CSC 3380
Aymond

Section 1

2/3/2020

Project News

- Next Milestone: #1
  - Due Tuesday 2/4, 11PM
  - Upload to Moodle (1 upload for entire team)
  - Outline is in Project Kickoff Lecture Notes

Enterprise Architecture

- We will spend some time talking about EA on Monday, when we begin our UML discussion
- Instructions for access will be posted to Moodle

- EA is installed on the 2324 PFT and 2326 PFT lab machines
  - And maybe 2317 PFT
- Students may enter 2326 PFT using the keypad at the back door

- Lab hours will be posted to Moodle, but check lab doors for updated information

Object Oriented Design

- **Source Control++**
- The Design Process
- Software System Architecture
- Architectural Styles
- Object Oriented Design Principles

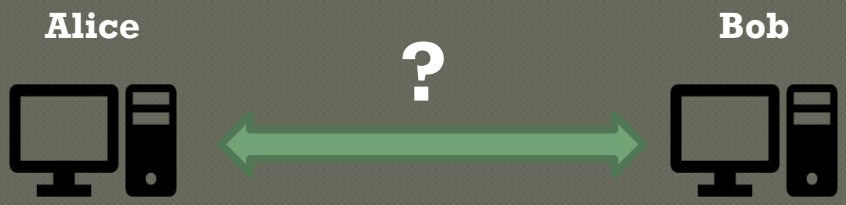
Source Control: The Problem

- Alice and Bob are working on a project
- Alice is programming the backend, Bob the GUI

- Can't be totally isolated, Bob needs a test GUI, Alice needs a working backend

- Two separate versions of the code: Alice's and Bob's
- Alice updates the backend; bob updates the frontend

- Neither person's code works without incorporating the other's
- How to fix?



The Centralized Solution

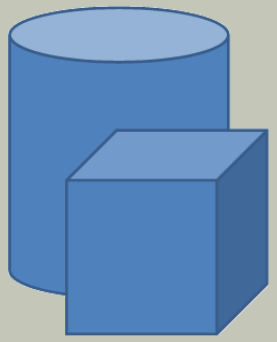
- Have a central master project shared by all.
- Everyone is forced to check master before pushing changes.
- Keeps changes small and manageable.

Centralized Solution Problems

- People now need to be connected to a network.
- Local experiments more difficult.
- Single point of failure.

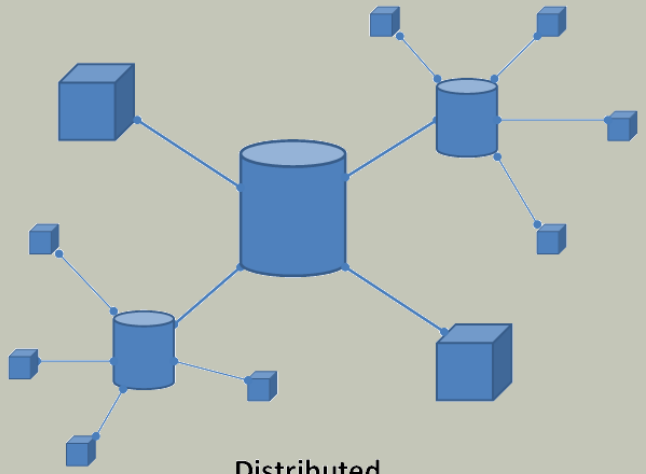
Decentralized (peer-to-peer)

- Every developer owns their own repository
- One is set aside as a central master
- Experimentation is easy, recovery is easy
- Parallel development is easier



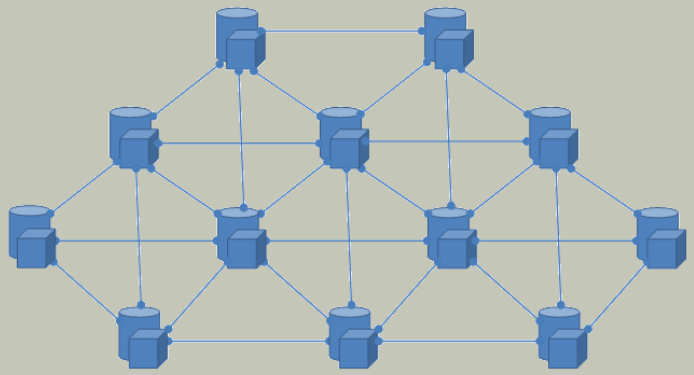
Centralized

one node does everything



Distributed

nodes distribute work to sub-nodes



Decentralized

nodes are only connected to peers

Git

- Led and named by Linus Torvalds
- Most popular source control system
- Distributed, extremely fast
- Complicated

