


CS 213

SOFTWARE

METHODOLOGY

Lily Chang, Fall 2024





Lecture Note #1

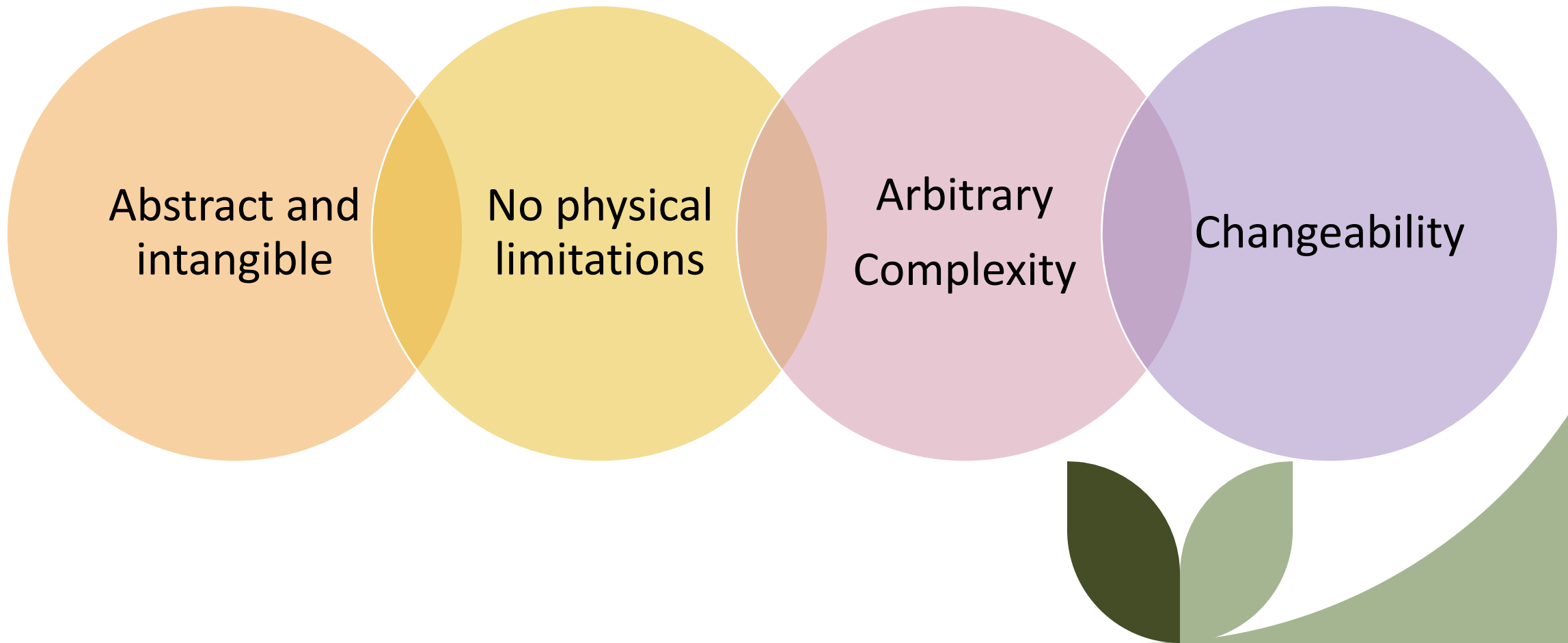
Course Introduction



What is Software?

- Computer programs
 - Data and Control
- Running environment
 - Hardware, software and the people involved
- Associated documentations and Configurations
 - Deployment and operation

Essence of Software



Arbitrary Complexity

■ Human Factor

- ❖ Involve a group of people with different perspectives from different disciplines and application domains
- ❖ For example, health care, business, engineering, etc.

■ Distributed software components

- ❖ Autonomous
- ❖ Service-oriented cloud computing
- ❖ Internet of Things (IoT)
- ❖ Cyber Physical Systems (CPS)



What do you see?



Changeability

- Software must evolve to remain useful
 - ✓ Discovered errors must be fixed
 - ✓ Software is cheap and easy to change as opposed to hardware components
 - ✓ The time between technological changes is often shorter than the duration of the project
 - ✓ Requirements change!!!



Software Methodology

- Methods for developing **quality software**
- Good practices for software development



Maintainability

Dependability
and Security

Efficiency

Usability and
Acceptability

Essential Lifecycle Activities in Software Development

Requirement
Analysis

Design

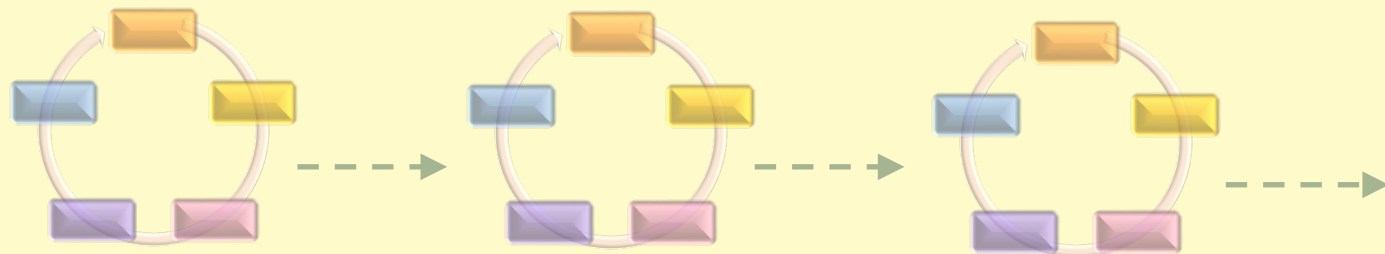
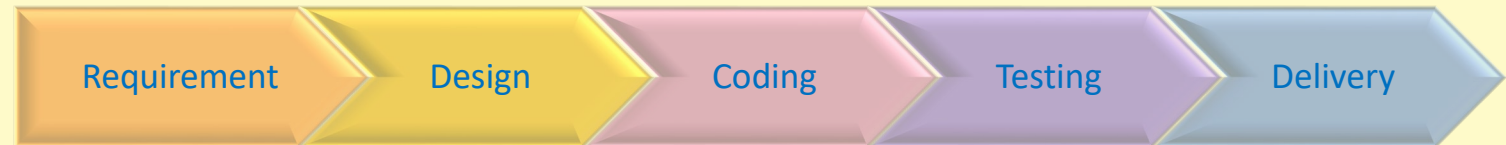
Implementation

Verification and
Validation

Delivery
Maintenance
Evolution

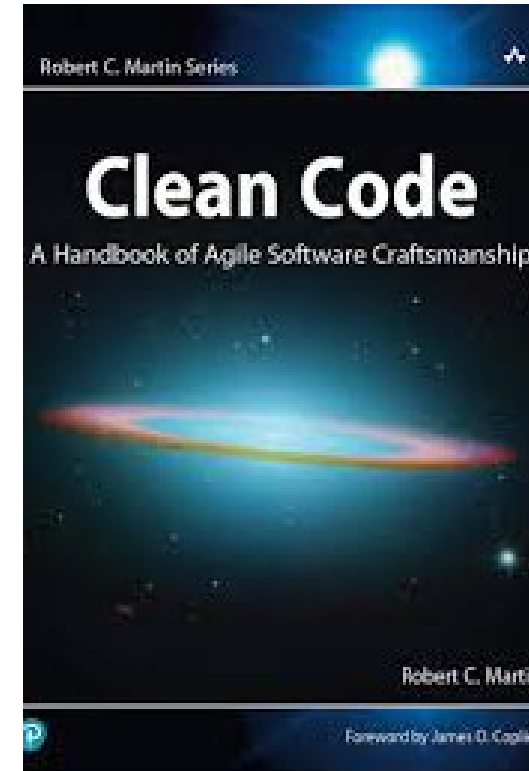
Software Process Models

- A Software Process Model is an abstract representation of a software production process
- Generic process models (paradigms) are used to manage software production
 - Waterfall model
 - Sequential process
 - Plan-driven
 - Incremental or evolutionary
 - Iterative process
 - Feature-driven
 - Spiral Model
 - Agile



