test cases that reproduces this problem, tested on 30.0-jre

**ben-manes** commented on 4 Dec 2020 · edited

The test case is confusing because it passes due to asserting the invalid behavior. I suspect that the Guava team would prefer to have a failing test to iterate against.

The compute methods are not a good fit with Guava's design, so the implementation has quirks and performance issues. I don't think those methods are commonly used, so they are likely not fully robust.

Can you try [Caffeine](#)? I believe my migration of your test passes, but flipping assertions leads me a little unsure. The builder is almost identical but since `async` is enabled by default, you'd need to specify `Caffeine.executor(Runnable::run)` to mirror Guava.

**netdpb** added `P3` `package=cache` `type=defect` labels on 5 Dec 2020

**ben-manes** linked a pull request that will close this issue on 5 Dec 2020

Fix compatibility between the cache compute methods and a load #5348 Open

Assignees  
No one assigned

Labels  
`P3` `package=cache` `type=defect`

Projects  
None yet

Milestone  
No milestone

Linked pull requests  
Successfully merging a pull request may close this issue.  
[Fix compatibility between the cache compute ...](#)

Notifications  
[Subscribe](#)  
You're not receiving notifications from this thread.

3 participants

[https://github.com/google/guava/i  
ssues](https://github.com/google/guava/issues)

open but has a  
linked pull request

# Communication with developers

- Get approval from developer about your intention to contribute

github.com/assertj/assertj-core/issues/1645

Facebook Google 翻译 Google 学术搜索 正在播放多功能老... e 2

**Closed** ShouldContainXXX: allow to replace "elements" by a more specific name #1645  
joel-costigliola opened this issue on 19 Oct 2019 · 5 comments

Same thing for `shouldcontain` .

joel-costigliola changed the title ~~ShouldContainOnly: allow to replace "elements" by a more specific name~~  
ShouldContainXXX: allow to replace "elements" by a more specific name on 19 Oct 2019

joel-costigliola added `good first issue` `improvement` labels on 8 Mar 2020

WuYff commented on 6 Apr 2020 • edited Contributor

Hi @joel-costigliola ,  
I would like to work on this issue.

4

joel-costigliola commented on 6 Apr 2020 Member Author

Go for it @WuYff!

WuYff commented on 9 Apr 2020 Contributor

Hi @joel-costigliola ,  
I tried to implement this improvement.  
My idea is: If the method is called through the class of Arrays/Maps/Spliterators, it will print the corresponding name. If not, I will use the class name of the parameter *Object actual* as the name.

<https://github.com/assertj/assertj-core/issues/1645>


- Leave a comment saying “I would like to work on this issue”
- Tell developer your idea of the implementation to check if it matches the requirement


# Project Resources


## All Project Resources available in Sakai


[All site files](#) ▾ / [CS304 CS 304 spring2021 Resources](#) / Project Resources


Move Copy Move to Trash Show Hide


 ☐ [Title](#) ^


 [Project Resources](#)

☐  [GitHub-OSS Fixit Website](#)

☐  [google-java-style-guide.pdf](#)

☐  [resourcenaming\\_cheatsheet.pdf](#)

☐  [Video on "How to start working on open-source project from 0"](#)

☐  [如何参与 Apache 项目社区\(Chinese\).](#)

- Video on “How to start working on open-source project” is recommended

# Today's goals

- Get more familiar with Version Control in SVN & GitHub
- Get more familiar with build systems such as Gradle.

This lecture is meant to directly help you with your upcoming project.

# Software configuration management (SCM)

- Four aspects
  - Change control
  - Version control
  - Building
  - Releasing
- Supported by tools
- Requires expertise and oversight
- More important on large projects



# SCM

- Four aspects
  - Change control
  - **Version control**
  - **Building**
  - Releasing
- Supported by tools
- Requires expertise and oversight
- More important on large projects

Continuously editing a file...

How to keep track of changes?

# The Good Old Way

- The good old way:

myFile-1.txt

myFile-2.txt

myFile-3.txt

myFile-4.txt

What did I change? When did that happen??

# The Good Old Way

- The good old way:

myFile-1.txt

myFile-2.txt

myFile-3.txt

myFile-4.txt

myFile-  
aug4morning.txt

myFile-aug4later-  
in-the-morning.txt

myFile-aug4-  
evening.txt

myFile-  
aug9.txt

myFile-  
final.txt

myFile-  
latest.txt

myFile-  
last.txt

myFile-final-  
final.txt



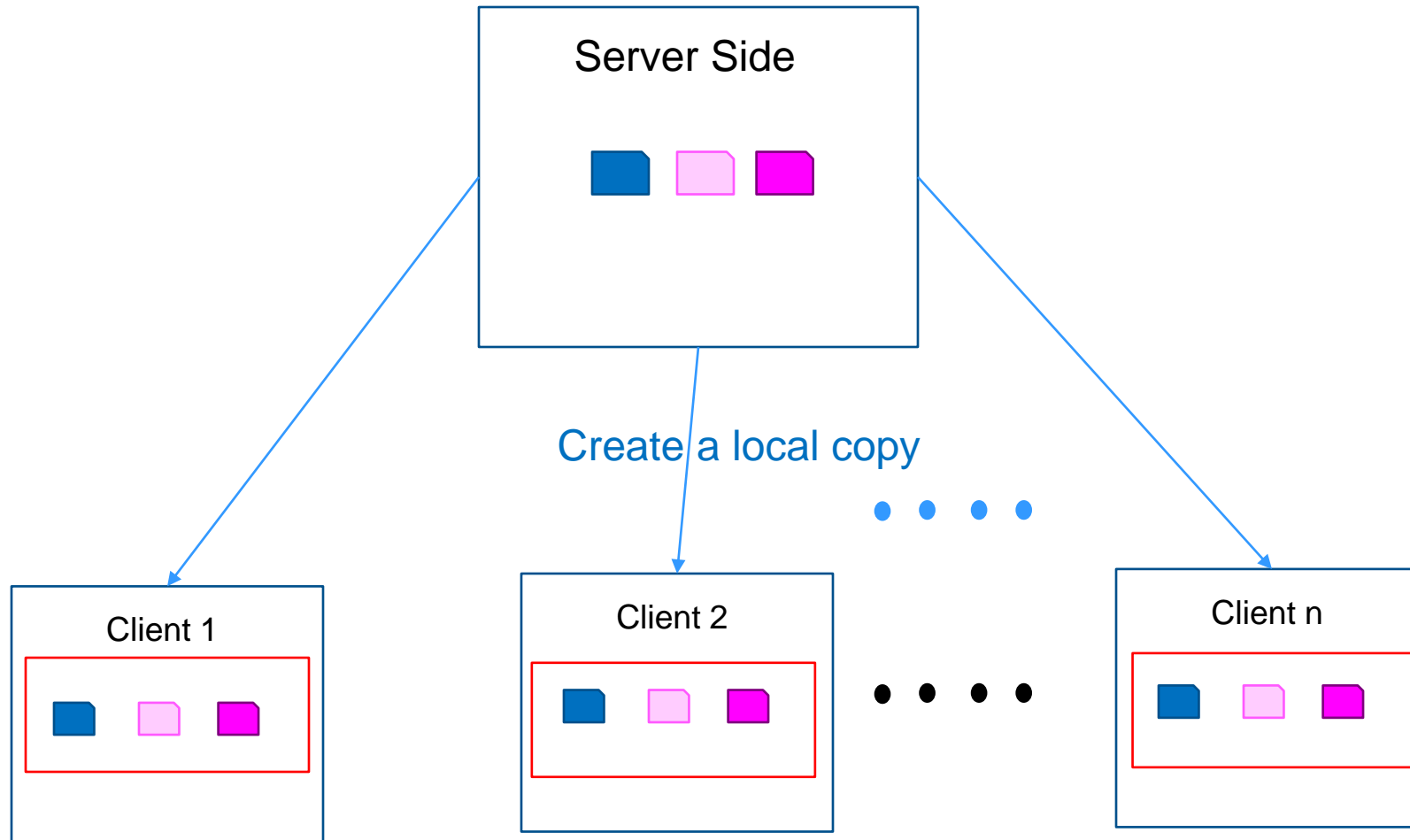
## Version Control To The Rescue!