The Repository

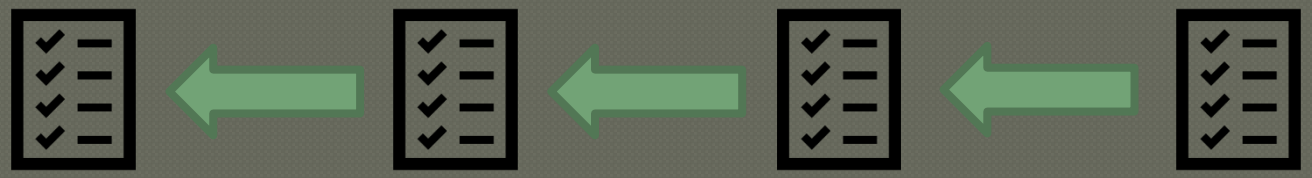
- Source control is organized around databases called *repositories (repos)*
- A repository is like a special directory for your code
- It has special features that make programming easier, it's more than just sharing a directory

Repository Features

- Repositories keep a history of every significant change made to every file in your source tree
- When you finish a version of your software, you commit it, which marks it as significant
- You then have a timeline of changes
- People can download particular versions

Commits

- A commit is the basic unit of change in a repository
- A commit encapsulates new/removed files and changed lines
- Each commit has a parent that its changes depend on
- The chain of commits forms a timeline



Annotate Commits

- Once you've written a chunk of code, write what you did
- Explain your changes as an imperative:
  - "Add logging to state.java ."
  - "Remove unnecessary comments."
  - "Refactor RoomBuilder to use static constructor."
- Keep your changes granular (only changing what they must)

