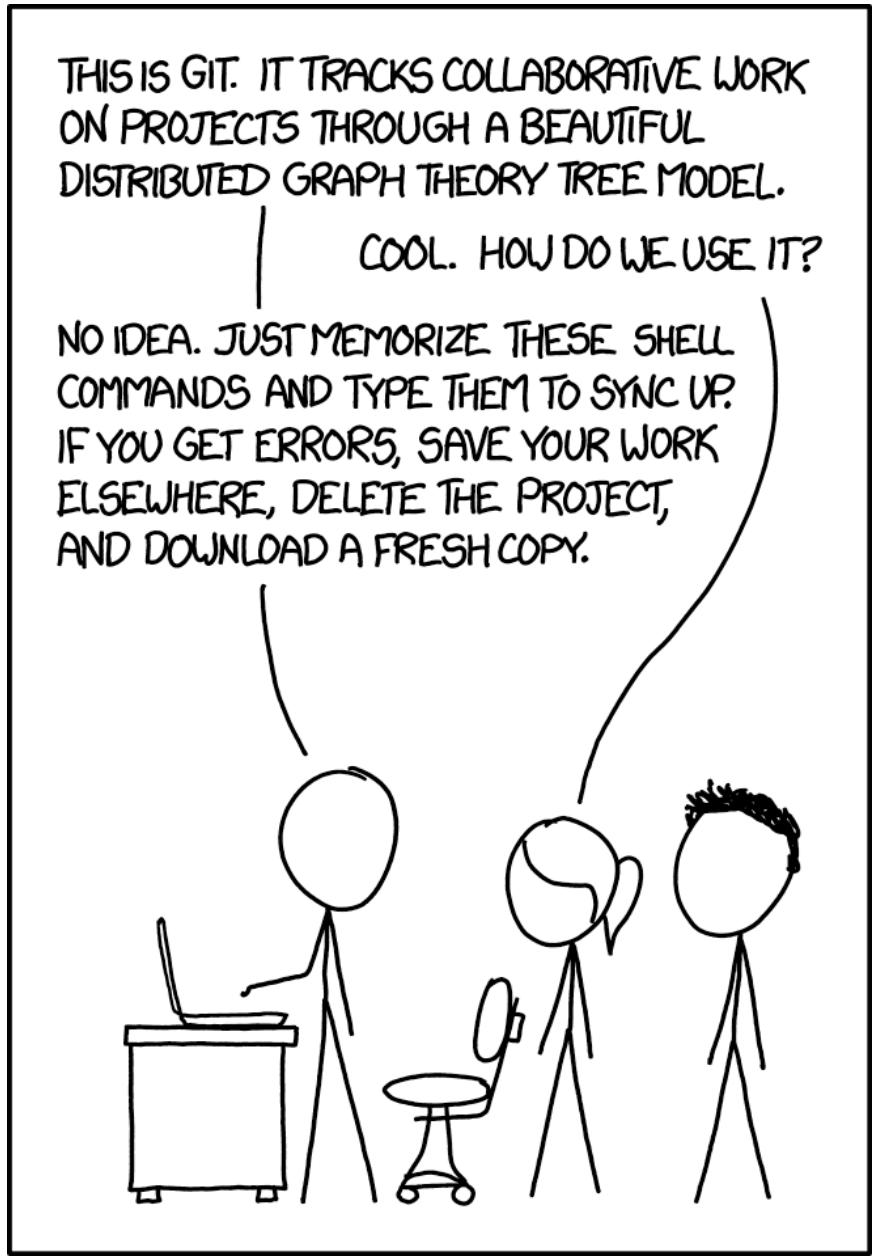

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.