# VCS: What and Why?

- A **V**ersion **C**ontrol **S**ystem is a software system that keeps track of the changes made to a set of files so that you can recall a specific version.
- It gives you the power to:
  - collaborate on a project with multiple other developers, merging changes and resolving conflicts
  - revert changes
  - go back in time to a specific version (tags can be your friend)

# VCS: Quick Overview



# SVN

## Apache Subversion





# What is the command for creating local copy in SVN?

---

svn checkout <address\_to\_remote> <name\_of\_local\_dir>  
git checkout <branch-name>

# Creating a Local Copy In SVN

- To create a local copy of the remote server:

`svn checkout` <address\_to\_remote> <name\_of\_local\_dir>

## Example:

`svn checkout` <https://subversion.ews.illinois.edu/svn/fa15-cs427/><netid> `cs427`

- A local directory named cs427 is created
- The data at the remote address on ews is copied into the local directory cs427
- SVN is setup to track any changes you do to the local copy

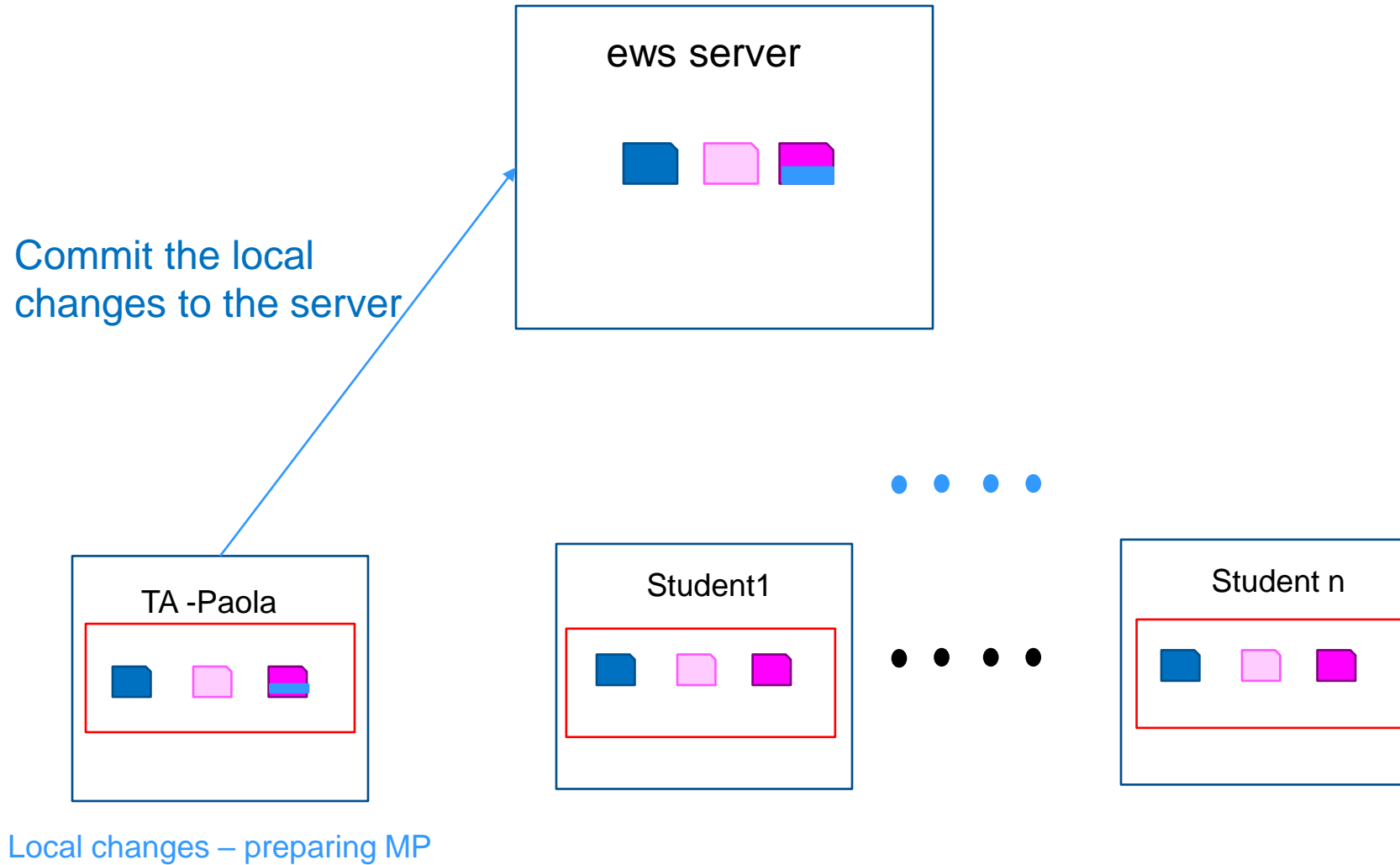
# What is the command for committing local changes?

---

svn commit -m "msg"

git commit -m "msg"

# VCS: Quick Overview



# Committing Local Changes To A Tracked File In SVN

1-21

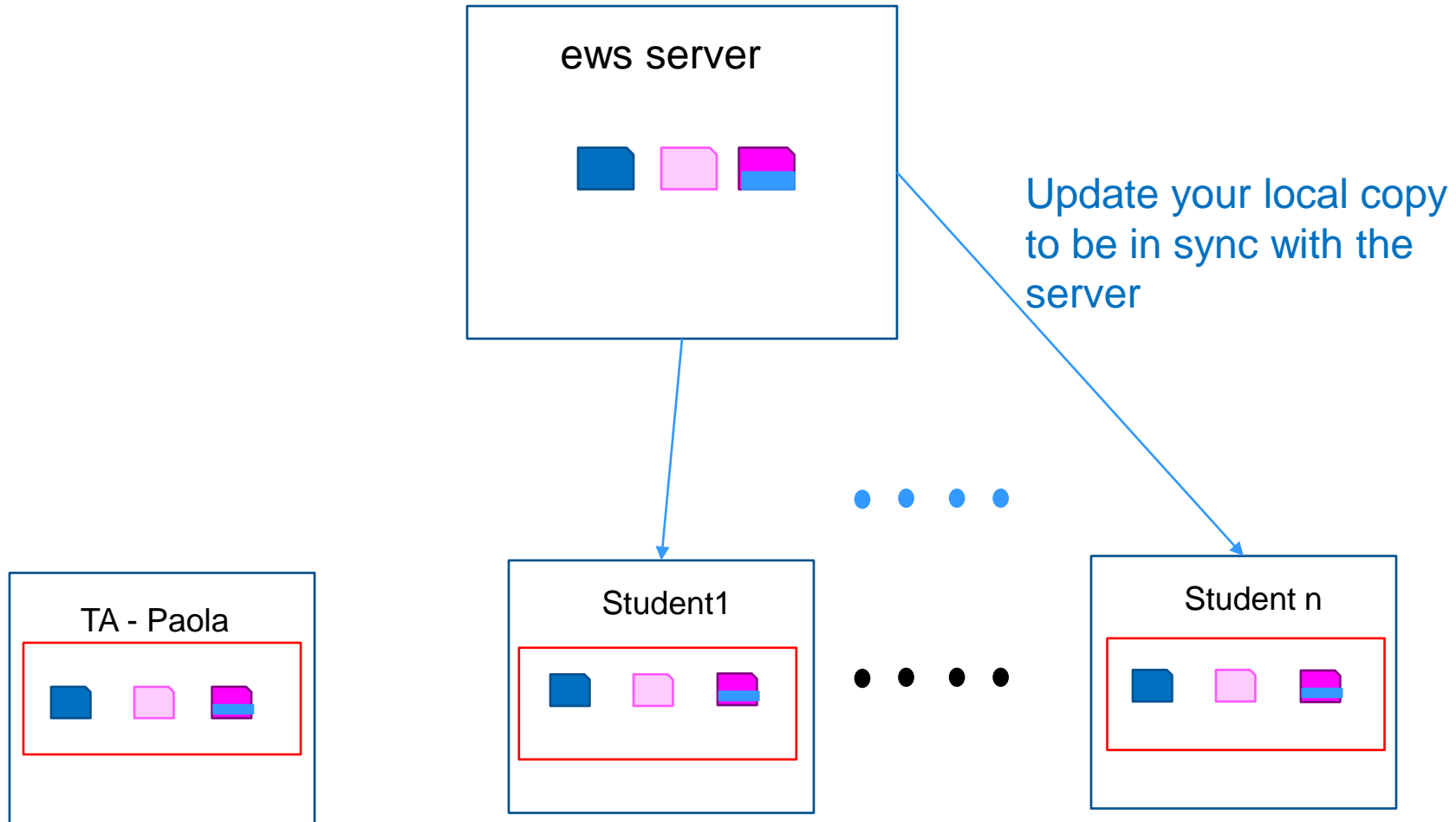
- To commit the local changes made to a file that svn is tracking to the server:

```
svn commit -m "<a meaningful descriptive message>"
```

