2019

Lecture 1: Mobile Application & Product Development

NYC Tech-in-Residence Corps
rdomanski@sbs.nyc.gov

Bhargava Chinthirla
CUNY John Jay College

Eric Spector
CUNY John Jay College

CSCI 380-04

Mobile Application and Product Development

01/30/2019

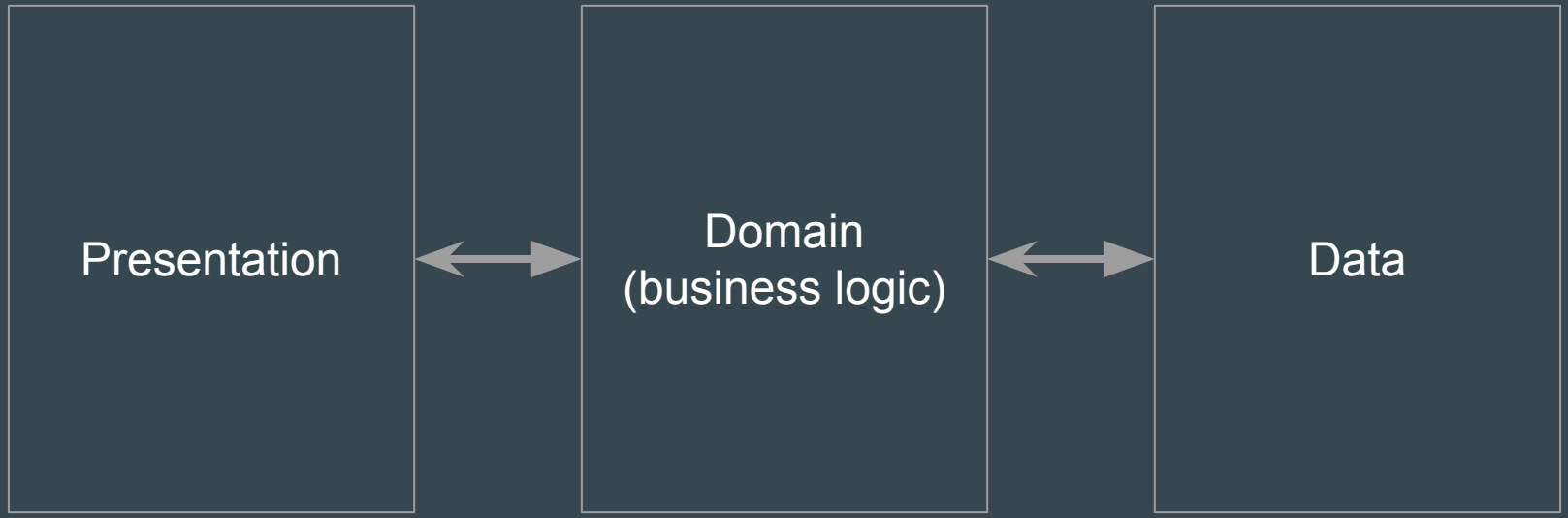
Course information

- **Adjunct Lecturers from Tech-In-Residences Corps:**
  - Bharg (bchinthirla@jjay.cuny.edu)
  - Eric Spector (espector@jjay.cuny.edu)
- **Time:** Wednesdays, 5:55 PM - 8:35 PM
- **Website:** https://bhargman.github.io/csci-380-04/
- **5 min Survey:** https://tinyurl.com/y7t7ycds

Course objective

The goal of this course is to teach you modern Android application development and software project management. Upon successful completion of this course, you will have planned, developed, and tested your own Android applications.

Android application layers



Textbook/Materials/Resources

- https://developer.android.com/
- https://android-developers.googleblog.com/
- **Optional:** Android Programming: The Big Nerd Ranch Guide

Grading Policy

- Class participation and group work - 15%
- Programming Assignments - 25%
- Midterm exam - 30%
- Final project - 30%

Academic Integrity/Honesty Policy

“You only learn if your work is your own.”