Good Practices for Software Development

- Software engineering is not just coding, but everything involved in "building" quality software
 - ✓ Requirements, design, implementation (coding), testing, deployment and operation
- Clean code, a set of fundamental principles in coding; for example,
 - ✓ Code should be understood by a developer other than its original author (software maintenance)
 - ✓ Easier to develop test cases for testing (software testing)
 - ✓ Minimal dependencies (loosely coupled) and provides a clear and minimal API (minimize the impact of software change)
 - ✓ Each function/method should only perform one task (easier to reuse, test and debug)



Coding Standard enforced this class

■ Readability

- ✓ Java class comments and method comments
- ✓ Code indentations and lineup
- ✓ Descriptive names for classes, methods, constants and variables
- ✓ No MAGIC numbers!
 - A magic number is a numeric value that remains unchanged during program execution, and it is used directly in code without a name

■ Modularity – reuse and testing

- ✓ Do only ONE THING in a method, this will keep the method short
- ✓ A long method is an indication that the method is doing too much!
- ✓ Test design will be straightforward



Agile Manifesto



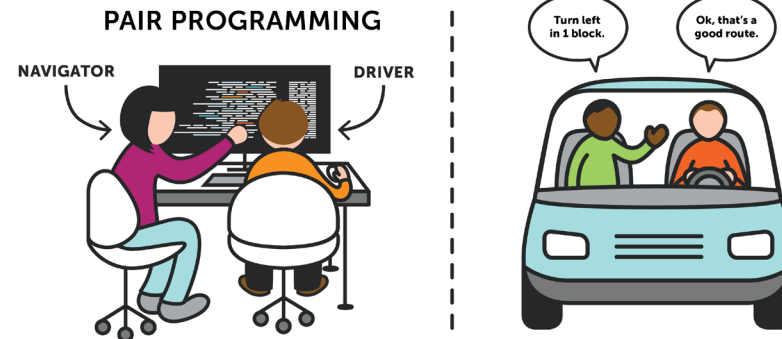
- Adapt software changes better
- Frequent delivery (continuous integration and delivery)
- Involve customers in the development process
- Popular agile methods
 - XP (eXtreme Programming)
 - Scrum framework for software production management

<https://www.agilealliance.org/agile101/subway-map-to-agile-practices/>



Pair Programming in eXtreme Programming (XP)

- Two developers share a single workstation (one screen, keyboard and mouse among the pair)
- The developer at the keyboard is the “driver”, the other one actively involved in the programming task but focuses more on overall direction, is the “navigator”
- It is expected that the developers swap roles every few minutes or so.



Pair Programming – Expected Benefits

Increased code quality: “programming out loud” leads to clearer articulation of the complexities and hidden details in coding tasks, reducing the risk of error or going down blind alleys

Better diffusion of knowledge among the team; in particular, when a developer unfamiliar with a component is pairing with one who knows it much better

Better transfer of skills, as junior developers pick up micro-techniques or broader skills from more experienced team members

Large **reduction in coordination efforts**, since there are $N/2$ pairs to coordinate instead of N individual developers

Improved **resiliency of a pair to interruptions**, compared to an individual developer: when one member of the pair must attend to an external prompt, the other can remain focused on the task and can assist in regaining focus afterward



Pair Programming – Common Pitfalls

Both programmers **must be actively engaging** with the task throughout a paired session, otherwise no benefit can be expected

A simplistic but often raised objection is that pairing “**doubles costs**”; this is the worst-case outcome of poorly applied pairing

At least the driver, and possibly both programmers, are expected to keep up a running commentary; pair programming is also “**programming out loud**” – if the driver is silent, the navigator should intervene

Pair programming cannot be fruitfully forced upon people, especially if **relationship issues**, including the most mundane (such as personal hygiene), are getting in the way; solve these first!



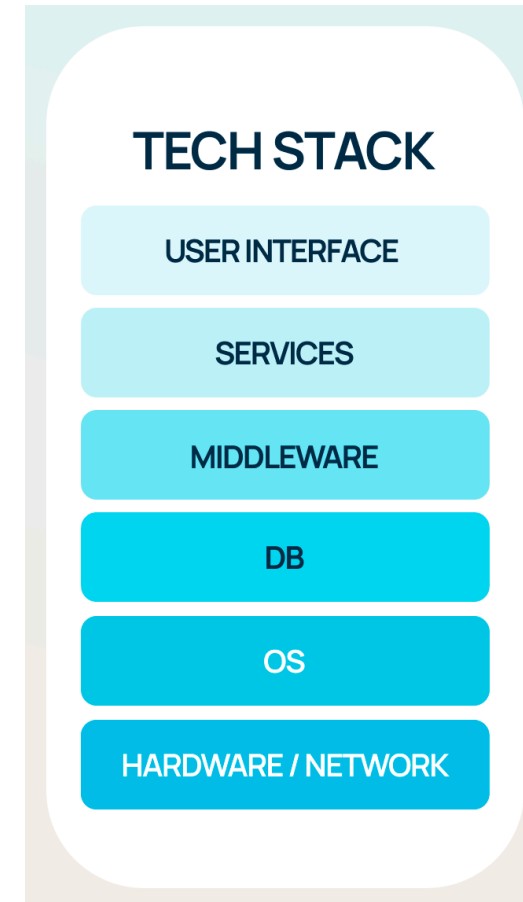
Object-Oriented Paradigm

- Why OO approach?
 - ✓ Software changes are unavoidable after delivery
 - ✓ OO approach is to better manage the future changes
- Object-oriented analysis, design and programming are related but distinct
 - ✓ **OOA** is concerned with developing an object model of the application/problem domain (What)
 - ✓ **OOD** is concerned with developing an object-oriented system model to implement requirements (solutions domain) (How)
 - ✓ **OOP** is concerned with realizing an OOD using an OO programming language such as Java or C++



Software Design

- Architectural Design
 - ✓ Software Architecture
 - Presentation logic, business logic, data access logic, data storage (system level)
 - Design patterns (component level)
 - UML - Class Diagram
- Frontend – user interactions
 - ✓ User Experience Design
 - ✓ User Interface Design
- Backend – processes and data management
 - ✓ Business logic
 - ✓ Data design
 - ✓ Data access design