For example:

```
svn commit -m "updated the grades for mp0"
```

# VCS: Quick Overview



# What is the command for updating local copy?

---

- `svn up`
- `git pull upstream master`

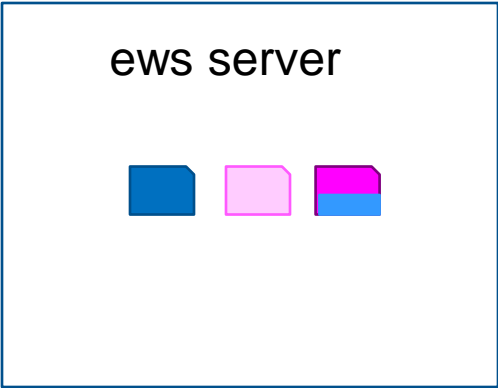
# Updating Local Copy From Server In SVN

- To update your local copy with the changes on the server:

`svn up`

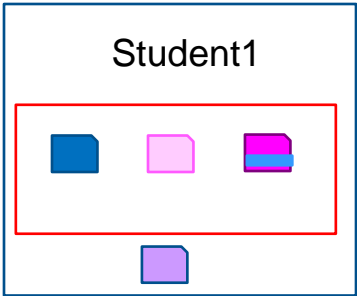
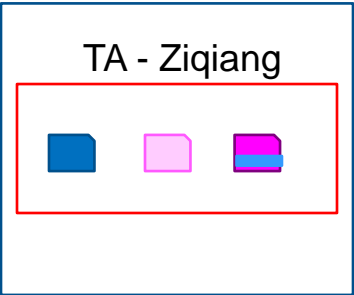


# VCS: Quick Overview

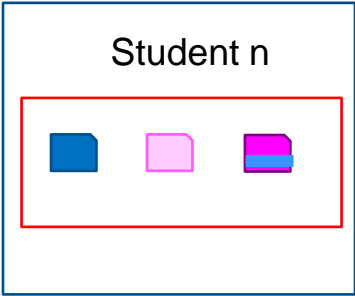


...

Who is Ziqiang?

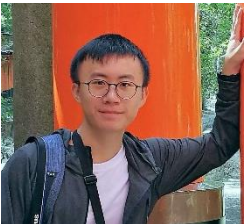
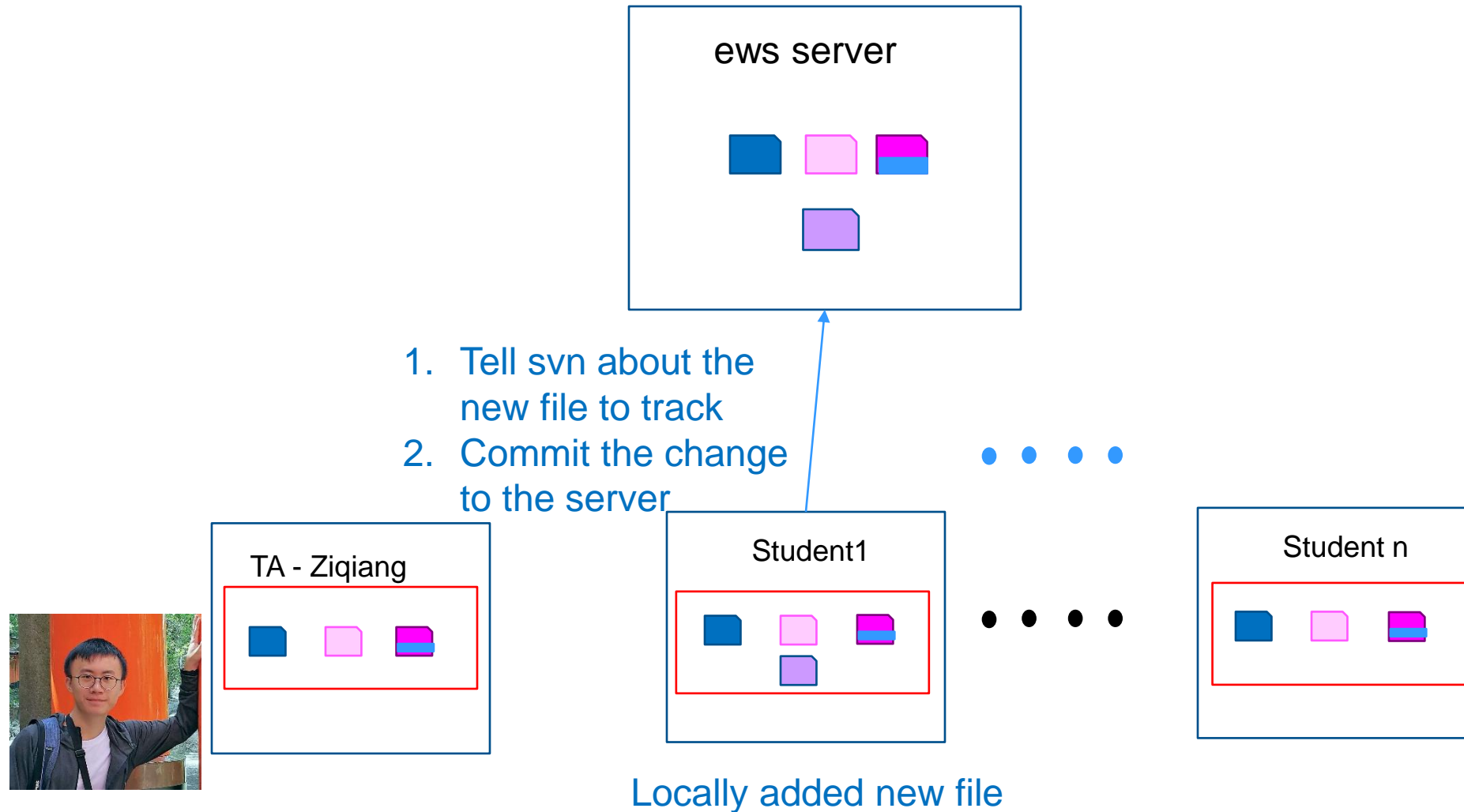


...