- https://www.jjay.cuny.edu/academic-integrity-0

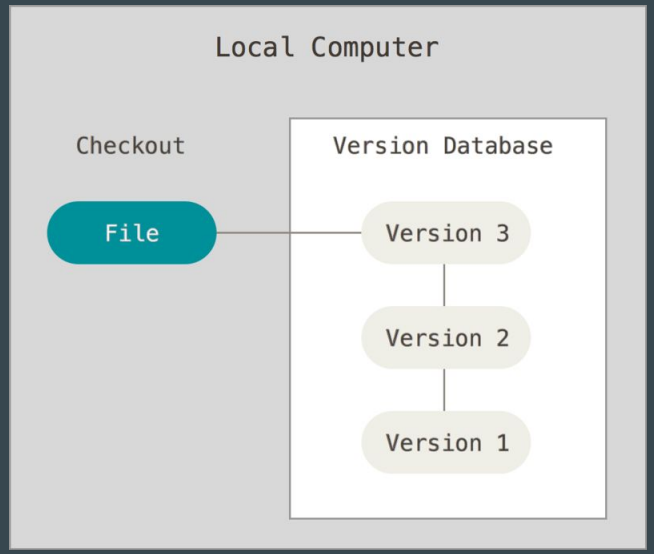
Other Classroom Policies

- No makeups will be given for the midterm.
- You will have two weeks for each programming assignment. Late assignments will be graded down 10% per day late.
- The lecturers reserve the right to give unannounced quizzes if it appears students are not putting the time in to prepare for class.
- See Syllabus for the full list of policies and the full schedule.

Did you prepare?

Local Version Control Systems (LVCS)

- File change history is kept on a local database, allowing the user to recreate what a file looked like at any point in time (e.g., *RCS*).



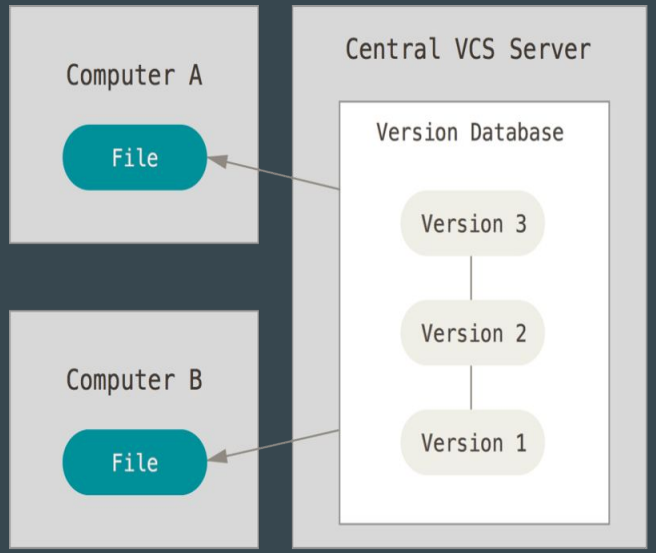
https://git-scm.com/book/en/v2/images/local.png

Disadvantages of LVCS and CVCS

- LVCS:
  - No capability to collaborate/share with other developers.
  - Single point of failure - broken hard drive + no back ups == ☠
- CVCS:
  - Single point of failure - broken hard drive + no back ups == ☠

Centralized Version Control System (CVCS)

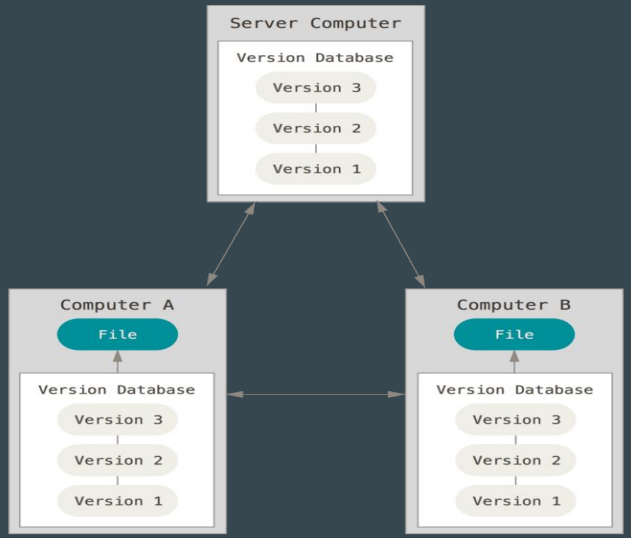
- File change history is kept on a private centralized server, allowing a team of developers to collaborate on a shared project by checking out the latest “snapshots” of files (e.g., *CVS*, *Subversion*).