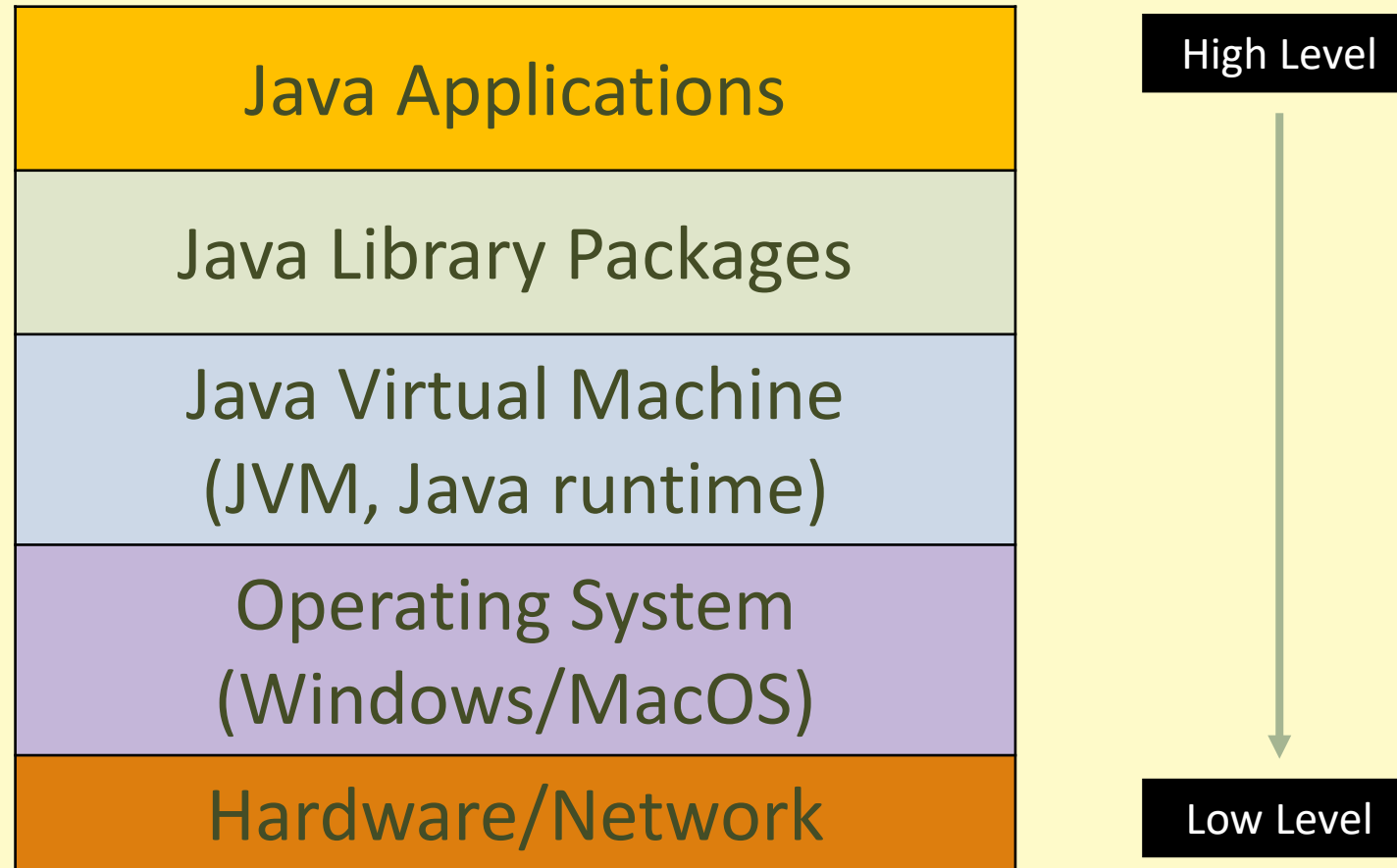


Java language

- A popular language - <https://www.tiobe.com/tiobe-index/>
- The prerequisite of this course is CS 112, Data Structures
 - Java language is used in CS 111 and CS 112
- Java is a pure OO programming language
 - Syntax derived from C language
 - Class hierarchy designed to support OOP
- Many open-source Java library classes (APIs) available
 - Software reuse reduces the efforts in software development



How Java applications run on computers



Integrated Development Environments

- IntelliJ community edition with JavaFX project support
- JavaFX – Java packages to support GUI programming
- Android Studio Koala
- iLab machines remote access: <https://weblogin.cs.rutgers.edu/>, use your NetID and CS password, must activate your CS account here: <https://services.cs.rutgers.edu/accounts/>
- iLab machines status report: <https://report.cs.rutgers.edu/nagiosnotes/iLab-machines.html>
- Version Control: GitHub



Remote Environment

Remote
Repository



Local
Repository



Staging Area



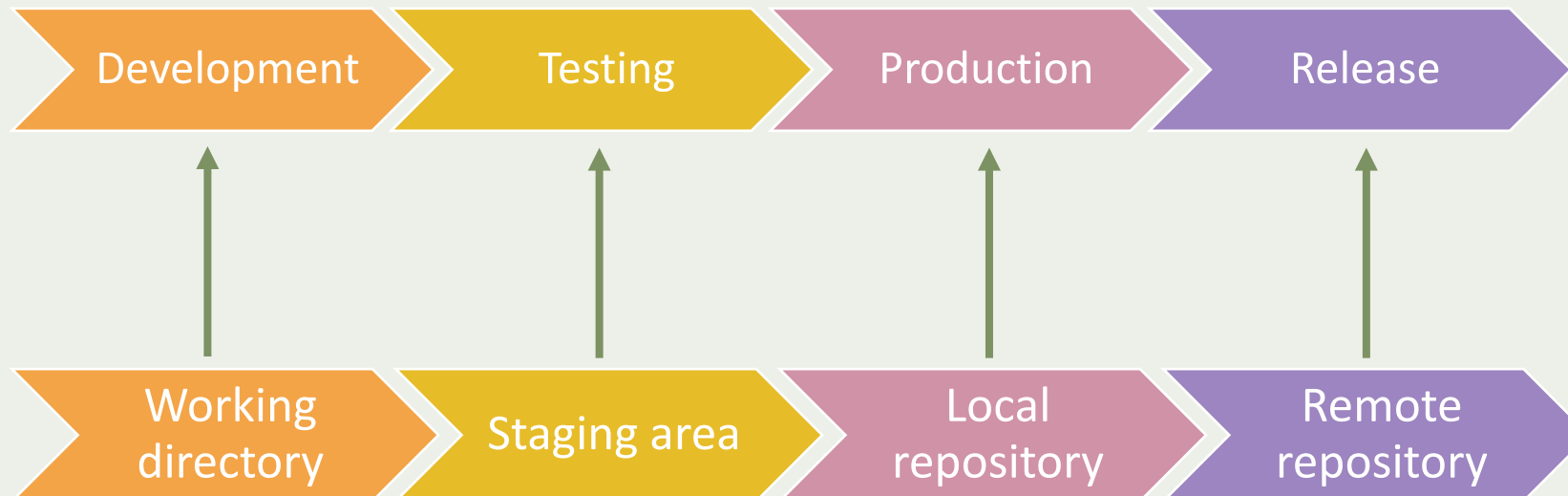
Working
Directory

Local Environment

The levels in Git

- Starting at the bottom is the working directory where content is created, edited, deleted
- The staging area serves as a holding area to accumulate and stage changes from the working directory before they are committed into the next level—the local repository
- The local repository is the actual source repository where content that Git manages is stored. Once content is committed to the local repository, it becomes a version in the repository and can be retrieved later.
- The combination of the working directory, staging area, and local repository make up your local environment, where users create and update content and get it in the form they want before making it available or visible to others, in the remote repository
- The remote repository is a separate Git repository intended to collect and host content pushed to it from one or more local repositories. its main purpose is to be a place to share and access content from multiple users.

Continuous Integration and Continuous Delivery (DevOp vs. Git Levels)