Locally added new file

# VCS: Quick Overview



# What is the command for telling svn/git about a new file to track?

---

- `svn add <file-name>`
- `git add <file-name>`

# Updating Local Copy From Server In SVN

- To tell svn about a new file to track:

```
svn add <name_of_new_file>
```

- To commit the change to the server (just like we did before):

```
svn commit -m"<a meaningful descriptive message>"
```

# Summary of Basic SVN Commands

- To tell svn about a new file to track:

`svn add <name_of_new_file>`

- To commit the change to the server:

`svn commit -m"<a meaningful descriptive message>"`

- To update local copy to be in sync with the server:

`svn up`

# More Commands

- **svn st:** shows the status of files in the current svn directory
- **svn rm:** removes a file from the set of tracked files (will be removed on the remote server as well)
- **svn mv:** moves a file from one directory to another (or renames if in same directory)
- **svn diff:** diff between two revisions, or diff a file to see uncommitted local changes.

# Tips

- Do **\*NOT\*** commit automatically generated files
  - For example, to ignore .class files:
    - `svn propset svn:ignore -R *.class .`
- The command “svn status” shows you the status of the files in the directory subtree you are in (and not the entire repo).
- The commands “svn commit”, “svn up”, etc take effect only w.r.t. the directory you are currently in

# SVN Directory Layout

- The general layout of an svn directory consists of 3 directories:
  - Trunk
    - most up to date current dev
  - Branches
    - releases, bug fixes, experimental
    - Do *\*not\** branch to: support a different hardware, or support a different customer
  - Tags
    - Mark a state of the code (release for e.g.)



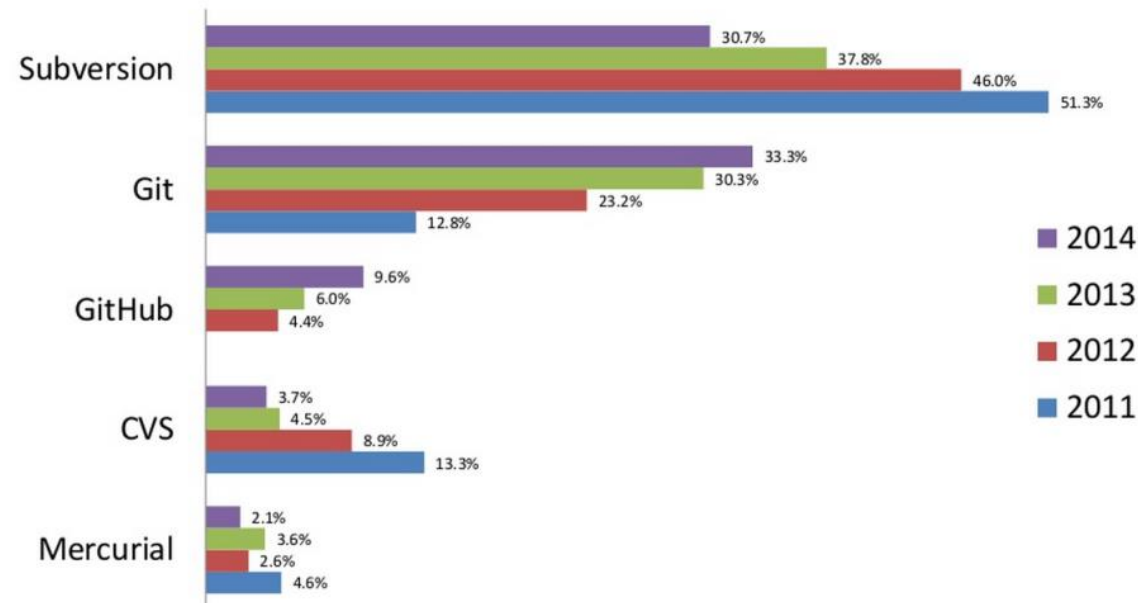
# From SVN to Git

- Starting from 2014, many projects migrate from SVN to Git

## Primary Code Management



What is the primary source code management system you typically use? (Choose one.)



# Git

---

Question: Who is the inventor of Git?

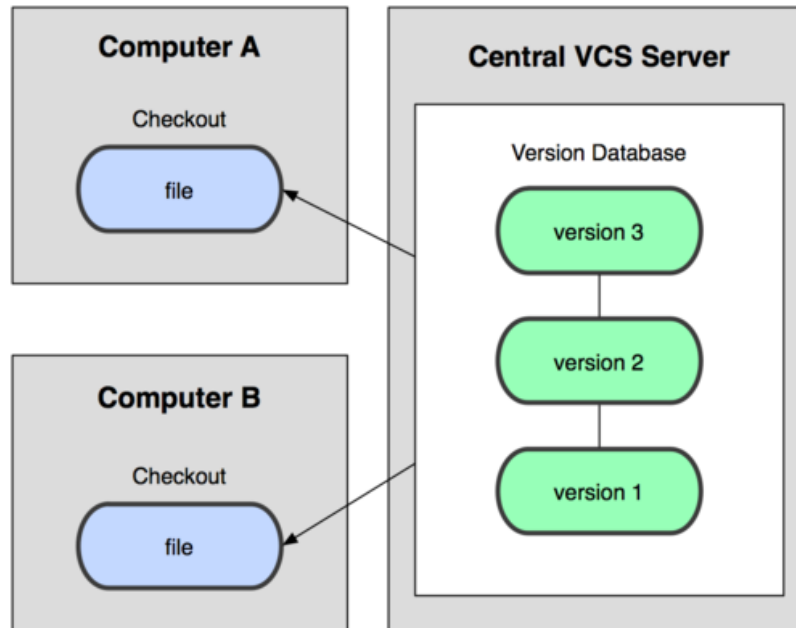
# Git History

- Came out of Linux development community
- Linus Torvalds, 2005
- Initial goals:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like Linux efficiently



# Git uses a distributed model

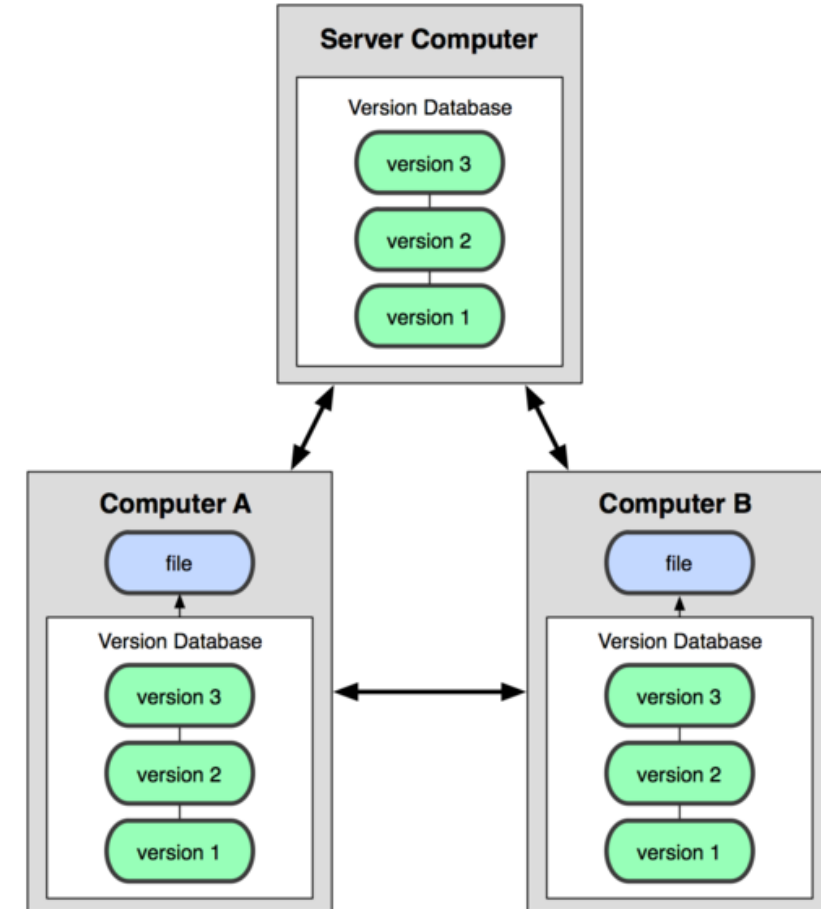
Centralized Model



(CVS, Subversion, Perforce)