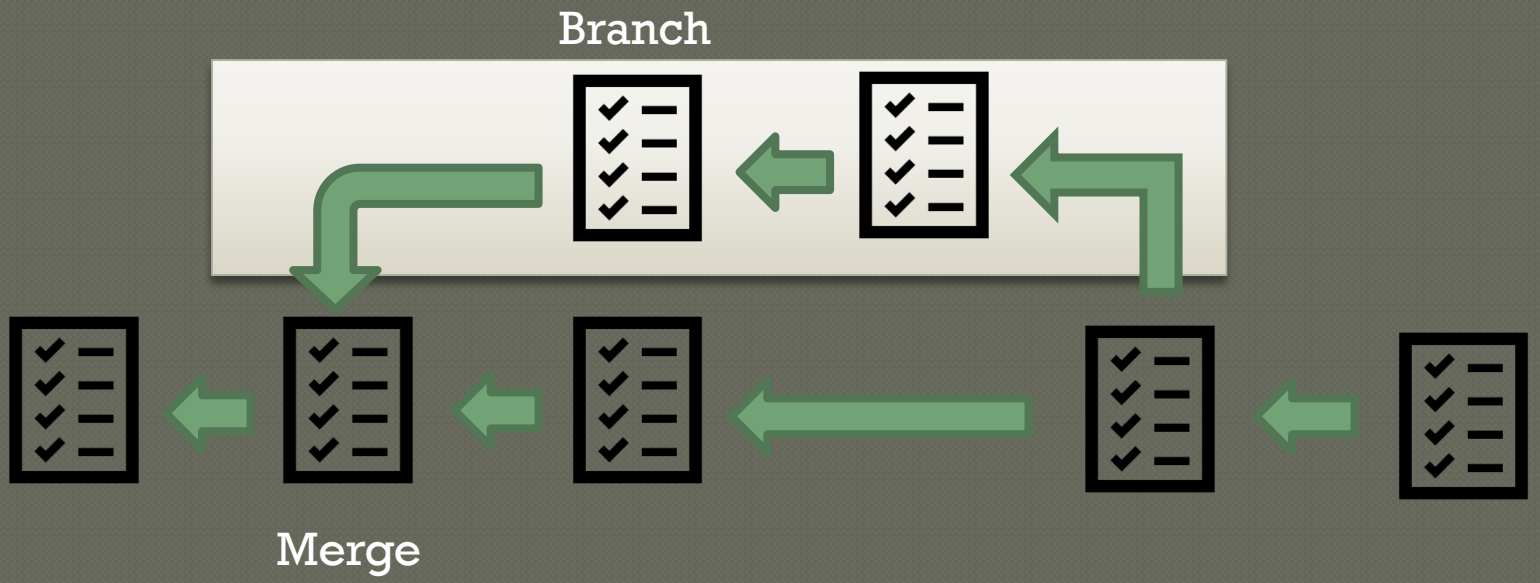
Commit Guidelines

- Keep your commits atomic
- DON'T BREAK THE BUILD

Branches

- Sometimes we need to experiment with a feature
- We want a branch in our timeline, to see if the feature works
- We can compare the software with, and without the feature
- If we like the feature we can merge its commits

Branching Diagram



Merging

- Merging creates a single commit with the changes from both commits