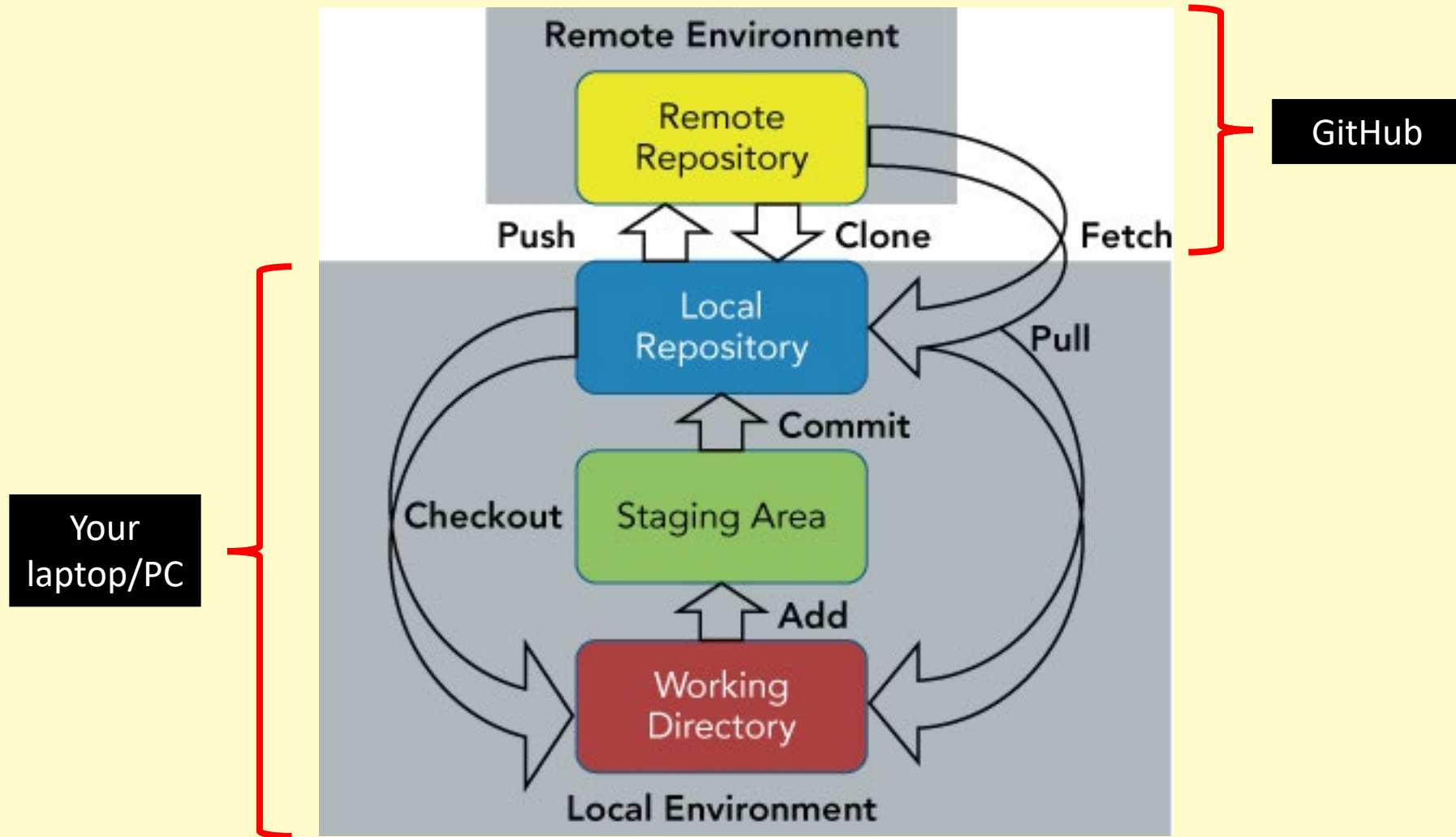


Git Core Commands

From	To	Command	Notes
Working Directory	Staging Area	Add	Stages local changes
Staging Area	Local Repository	Commit	Commits only content in staging area
Local Repository	Remote Repository	Push	Syncs content at time of push
Local Repository	Working Directory	Checkout	Switches current branch
Remote Repository	Local Environment	Clone	Creates local repository and working directory
Remote Repository	Local Repository	Fetch	Updates references for remote branches
Remote Repository	Local Repository and Working Directory	Pull	Fetches and merges to local branch and working directory

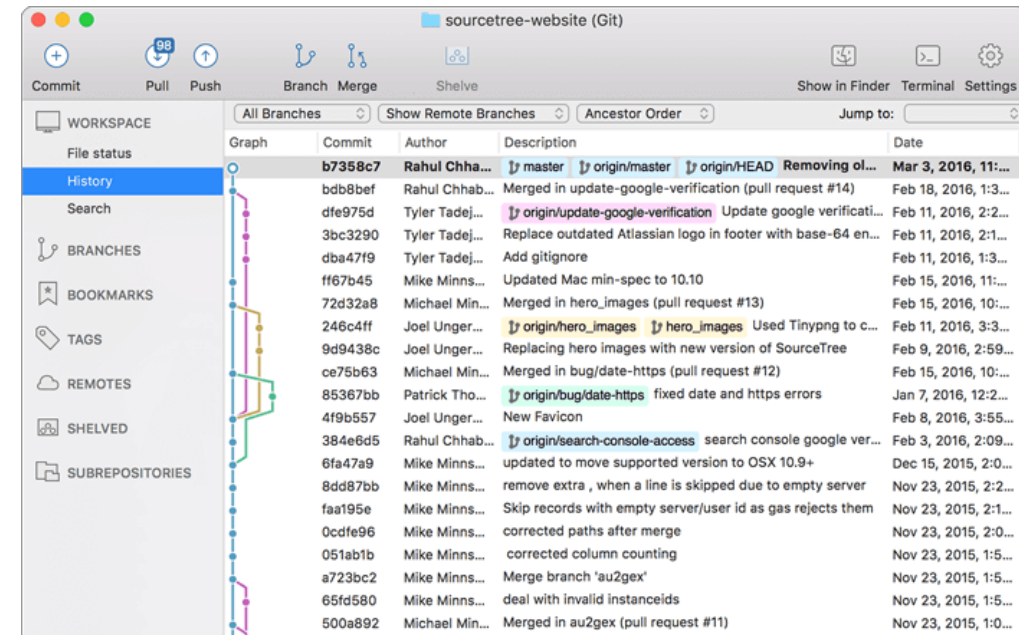


Git Commands



Git GUI Clients

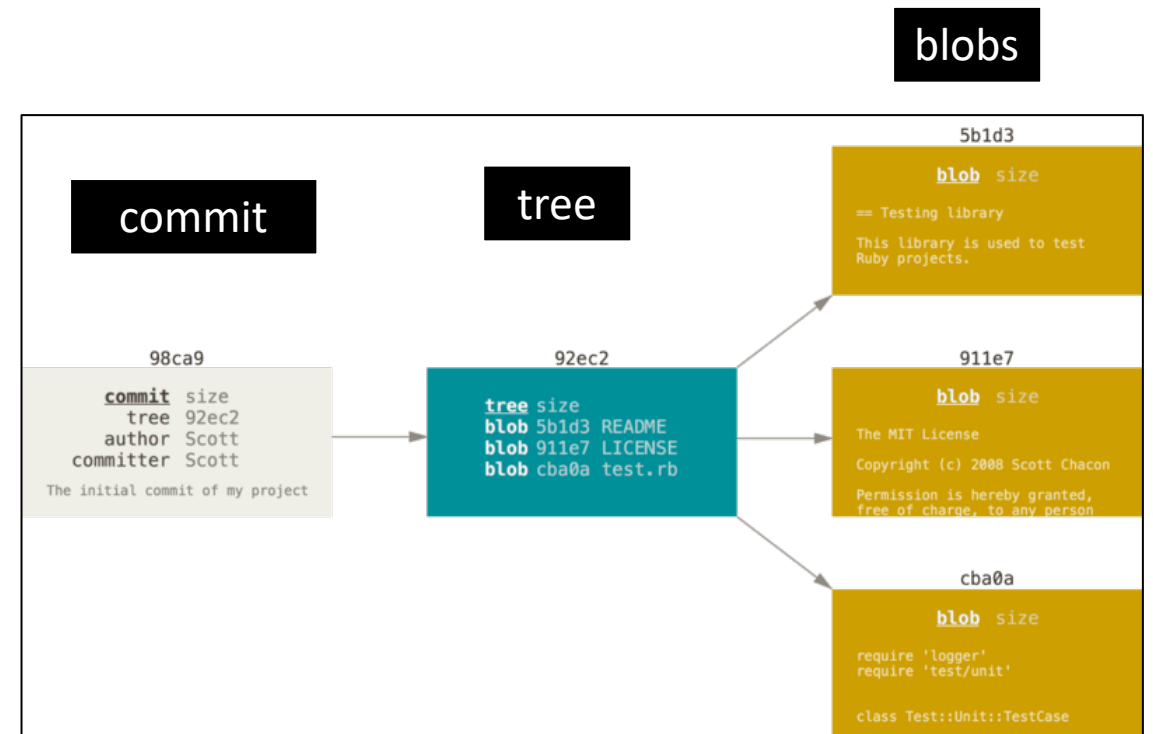
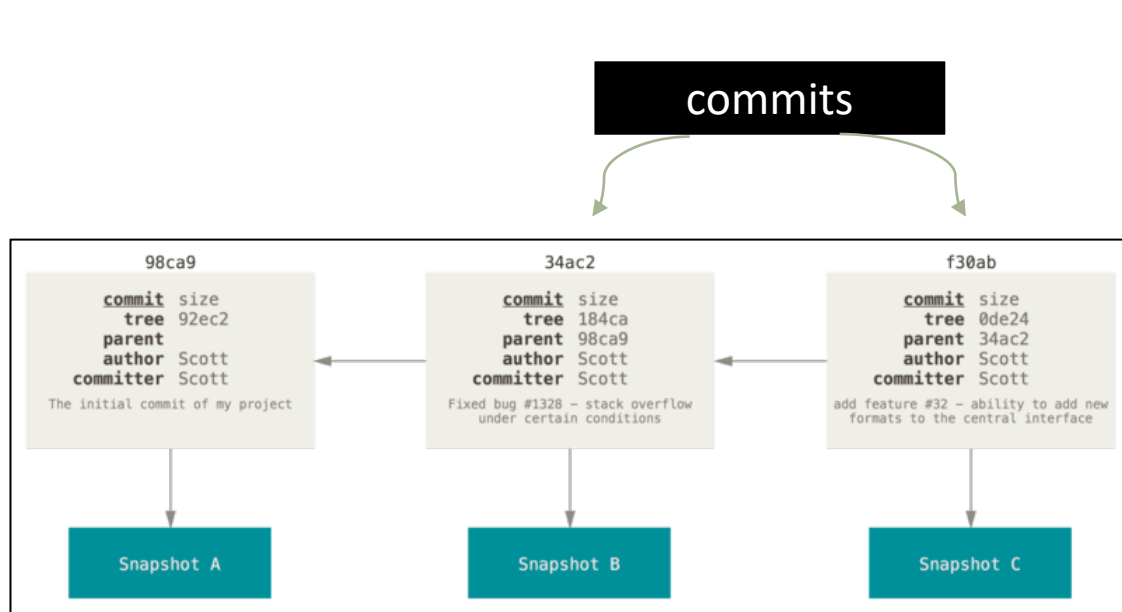
- Provide GUI interfaces for working with repositories and may support GUI-based conventions such as drag-and-drop to move content between levels.
- They often provide graphical tools for labor-intensive operations such as merging.
- Examples include SourceTree, SmartGit, TortoiseGit, and Git Extensions.
- IDEs nowadays also integrate version control systems into the menu