Result: Many operations are local

Distributed Model



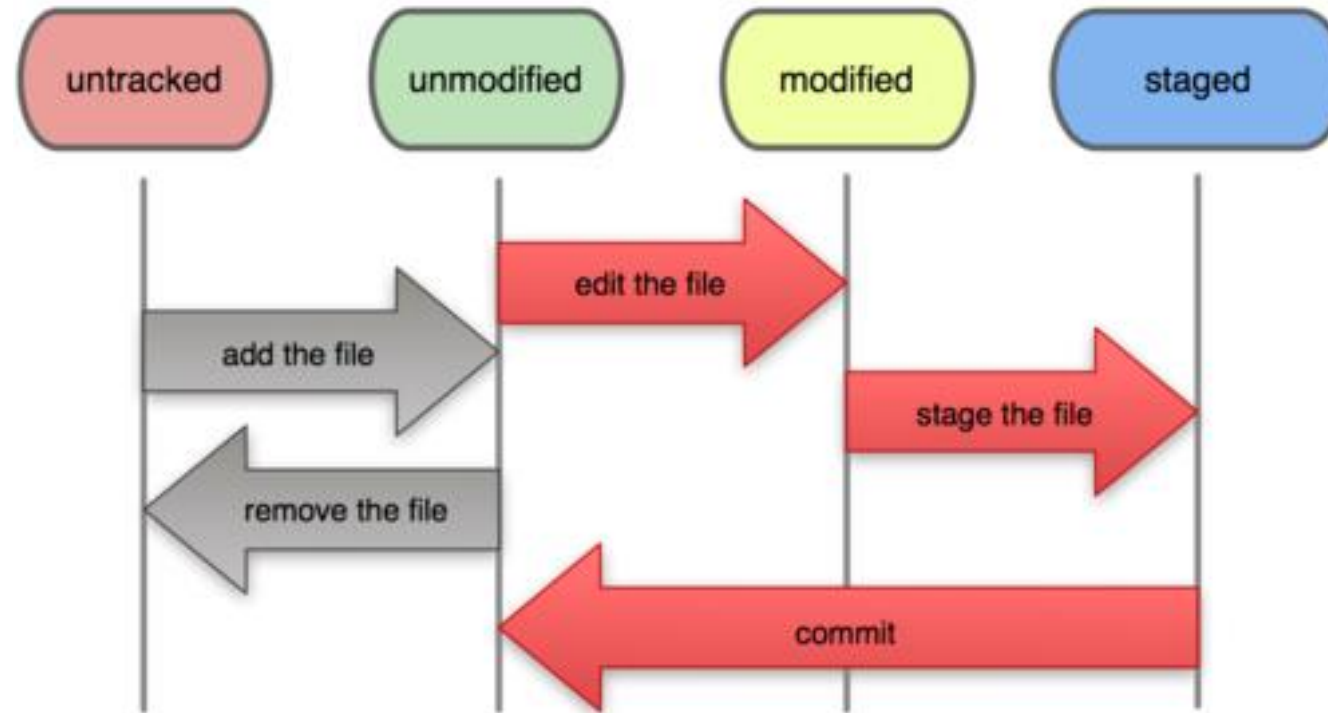
Result: Many operations are local  
(Git, Mercurial)

# Git uses checksums

- In Subversion each modification to the central repo incremented the version # of the overall repo.
- How will this numbering scheme work **when each user has their own copy of the repo**, and commits changes to their local copy of the repo before pushing to the central server?????
- Instead, Git generates a unique SHA-1 hash – 40 character string of hex digits, for every commit. Refer to commits by this ID rather than a version number. Often we only see the first 7 characters:
  - 1677b2d Edited first line of readme
  - 258efa7 Added line to readme
  - 0e52da7 Initial commit