https://git-scm.com/book/en/v2/images/centralized.png

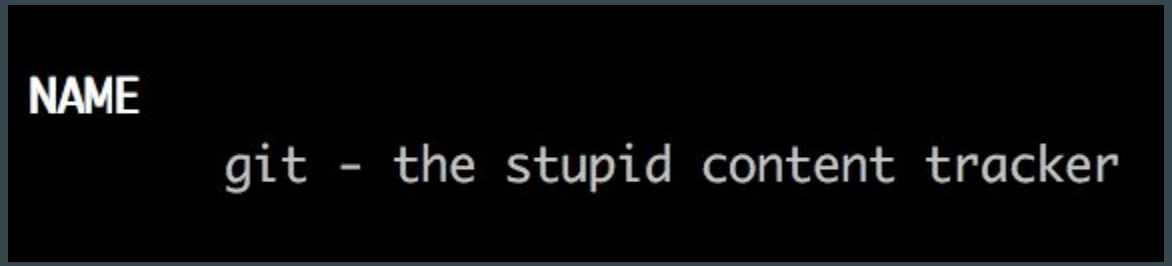
Distributed Version Control System (DVCS)

- Clients fully mirror a code repository, including its history (e.g., Git, Mercurial).



https://git-scm.com/book/en/v2/images/distributed.png

Git



- **Free resource:** https://git-scm.com/book/en/v2

Delta based version-control

https://git-scm.com/book/en/v2/images/deltas.png

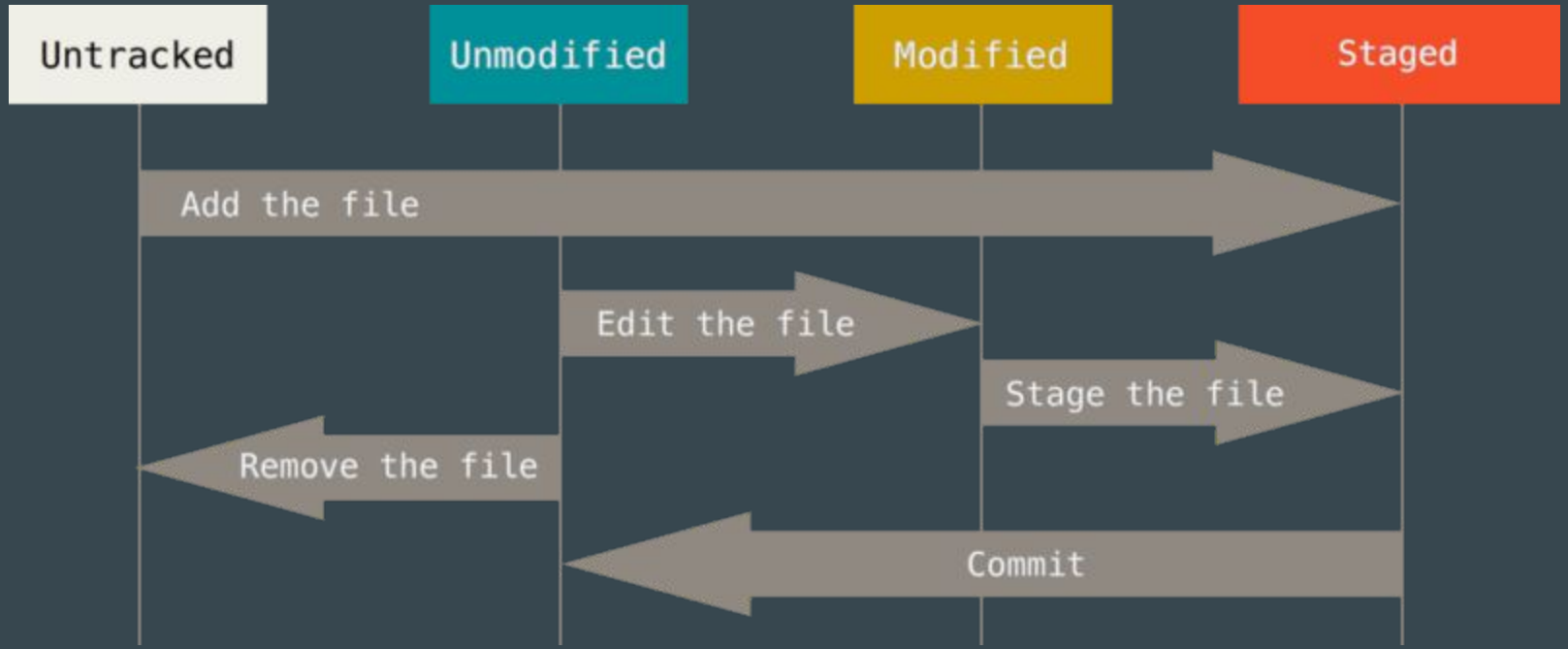
Git - stream of snapshots

https://git-scm.com/book/en/v2/images/snapshots.png

Git - initialization