- Sometimes these commits conflict with each other:
  - **Branch1:** `puts( "Hello, world!" );`
  - **Branch2:** `puts( "Goodbye, world." );`
- How do we solve this?

Fixing Merge Conflicts

- **Must be fixed manually, the computer doesn't know which code you wanted.**

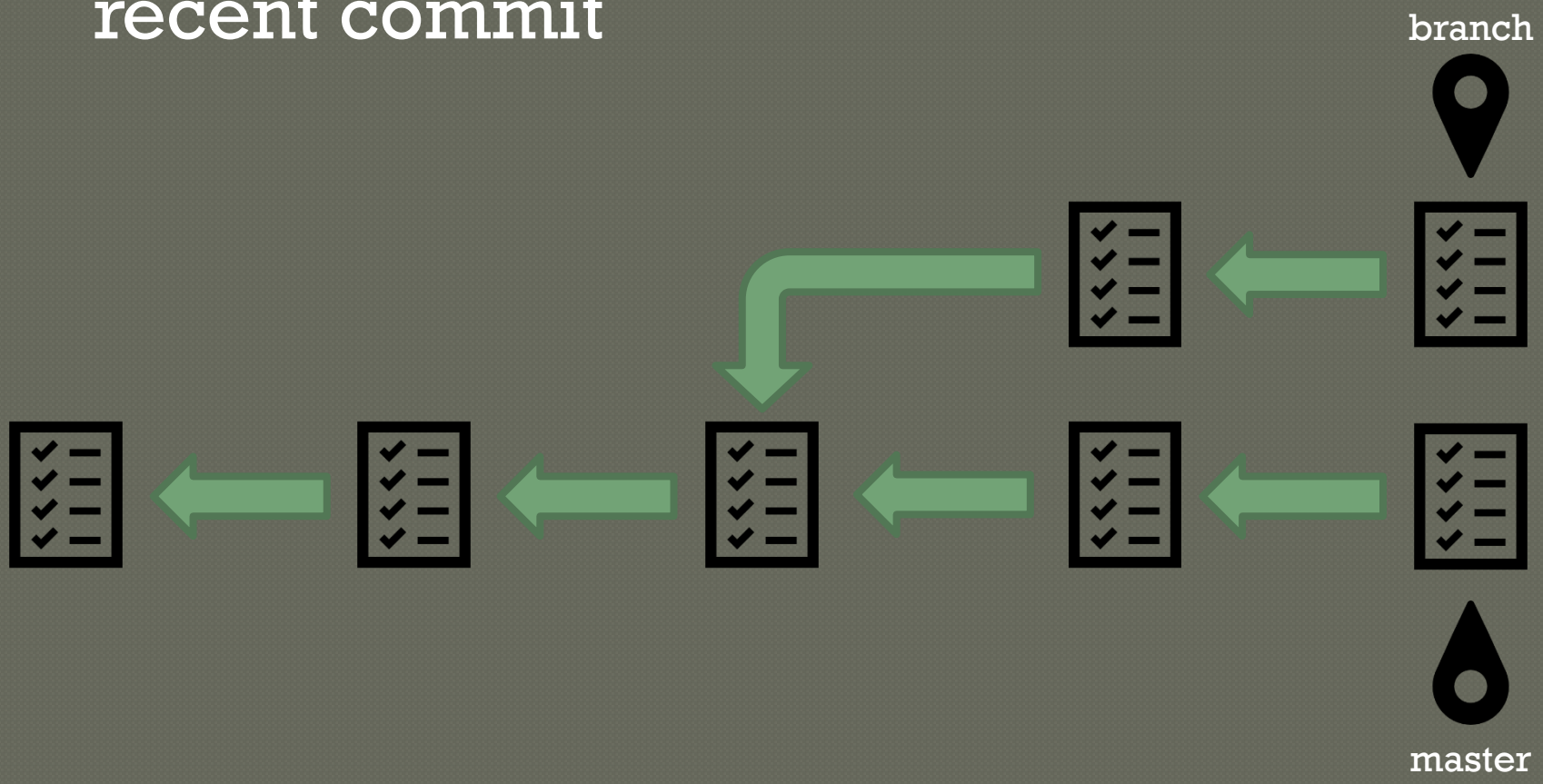
```
<<<<<<<
puts( "This is the code from the
branch being merged into." );
=======
puts( "This is the code from the
branch being merged from." );
>>>>>>>
```