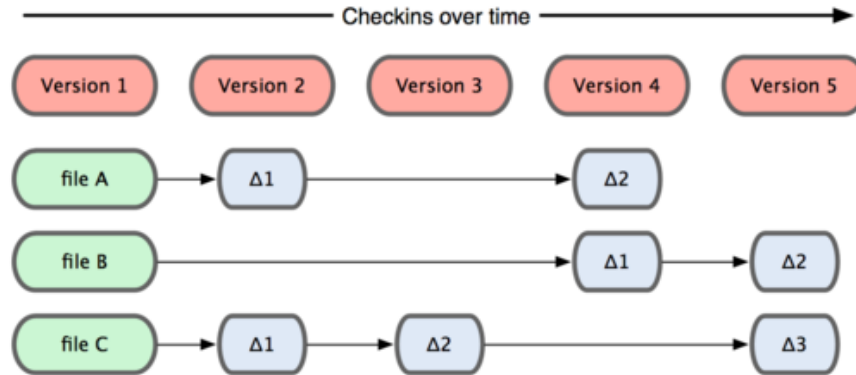
# Git file lifecycle

## File Status Lifecycle

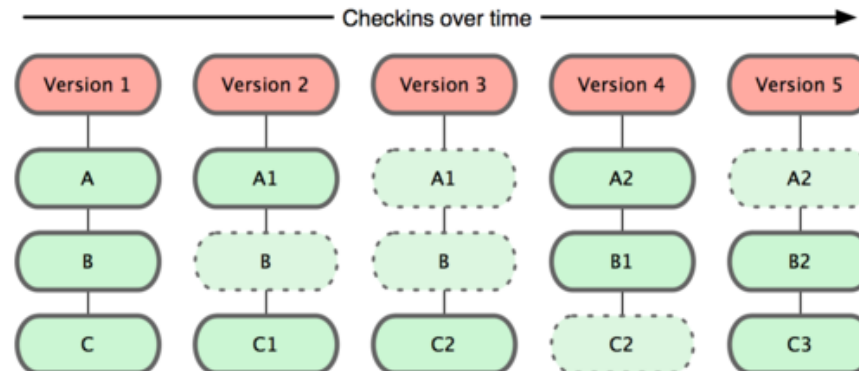


# Git takes snapshots

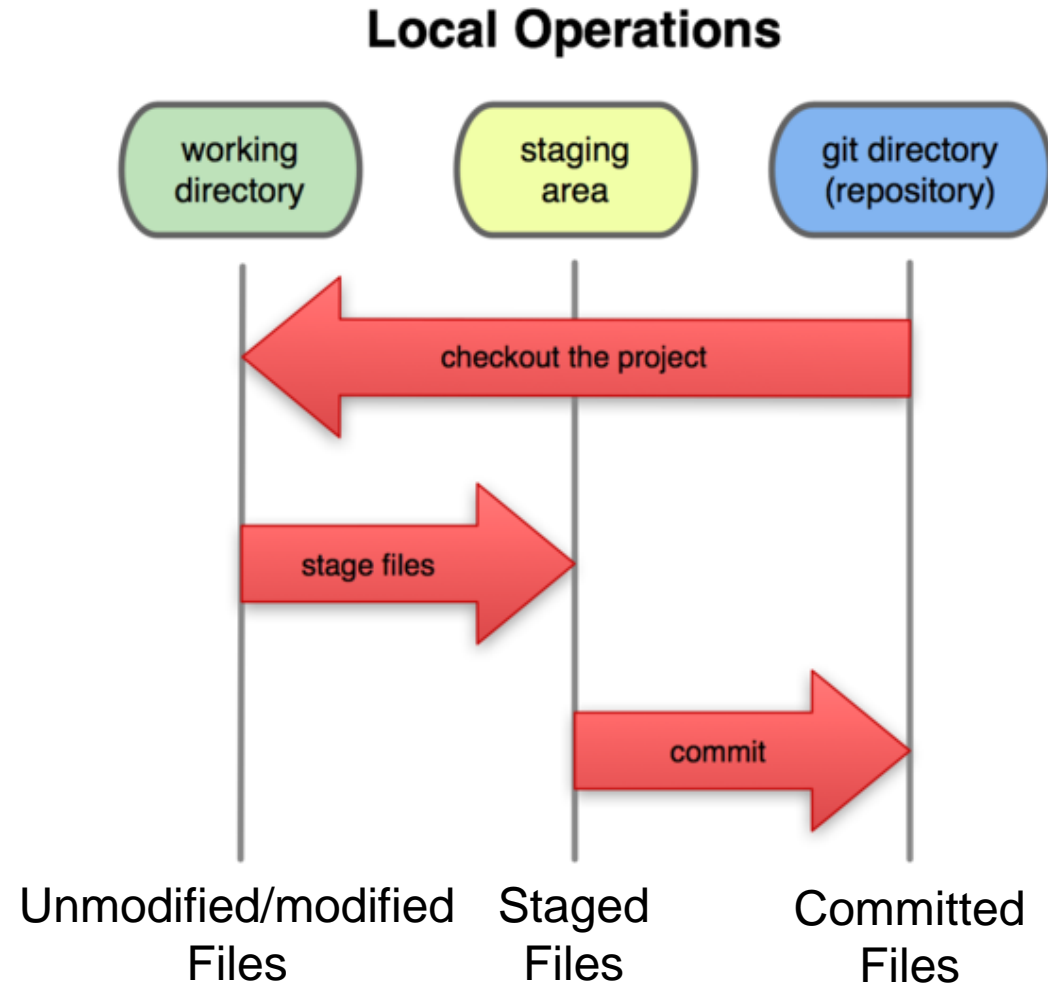
## Subversion



## Git



# A Local Git project has three areas



Note: working directory sometimes called the “working tree”, staging area sometimes called the “index”.



# When you talk about Git, do you mean GitHub?



## Aside: So what is github?

- [GitHub.com](https://github.com) is a site for online storage of Git repositories.
- Many open source projects use it, such as the [Linux kernel](https://www.kernel.org/).
- You can get free space for open source projects or you can pay for private projects.

**Question:** Do I have to use github to use Git?

**Answer:** No!

- you can use Git completely locally for your own purposes, or
- you or someone else could set up a server to share files, or
- you could share a repo with users on the same file system, such as we did for homework 9 (as long everyone has the needed file permissions).

# Role of GitHub

- “Git comes with a nice pull-request generation module, but GitHub instead decided to replace it with their own totally inferior version. As a result, I consider GitHub useless for these kinds of things. It's fine for hosting, but the pull requests and the online commit editing, are just pure garbage.”  
--Linus Torvalds
- What doesn't Linus Torvalds like about GitHub?
  - “the way you can clone a [code repository], make changes on the web, and write total crap commit messages, without GitHub in any way **making sure that the end result looks good**”



# What is pull request?

- Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.
- You will learn about pull request in today's lab!

# Basic Workflow

Basic Git workflow:

1. **Modify** files in your working directory.
  2. **Stage** files, adding snapshots of them to your staging area.
  3. Do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.
- Notes:
    - If a particular version of a file is in the **git directory**, it's considered **committed**.
    - If it's modified but has been added to the **staging area**, it is **staged**.
    - If it was **changed** since it was checked out but has not been staged, it is **modified**.

# SVN vs. Git

- SVN:
  - central repository approach – the main repository is the only “true” source, only the main repository has the complete file history
  - Users check out local copies of the current version
- Git:
  - Distributed repository approach – every checkout of the repository is a full fledged repository, complete with history
  - Greater redundancy and speed
  - Branching and merging repositories is more heavily used as a result

# Version control workflow

- Repository - contains complete history
- Check out latest version
- Hack, hack, hack, ...
- Check it in, making a new version

# What happens if two people are changing same software at same time?

---



## Version control: parallel work

- What happens if two people are changing same software at same time?
- Jack checks out V23, changes it, and checks in V24
- Jill checks out V23, changes it, and checks in ???

## Version control: parallel work

- What happens if two people are changing same software at same time?
  - One solution (locking): Impossible! First person must *lock* software before making changes. Second person must wait until lock is released.
  - Another solution (merging): The second person to check in the software must *merge* the changes. Automatic merging works most of the time (but can silently introduce bugs).

## Version control: no locking

- Locking is bad because it forces you to wait when you want to get work done
- “John has locked ELFReader.java, but he is on vacation. Can you break the lock?”

## Version control: merging risk

- Merging can be bad because sometimes it produces errors
- 1) Two people change same line. System will force the one merging the changes to perform the merge manually.
- 2) Changes don't seem to conflict, but merging causes a bug.

# Golden rule of v.c. software

- Check in quickly!
- The longer you have code checked out
  - The more you interfere with others (locking)
  - The harder it is to merge (merging)
- Break work into small steps. Finish each step the same day you start it.

# Build management

---

# Build management

- How do you build the product?
  - Which compiler? Which flags?
  - Which source files?
  - Which libraries should be linked?
- “Which” also means “which version”

# Building

- Building should be automatic
- Test build procedure - build regularly
- Need a tool
  - make - original Unix tool, 30 years old
  - ant - Java based, uses XML
  - mvn – used by Jenkins
  - Gradle – Groovy and Kotlin based
  - Hundreds more ...





# Build tool

- Knows components of system
- How to compile
- How to make final executable
- How to make debugging version
- How to delete temporary files
- How to test
- How to make manual
- ...

# Build.gradle snippet

Branch: master [android / build.gradle](#) Find file Copy path

 tobiasKaminsky daily dev 20190221 bf99c2d 4 days ago

16 contributors 

321 lines (268 sloc) 11.4 KB Raw Blame History

```
1 // Gradle build file
2 //
3 // This project was started in Eclipse and later moved to Android Studio. In the transition, both IDEs were supported.
4 // Due to this, the files layout is not the usual in new projects created with Android Studio / gradle. This file
5 // merges declarations usually split in two separates build.gradle file, one for global settings of the project in
6 // its root folder, another one for the app module in subfolder of root.
7
8 buildscript {
9     repositories {
10         google()
11         jcenter()
12         maven {
13             url 'https://oss.sonatype.org/content/repositories/snapshots/'
14         }
15         mavenCentral()
16     }
17     dependencies {
18         classpath 'com.android.tools.build:gradle:3.3.1'
19         classpath('com.dicedmelon.gradle:jacoco-android:0.1.3') {
20             exclude group: 'org.codehaus.groovy', module: 'groovy-all'
21         }
22     }
23 }
24
25 apply plugin: 'com.android.application'
26 apply plugin: 'checkstyle'
27 apply plugin: 'pmd'
```

# Daily build and smoke test

- Ways to break the build process
  - Check in bad code
  - Forget to include file in makefile
  - Move a library
- Every day (night) build the latest version of product and run simple test suite