Git – Branching

- Branching means you diverge from the main line of development and continue to do work without messing with that main line.
- In many VCS tools, this is a somewhat expensive process, often requiring you to create a new copy of your source code directory
- The way Git branches is incredibly lightweight, making branching operations nearly instantaneous, and switching back and forth between branches generally just as fast.
- A branch in Git is a simple file that contains the 40-character **SHA-1 checksum** of the commit it points to
 - ✓ SHA (Secure Hash Algorithm) – generate a fix-length message digest from the input (plaintext)
 - ✓ Branches are cheap to create and destroy.
 - ✓ Creating a new branch is as quick and simple as writing 41 bytes to a file (40 characters and a newline)



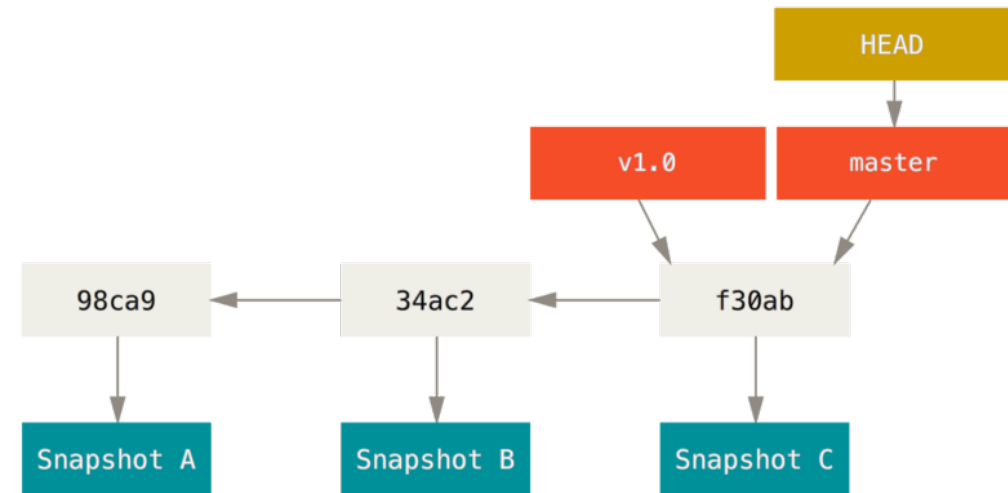


GIT – Branching

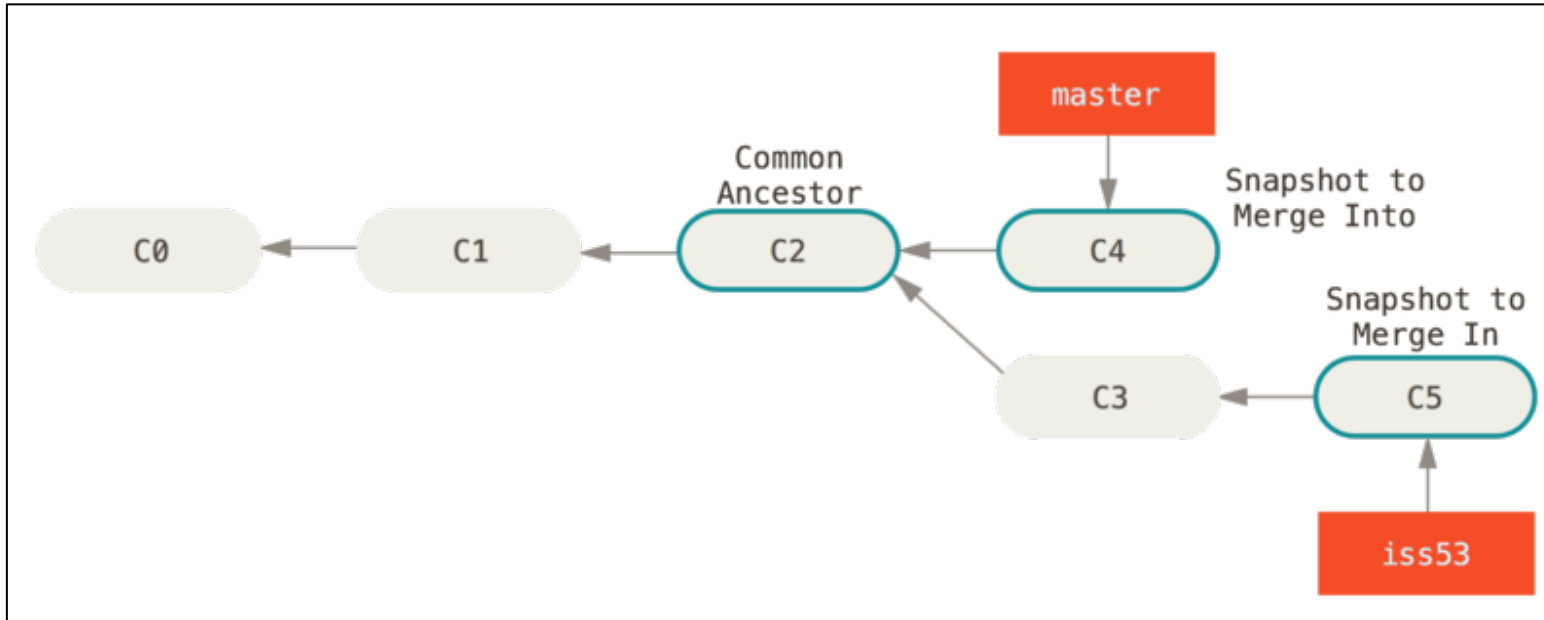
- A commit and its tree (right)
 - Git creates 5 objects for a commit with 3 files
- Commits and their parent (left) – subsequent commits point to their direct parent commit