- Any directory can be turned into a git repository by using running `git init` in command line.
- Once a directory is initialized as a git repository, create a file called `.gitignore` and type out any files/directories that you don't want git to track. For example:
  - Build directories and output binaries
  - Secret files
  - IDE configuration files
  - Sample `.gitignore` file for an android repo: https://github.com/github/gitignore/blob/master/Android.gitignore

Git lifecycle diagram



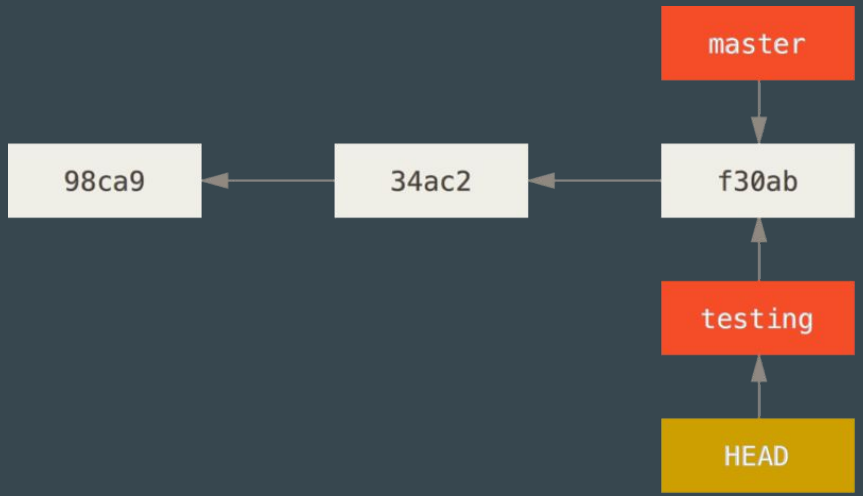
https://git-scm.com/book/en/v2/images/lifecycle.png