# Summary So far

- Version Control
- Basic Commands in SVN
- Build Systems
- Gradle Example For Build Automation

# Software product

- Product = set of components/documents
  - Code
  - Test suites
  - Operation manuals (admins, end-users)
  - Requirements
  - Specifications
  - Design documentation
  - Plans/schedules

# Software products

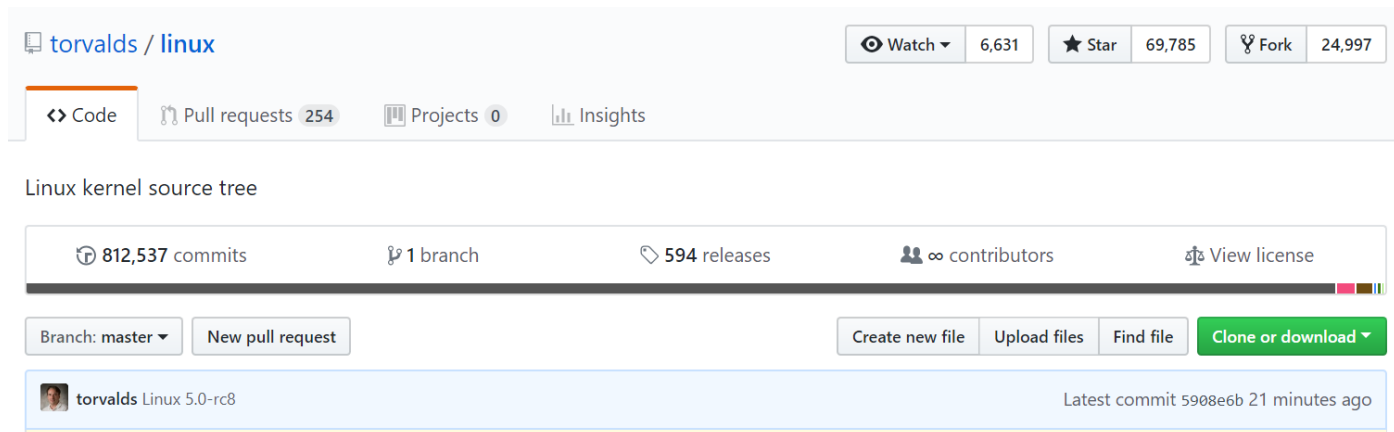
- Need to keep track of how you created a product
  - Rules for building the executable
  - Version of code
  - Version of libraries
  - The compiler
  - The operating system
- SCM tool should be able to keep track of all of these (and more)

# SCM

- Keep track of how software changes over time and be able to reproduce any version of the software
- Control how software changes; make sure needed changes have been made and no improper changes have been made

# Many versions

- Sequence of versions during development
  - Prototypes, daily (weekly) builds
  - Alpha/beta release
  - Final release
- Different released versions
  - Linux - many versions
  - 5ESS - tailored for each customer





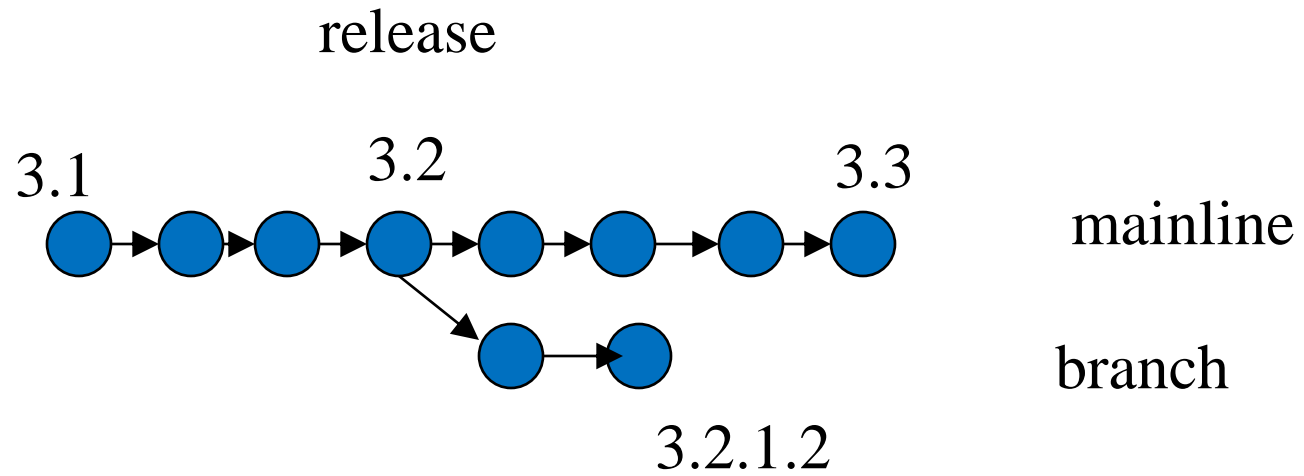
# Non-code resources

- Test suite for version 6.4.3 does not work for other versions. It is almost like the test suites for 6.4.2 and 6.4.4.
- Manual for version 6.4.3 is slightly different than for other versions.
- Design ...
- Use cases ...

# Change request

- We decided to implement a change
  - Has it been implemented fully? (tests, code, manuals, documentation)
  - What parts of the system were affected by that change?
- I look at a program/document
  - Why is it like this?
  - When was it written, and by whom?
- Tracing both ways: change control + version control linked through IDs

# Versions



Note: You will see more of this in the lab this week!

# Branches

- Traditional advice for old version control
  - Avoid (long-lived) branches if possible
- Modern version control (Git, HG...)
  - Encourage use of (short-lived) branches
- Good reasons to branch
  - Fixing bugs in customer version
  - Experimental version
  - Political fights

# Bad reasons to branch

- Support different hardware platform
  - Make subclasses / use conditional compilation / make portability library
- Support different customer
  - Separate unchanged code and changed code
  - Unchanged code goes in a library
  - Make subclasses / use conditional compilation for changed code

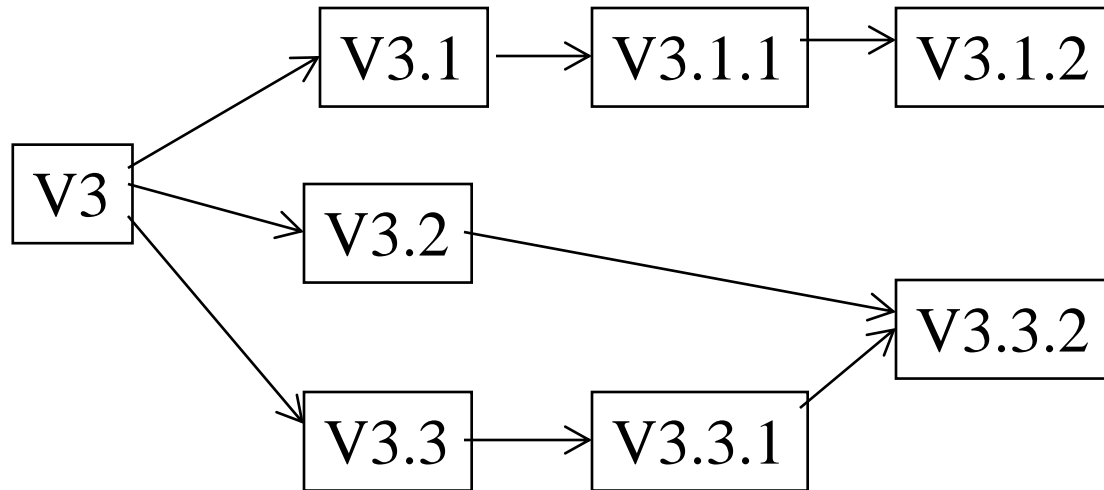
# SCM according to the SEI

- A discipline for controlling the evolution of software systems
- Has many aspects
  - Identification
  - Control
  - Status accounting
  - Audit and review

# Identification

- What are the configuration items? (I.e., what is under configuration management?)
- How do you name them?
- What is the relationship between items?
  - Versions
  - Baseline
  - Release

# Versions





# Baseline

- Baseline is a software configuration item that has been reviewed and agreed upon, and that can be changed only through formal change control procedures
- Intermediate versions that haven't been reviewed are SCIs but not baselines

# Release

- Release is a software configuration item that the developers give to other people
- Release should be a baseline

# Control

- Who is allowed to read/write configuration items?
- How do you know if changes are allowed/correct?