Git – Branching

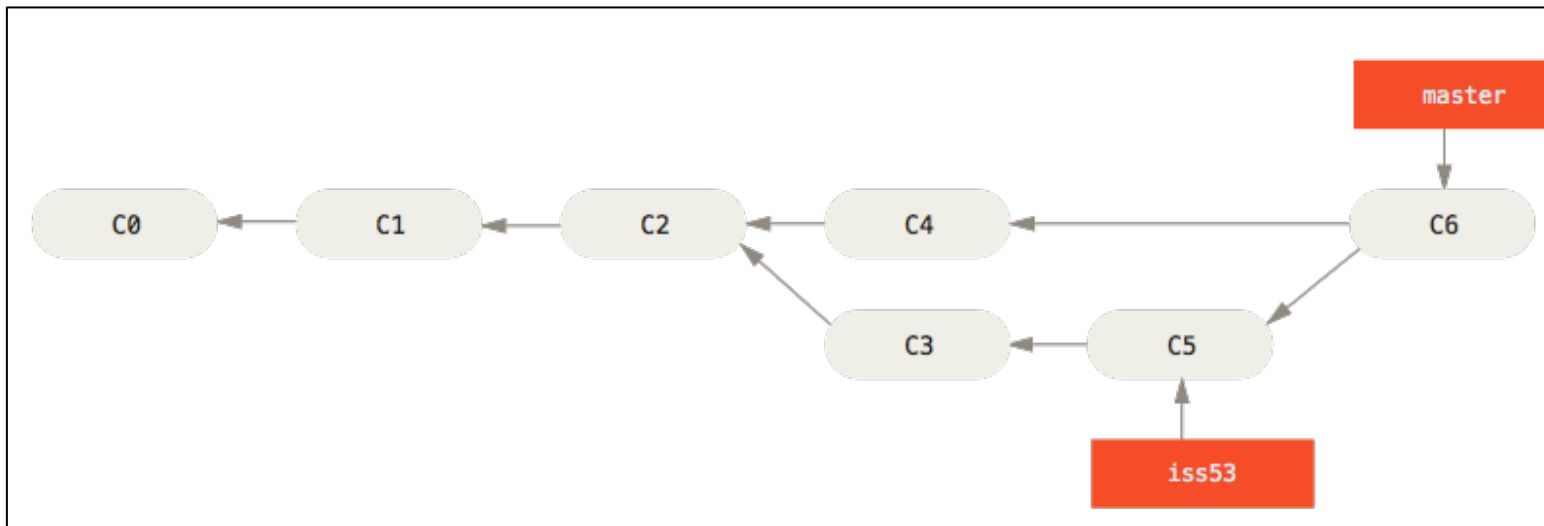
- A branch in Git is simply a lightweight movable pointer to one of these commits.
- The default branch name in Git is master. As you start making commits, you're given a master branch that points to the last commit you made.
- The “master” branch in Git is not a special branch. It is exactly like any other branch.
- HEAD is a pointer to the local branch you're currently on



A branch and its commit history

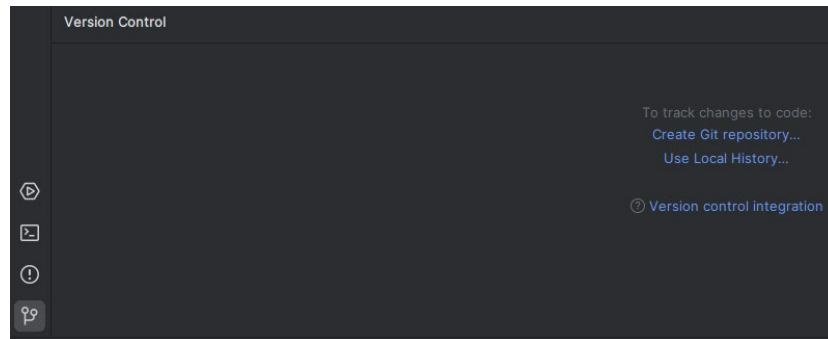


merging

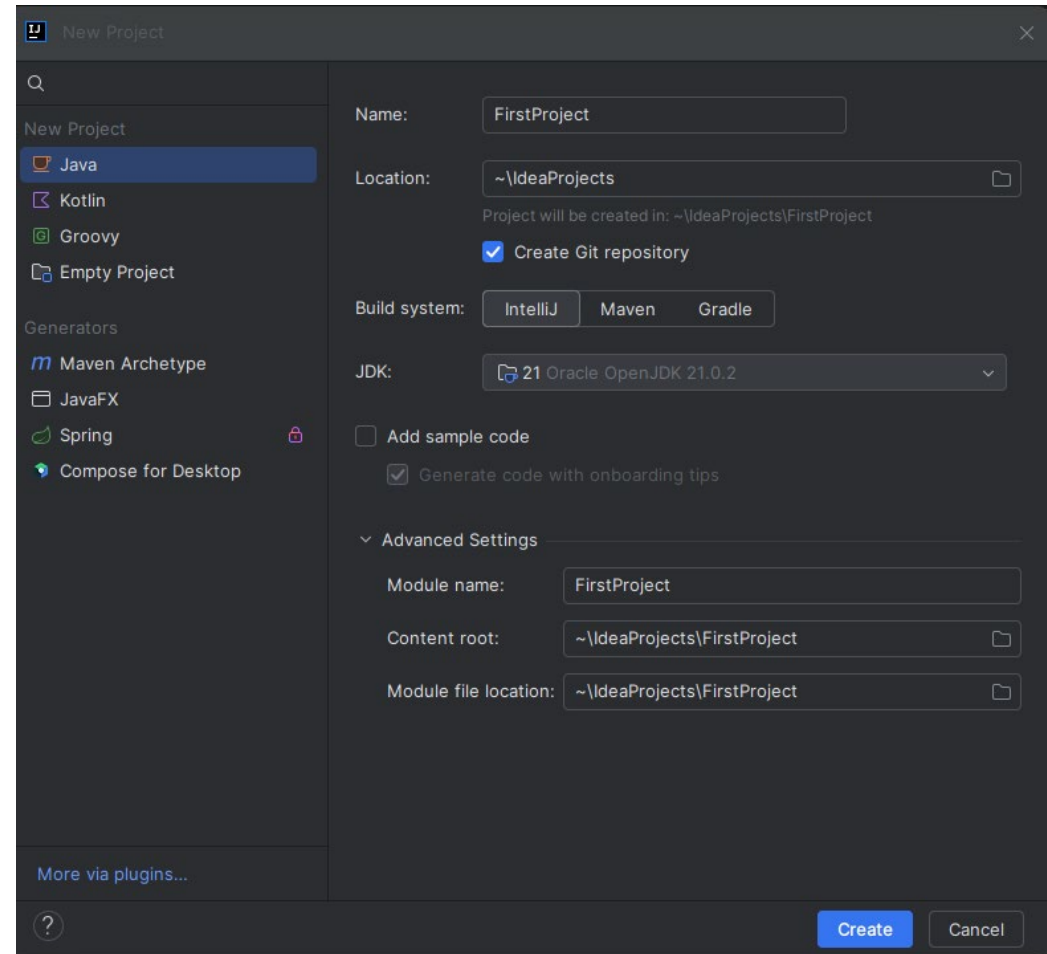
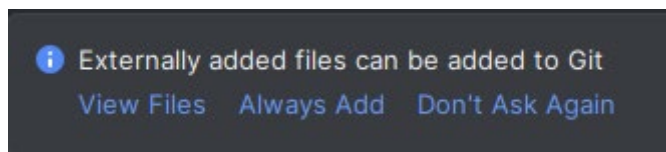


Using IntelliJ with GitHub

- Check “create Git repository” when creating a Java project
 - if you didn’t check, then