Git lifecycle commands

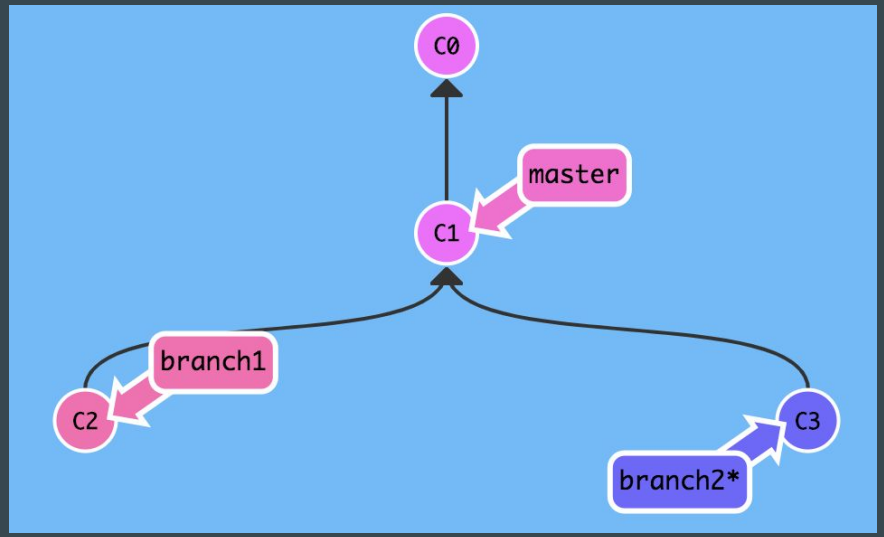
- `git add .`
  - Stage all new and edited files to get them ready to commit
- `git reset .`
  - Unstage all staged files
- `git commit -m“Commit message”`
  - Commit all staged files as a snapshot
- `git status`
  - Your best friend

Git - branching

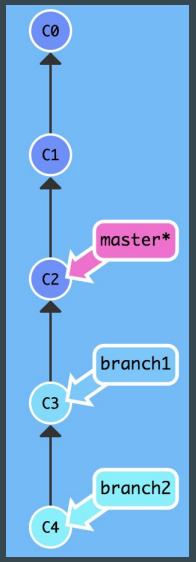
- All git repos start off on the `master` branch.
- Use `git checkout -b [branch_name]` to create a new branch off of the current branch and check it out. For example, `git checkout -b testing`:



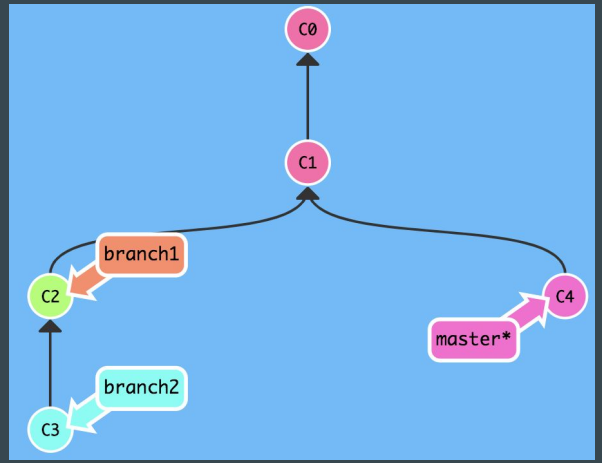
https://git-scm.com/book/en/v2/images/two-branches.png



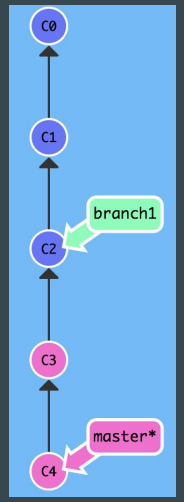
Repo 1



Repo 2



Repo 3



Repo 4

Git - merging and rebasing

- When collaborating with other developers, you'll want to pull in their changes from time to time into your own branch.
- Merging: Pulls in commits from another branch (e.g., from `master` to your working branch) as a new `merge commit`.
- Rebasing: Pull in commits from another branch (e.g., from `master` to your working branch) and copies your working branch's commits one-by-one, preserving a clean linear history.

Github

- To collaborate with other developers, you need to manage your repositories on remote servers (e.g., Github, Gitlab, etc.).
- We'll be using Github for most of your programming assignments and for your final project.
- To authenticate your local machine with your Github account, set up an SSH key: https://help.github.com/articles/connecting-to-github-with-ssh/

Android Studio + Java

- Unlike compiled languages (e.g., Pascal or C), Java uses both a compiler and an interpreter.
- Compiled languages need to be re-compiled when their program needs to run on a different architecture.
- Java is compiled (to java bytecode) *and* interpreted (by Java Virtual Machine), so any java program that's compiled on one architecture can also run on any other architecture (as long as it has a java interpreter).
- Android Studio comes bundled with OpenJDK, an open source implementation of Java, so you don't have to install any other SDKs in order to develop.