# Status accounting

- Reporting the status of components and change requests
  - Which components have changed this week?
  - Which components did Bob change?
  - Which components have the most changes?
  - Which change requests are more than a month old and of priority 3 or greater?

# Audit and review

- How do we know that the build script is OK?
- How do we know that only authorized people can change the database interface?
- Can we actually run the version from September 8, 2015?

# Change control

- Change request/engineering change order
  - New feature
  - Bug report
- Change control authority - decides which changes should be carried out
- Should link code changes to change requests

# Example: Bugzilla

- Originally developed for Mozilla  
<http://bugzilla.mozilla.org>  
[https://developer.mozilla.org/en/Bug\\_writing\\_guidelines](https://developer.mozilla.org/en/Bug_writing_guidelines)
- Bugzilla is a database for bugs. It lets people report bugs and assigns these bugs to the appropriate developers. Developers can use Bugzilla to keep a to-do list as well as to prioritize, schedule and track dependencies

# Bugzilla

- Each bug report has ID, name of reporter, description of bug, component, developer, dependency, attachments
- Status: unconfirmed, new, resolved, verified, closed
- Severity: blocker, critical, major, normal, minor, trivial, enhancement



# Resolving a bug

FIXED:

INVALID: not a bug, or not a bug in Mozilla  
bug that will never be fixed  
fixed now

WONTFIX : a

LATER: a bug that won't be

REMIND: maybe now, maybe later

DUPLICATE

WORKSFORME

# Topics to ponder

- What is the dynamics of people using a change control system?
  - Developers
  - Testers
  - Users
  - Managers
- What about security issues in Bugzilla?

# To make SCM work requires

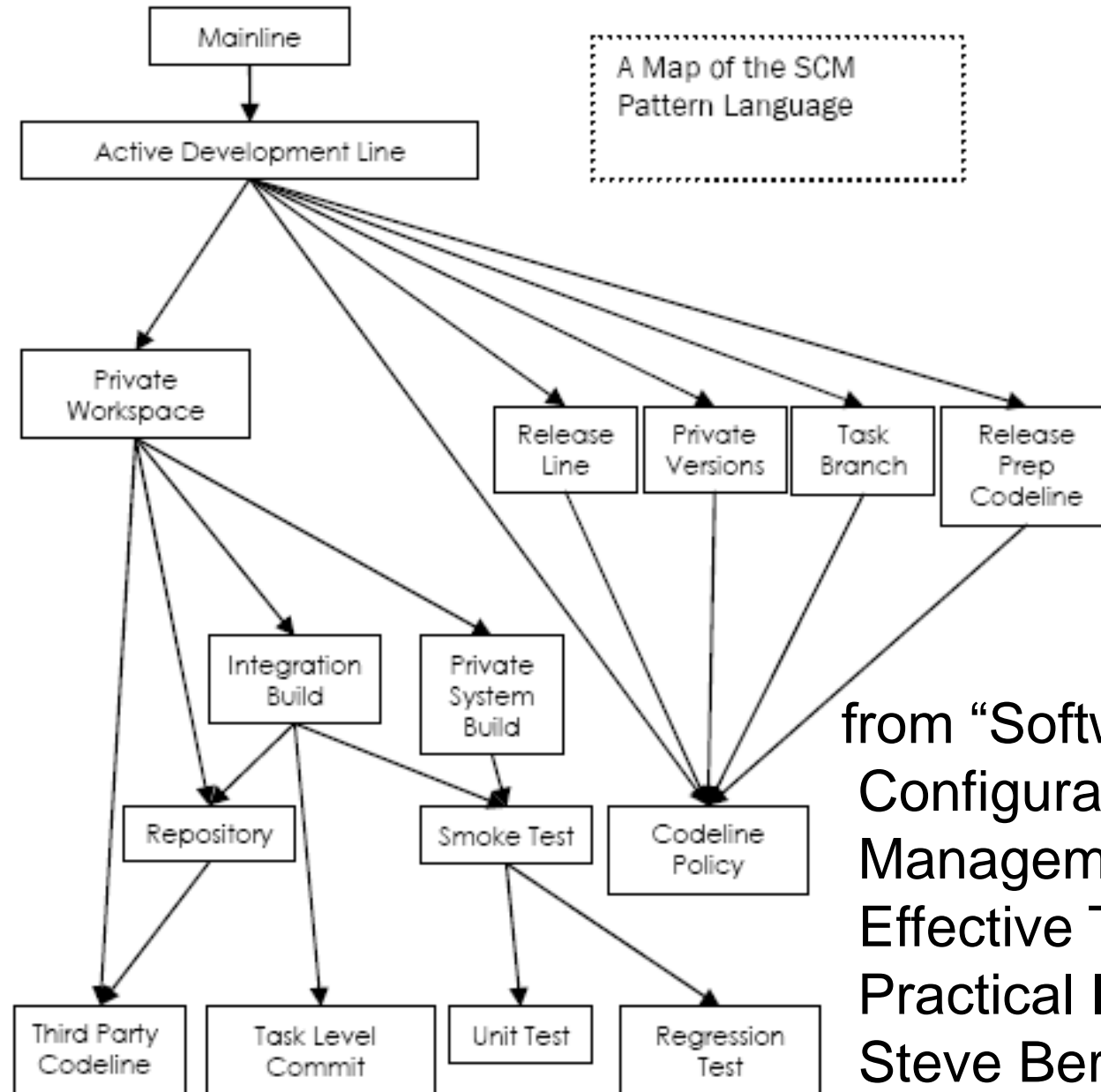
- Bureaucracy
- Discipline/training
- Tools
  - Version control – cvs, subversion, git...
  - Change control – bugzilla, mantis, jira...
  - Building – make, ant, mvn...
  - Releasing – Maven Central and Nexus...

# SCM Manager

- Complex tools need expert to manage them
- SCM expert will
  - Maintain tools
  - Maintain configuration files, make branches
  - Do the merging
  - Create policies on version control, change control

# Alternatives

- Toolsmith supports SCM tools
- Architect defines configuration files
- Developers merge their code back into mainline
- Managers and technical leads define policies for version control and change control



from “Software Configuration Management Patterns: Effective Teamwork, Practical Integration” by Steve Berczuk with Brad Appleton

# Various tests

- Smoke test
  - Ensure that the system still runs after you make a change
- Unit test
  - Ensure that a module is not broken after you make a change
- Regression test
  - Ensure that existing code doesn't get worse as you make other improvements

# Developer issues

- Private Workspace
  - Prevent integration issues from distracting you, and from your changes causing others problems, by developing in a Private Workspace.
- Private System Build
  - Avoid breaking the build by doing a Private System Build before committing changes to the Repository



# Codeline Policy

- Active Development Line
- Release Line
  - Holds bug fixes for a release
- Private Versions
- Task Branch
  - Hide a disruptive task from the rest of the team
- Release Prep Codeline

# Summary of SCM

- Four aspects
  - Change control
  - Version control
  - Building
  - Releasing
- Supported by tools
- Requires expertise and oversight
- More important on large projects