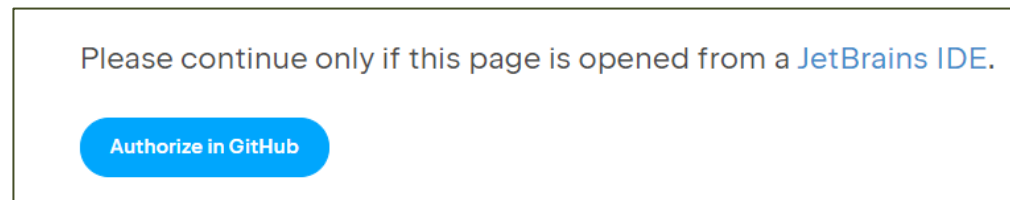
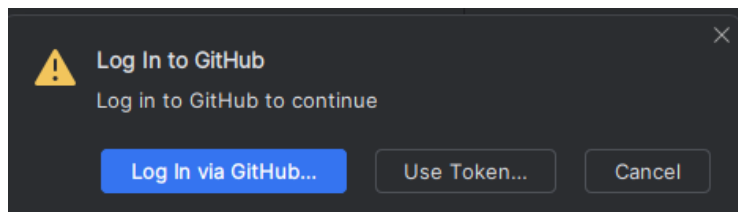
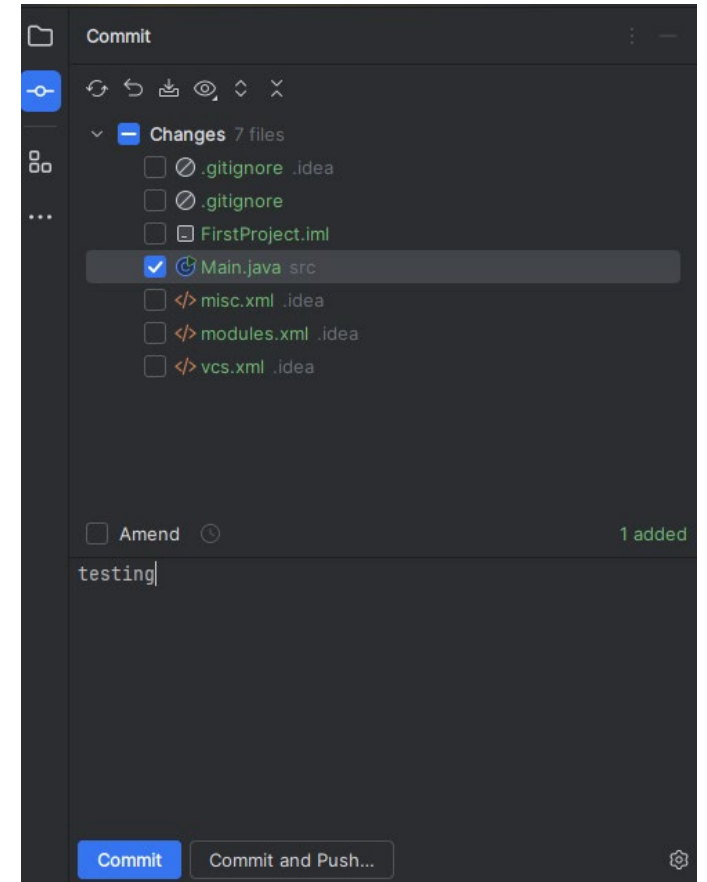
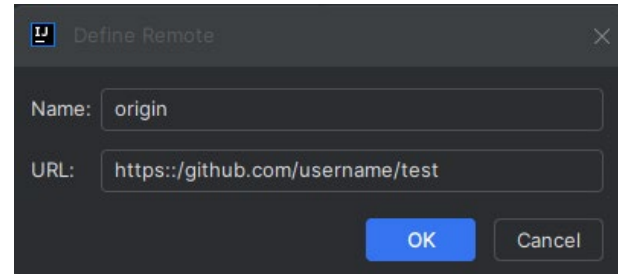
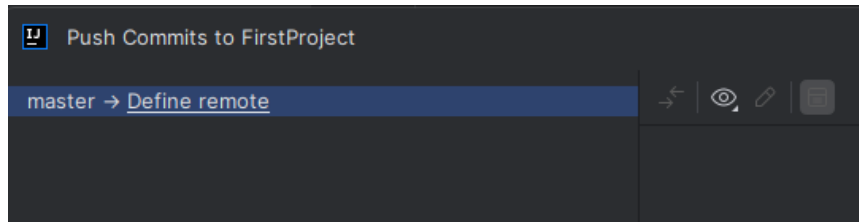


- Select Always Add, which will always add the new Java source files to the staging area



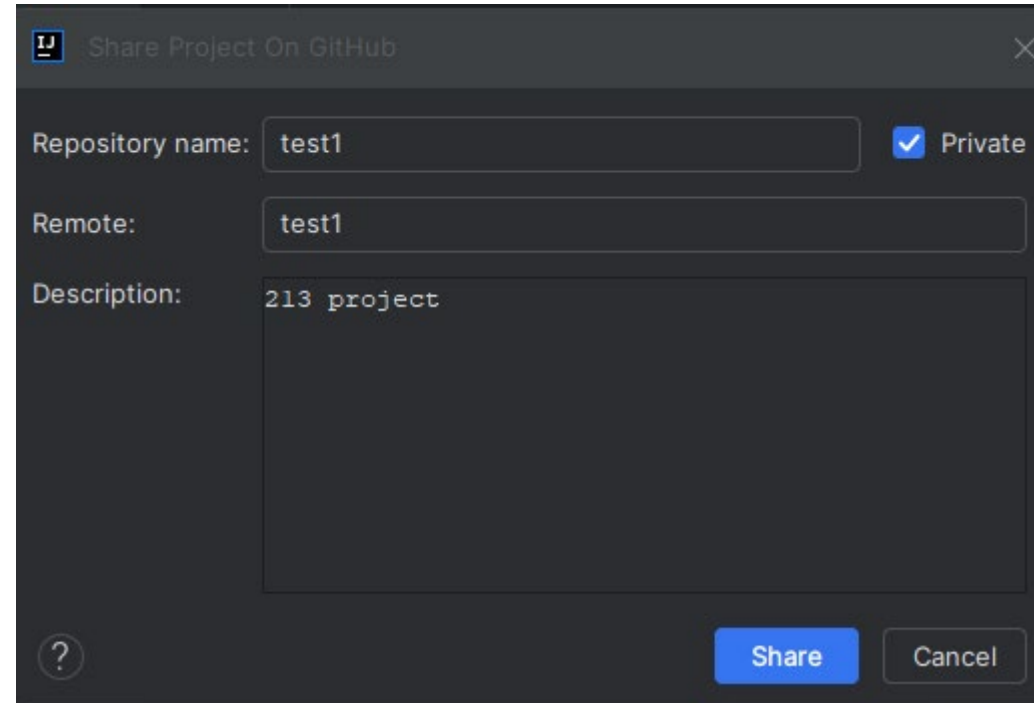
Using IntelliJ with GitHub

- Commit the changes to local repository (1)
- Push the changes to remote repository (2)
- Must define the remote repository the first push (3, 4, 5)