Java - Data types and variables

```
System.out.println("hello world");
```

`hello world` is a literal (actual data)

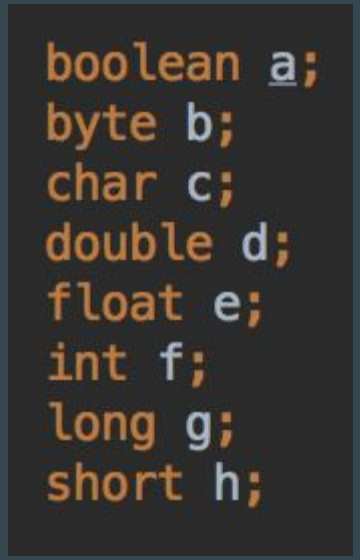
```
int count = 4;
```

`count` is a variable (a container for data) and `int` is its type (i.e., the kind of data it is)

```
public static void main(String[] args) {
    int count = 4;
    System.out.println(count);
}
```

Just like C/C++, Java has its own main method, which is the entry point into a program.

Java - Primitives



Primitives are built in data types in a programming language. The image to the left shows the primitive data types of Java. They're the building blocks of a much more complicated data type, known as `class`.

For example:

- `String` class is an array of `char`.
- `Date` class can be seen as a composite of three `int`s (for month, day, and year).

The variables (or `fields`) of a `class` define what sort of operations are valid on it, these operations are known as `method`s.

Java - methods

```
AccessModifier ReturnType MethodName(optional parameter list) {

}
```

- `AccessModifier` - used to control the visibility of a method
- `ReturnType` - used to declare what type of data this method returns (void returns nothing)
- `MethodName` - used to name the method
- `optional parameter list` - can have 0 to n types of parameters

There is a special type of method called the `constructor`, which can be called when an object needs to be created using `new`. Java has *no* concept of `destructor` methods.

Java - class and method

```java
class Student {
    String name;
    String id;
    int level;

    Student(String name, String id, int level) {
        this.name = name;
        this.id = id;
        this.level = level;
    }

    void printName() {
        System.out.println(name);
    }
}
```

```java
Student student = new Student( name: "Bharg",  id: "3D2Y",  level: 1);
student.printName();
Student student1 = student;
```

Java - conditional, iteration, and arrays

- These are all really similar to how C/C++ does things, so we don't need to go in depth.
- If you've used any sort of these statements in another programming language, chances are you'll pick this up in Java really easily.
- See sections 4, 5, and 6 in the primer reading for assignment 1 if you'd like to go in depth.

Java - inheritance

```
class <child-class> extends <parent-class> {

}
```

- Inheritance is the process of one class inheriting the features (fields and methods) of another class
- A class can only *inherit* from **one** other class
- A class can be *inherited* by any number of classes
- So, a child can have only one parent, but a parent can have many children

```java
class Person {
    String name;
    String id;

    Person(String name, String id) {
        this.name = name;
        this.id = id;
    }

    void printName() {
        System.out.println(name);
    }
}

class Lecturer extends Person {
    String email;

    Lecturer(String name, String id, String email) {
        super(name, id);
        this.email = email;
    }
}

class Student extends Person {
    int level;

    Student(String name, String id, int level) {
        super(name, id);
        this.level = level;
    }
}
```

```java
Student student =
        new Student( name: "Bharg",  id: "3D2Y",  level: 1);
Lecturer lecturer =
        new Lecturer( name: "Chin",  id: "524",  email: "bchinthiral@jjay.cuny.edu");
student.printName();
lecturer.printName();
```

Java - abstract and interface classes

- Abstract classes:
  - cannot be instantiated, but can contain implementation of fields and methods
  - can only be inherited (by using `extends` keyword)
- Interface classes:
  - cannot be instantiated, and cannot contain any implementation details
  - used to specify a “contract” or “blueprint” of what methods a class can implement
  - can only be implemented (by using `implements` keyword)
- A class can *extend* only one type, but can *implement* many types

Java - access modifiers

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html