How Branches Work

- Branches are pointers/references to commits
- Each commit stores its ancestors, so branches are like references to linked-list nodes
- The main branch is called master

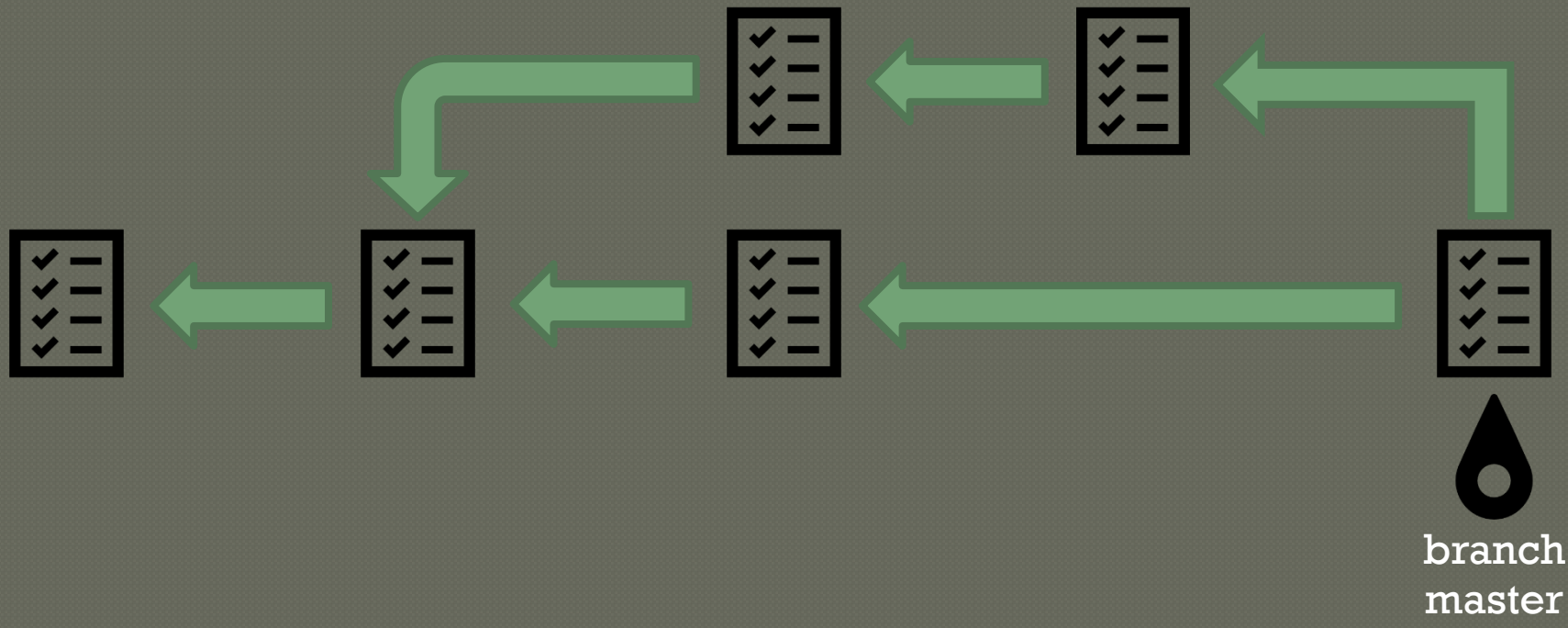
How Branches Work

- branch normally points to the most recent commit



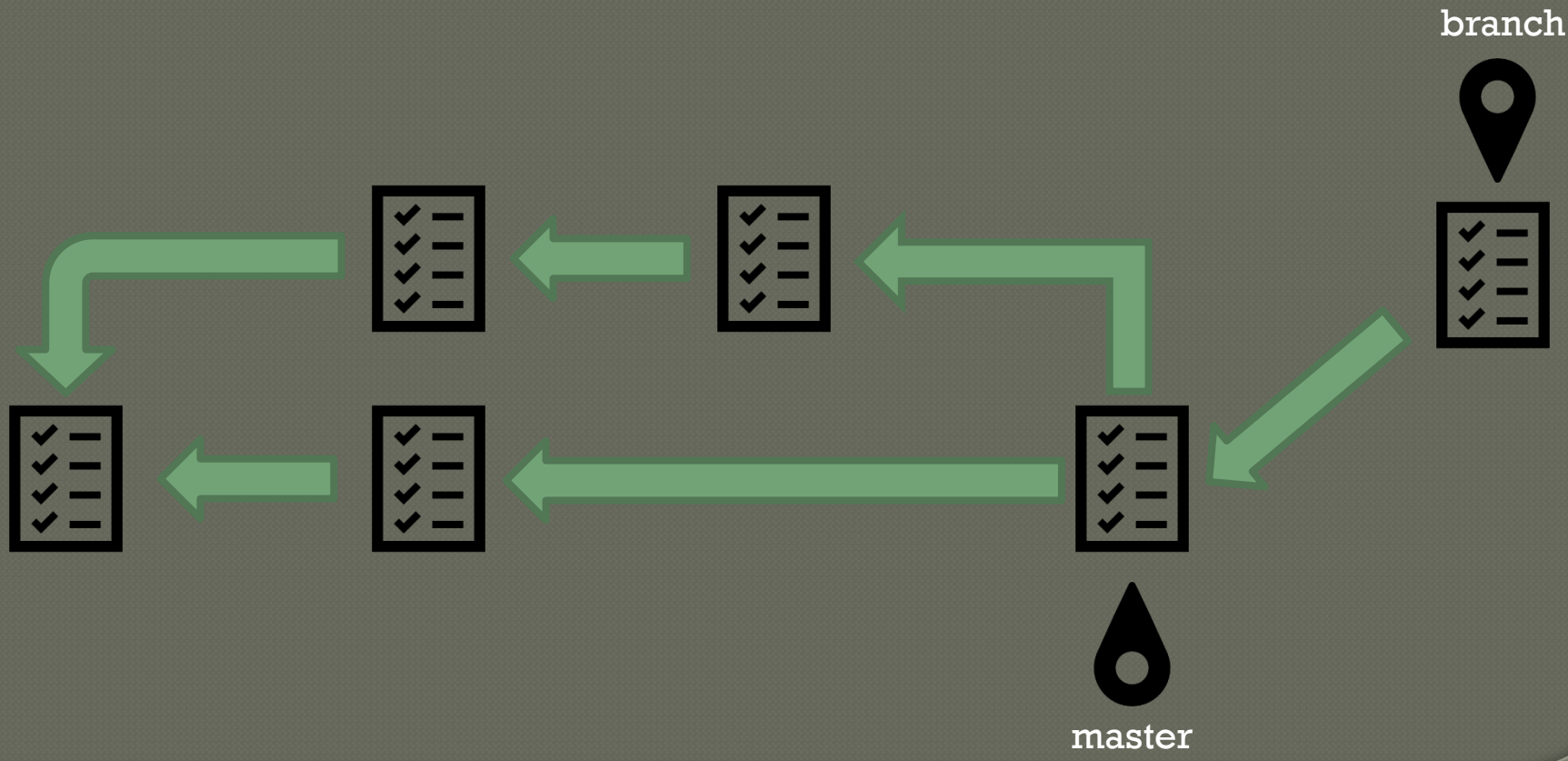
How Branches Work

- After merging, the branches point to the same commit



How Branches Work

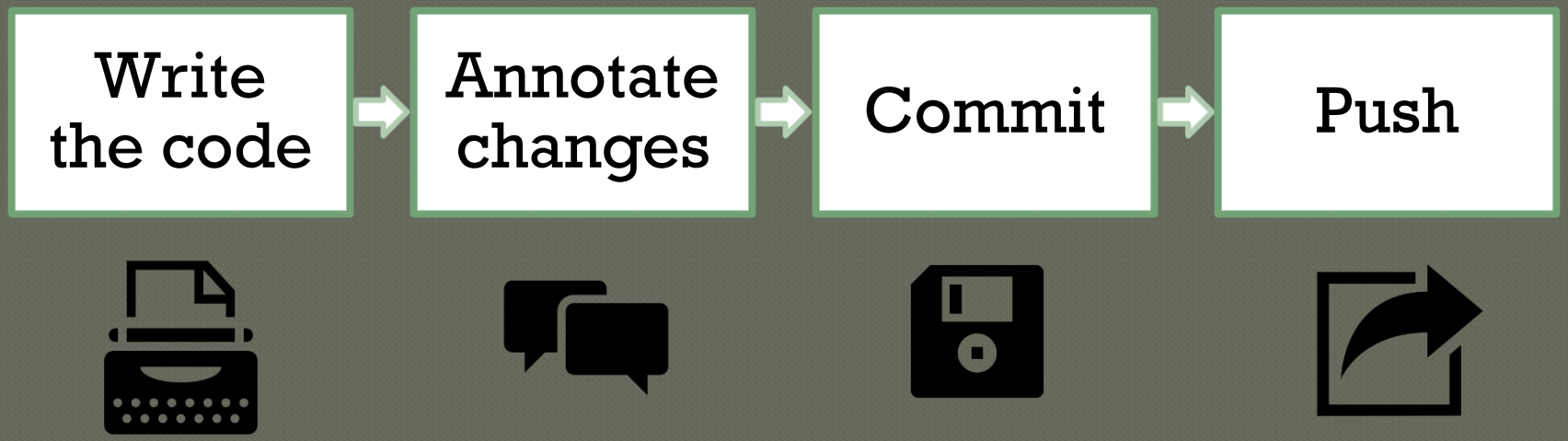
- They can split off again.
- They can be pointed to another commit



Pulling and Pushing

- No one else sees your changes until you push
- Pushing is automatic, your cloned repo remembers where to push
- Have to pull before you can push (to make sure you've seen changes before overwriting them)
- Pulling can cause a merge conflict
- Must fix conflict before pushing

Your Workflow



.gitignore

- A file named .gitignore contains a list of patterns.
- Each line has one pattern. Patterns can use wildcards.
- Files in the current directory or below which match any pattern are ignored.
- Use for derived files, garbage, and resources:
  - Images
  - Binaries
  - Logs

Object Oriented Design

- Source Control++
- **The Design Process**
- Software System Architecture
- Architectural Styles
- Object Oriented Design Principles

The Design Process

- Purpose
  - Specify a solution to a given problem (usually expressed as a functional specification)
- Software System Architect
  - Postulates a solution
  - Models it in a design framework
  - Establishes and maintains the vision for the solution
  - Evaluates design against original requirements
- Software Designer
  - Designs the internal working of system components
    - Defines subsystems
    - Crafts process logic
    - Details data flow between and within system components and external sources and interfaces
  - Produces a specification of the design, detailed enough that
    - A programmer can implement it
    - A tester can test it
    - A technical writer can document it

Objectives of the Design Process

- Produce a set of specifications that describe the intended form of implementation for the software system
- The design specifications
  - Describe
    - The form (structure) of the solution
    - The way that the components are to fit together
  - Act as a set of "blueprints" that show how the system is to be constructed

Design Phases

- Problem-solving
  - Extensive use of abstraction:
    - Separation of logical aspects of the design from the physical aspects of the design
    - **Create a metaphor for the system**
  - Making choices: tradeoffs between qualities attempting to achieve
  - Ultimate criterion: "fitness of purpose"
- Model-building
  - Build a highly abstract view of the chosen solution (architectural design)
  - Model the detailed interaction between components (detailed design)

Desirable Features…

- Fitness for purpose
  - The system must work, and work correctly
  - It should
    - perform the required tasks
    - in the specified manner and
    - within the specified constraints
    - of the specified resources
- Robustness
  - The design should be **stable against changes** such as features as file and data structures, user interface, etc.