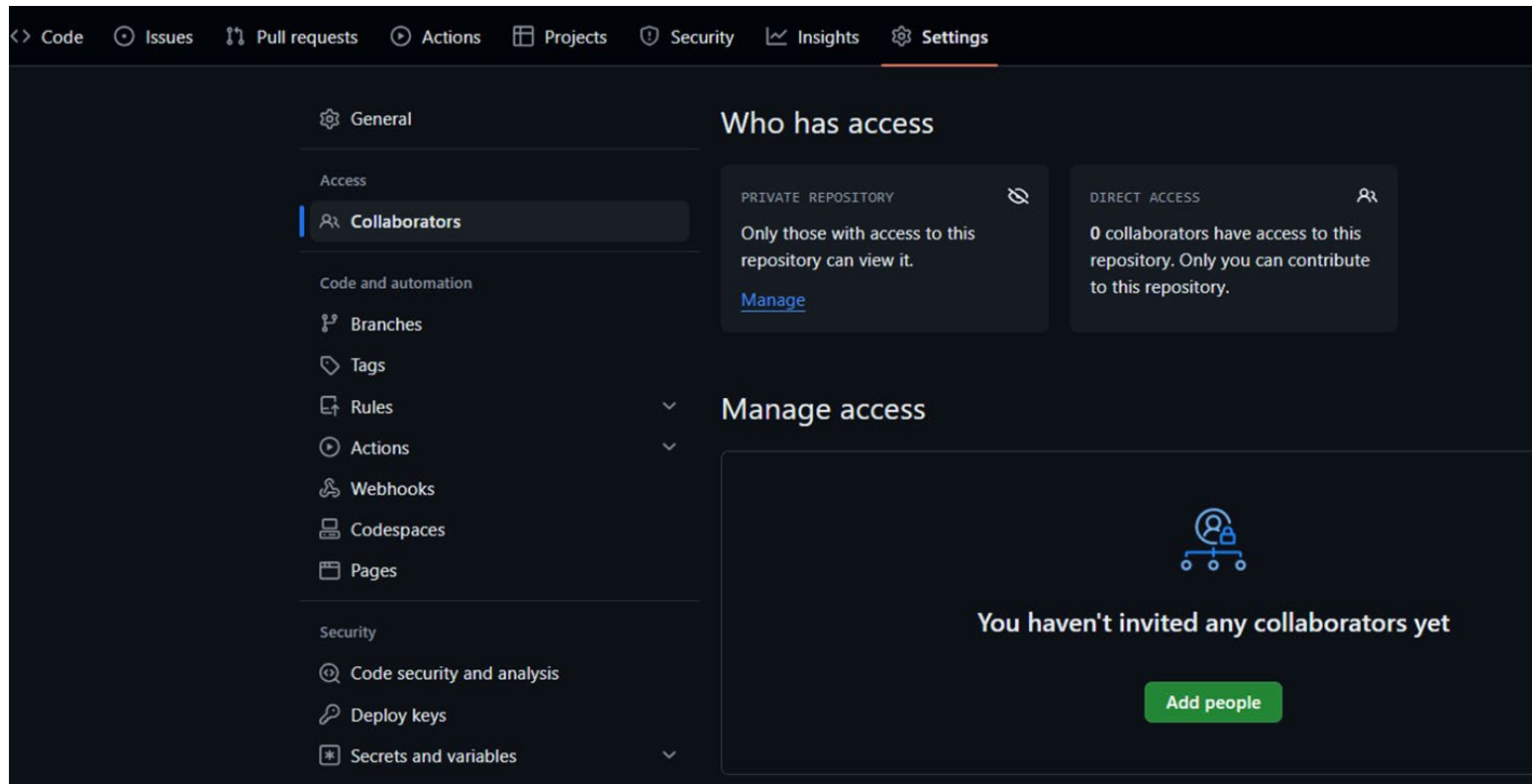
Using IntelliJ with GitHub

- If you create a project in IntelliJ and haven't created a repository on GitHub
- From the Menu
 - Select Git/GitHub/Share Project on GitHub
 - Enter the repository name and make it private
 - This will create a new repository on the remote site



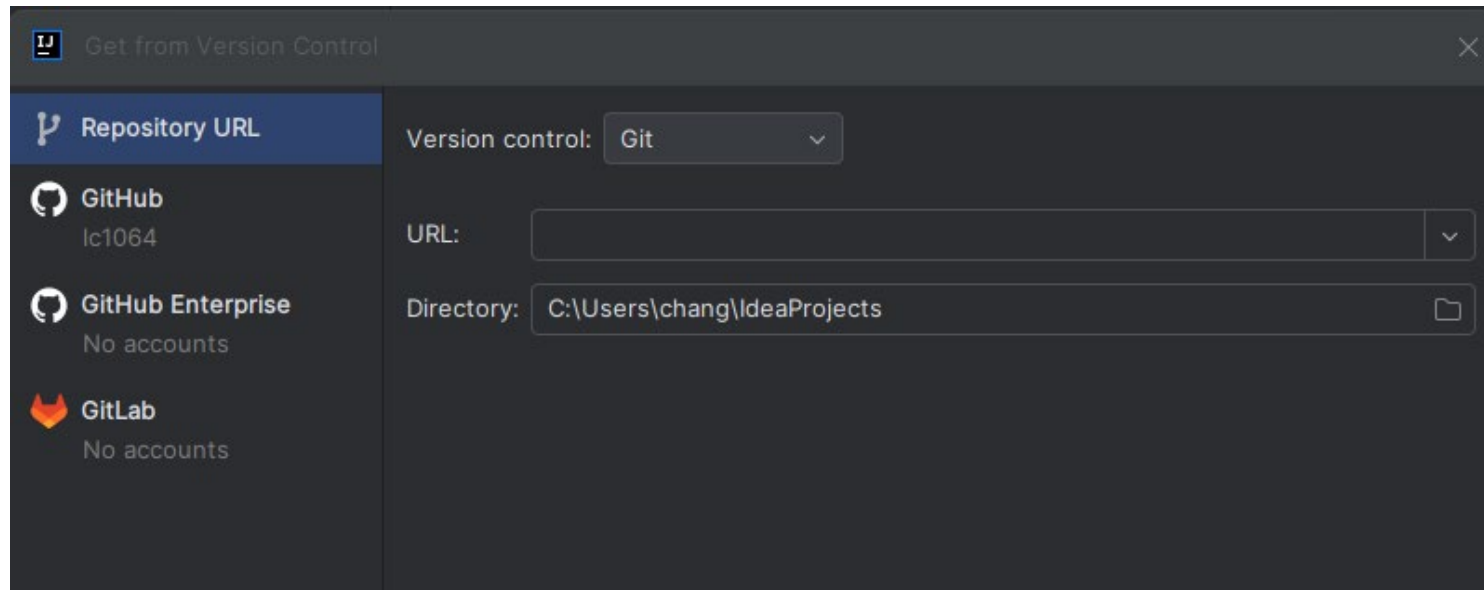
Using IntelliJ with GitHub

- If you have created a repository and wanted to grant access to your partner
- In GitHub, select Repositories/Settings/Collaborators and Add people



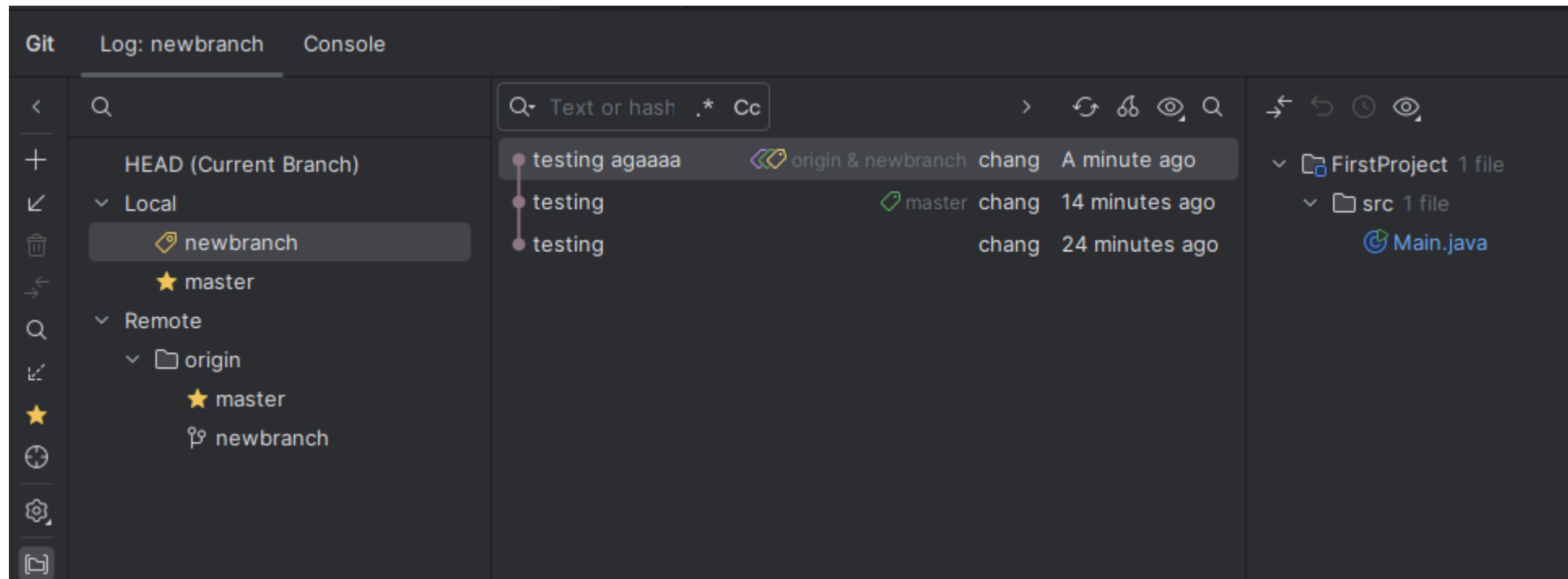
Using IntelliJ with GitHub

- If your partner created a repository on GitHub, you could create a project from GitHub repository
- In IntelliJ, from the Menu
 - Select File/New/Project from Version Control
 - Enter the URL of the repository



Using IntelliJ with GitHub

- If you don't want to mess up your stable version on GitHub, create a new Branch locally, then merge the branches after the branch is well tested



Using IntelliJ with GitHub

- Merging the branches