I DON'T ALWAYS IGNORE YOUR EMAILS
BUT WHEN I DO IT'S BECAUSE THE ANSWER IS IN THE SYLLABUS

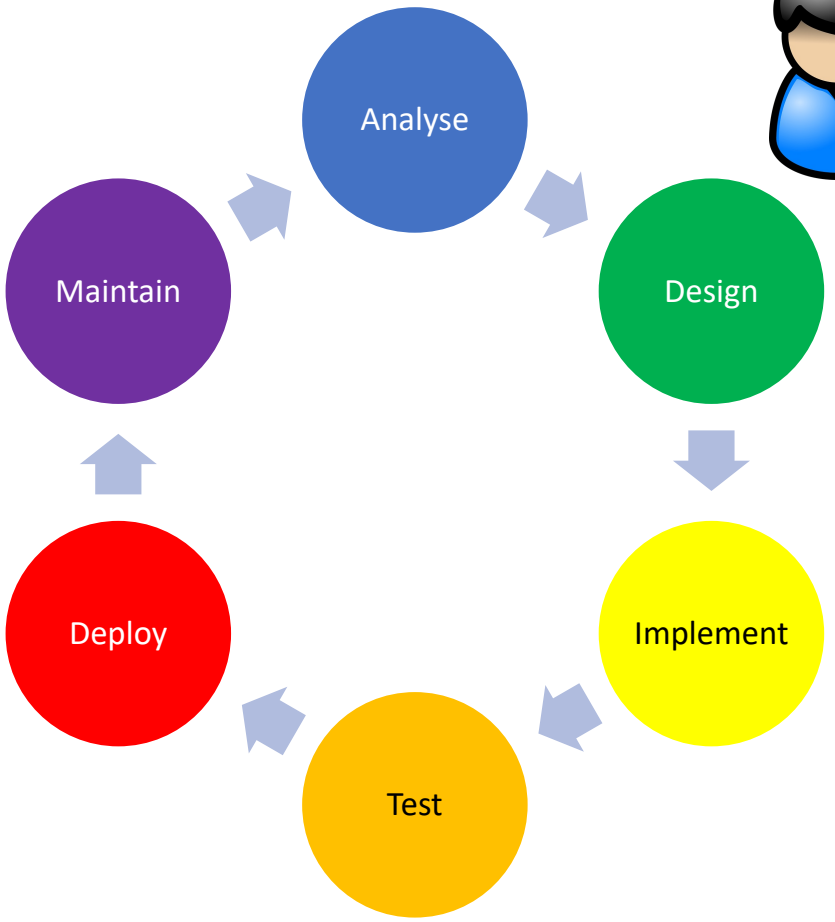
Tools of the Trade

CSCI 2134: Software Development

Agenda

- Announcements
  - Welcome to class!
  - Very quick syllabus rundown. (See Lecture 2)
  - Lab 0 is next week September 11—15
  - No class September 11--15. Lecture 2 is asynchronous and online.
    - TAs will be available to help you set up your development environment for Lab 1 and Assignment 1.
- Lecture Contents
  - Brightspace Quiz
  - A Brief Survey of Software Development Tools
  - Integrated Development Environments
  - Version Control
  - Introduction to Git
- Readings:
  - This Lecture: Chapter 30
  - Next Lecture, Chapter 20, 31

We need the right tools to develop software

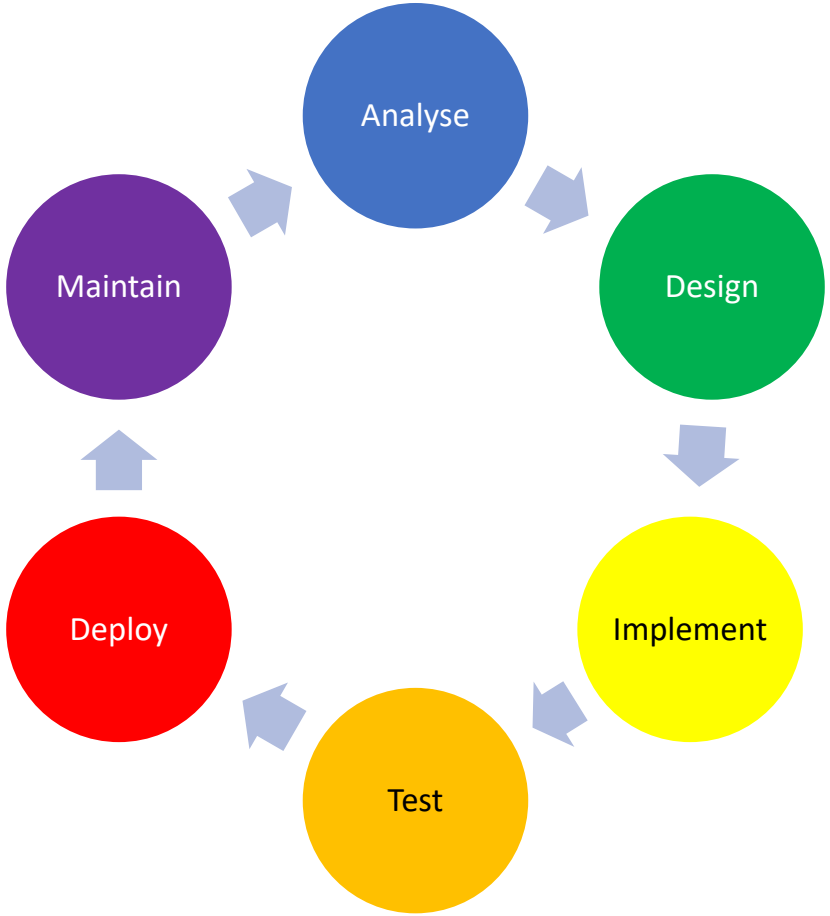


Have you ever …

- Used notepad or another editor to edit code?
- Debug code using print statements?
- Make copies of your code in case you need to return to a previous version?
- Share your code by emailing it?
- Felt frustrated with the tools you were using?

Software Development Needs Tools

- Design Tools:
  E.g., CASE tools, Diagram tools, UML tools
- **Integrated Development Environments**
  E.g., IntelliJ, Eclipse, …
- Compilers (program translators)
  E.g. javac, gcc, python, cpp, …
- Build tools
  E.g. make, ant, gradle, maven, …
- Frameworks, libraries, code generators
- **Version control**
  E.g., Git, SVN, RCS, …
- Code analyzers
- **Debuggers**
- Profilers
- **Testing frameworks**
  e.g., JUnit



Integrated Development Environment (IDE)

Modern IDEs are applications that incorporate (or can invoke) most if not all the tools that a developer needs!

Function	Comment
Editing	Built-in
Compiling	Invokes compiler to generate the executable
Building	Invokes external tools such as ant, make, gradle, or maven; or has built-in functionality
Version Control	Invokes external tools such as Git or SVN
Refactoring	Built-in
Debugging	Built-in
Profiling	Built-in
Testing	Built-in and/or support for external framewoks
Source Manipulation	Built-in. E.g., diffing, merging, cross-referencing, formatting, templates, linting, etc

Editing Functionality in Modern IDEs

- Console to see the output / integration with running the code
- Coloured text / text annotations / text formatting
- Line numbers / help to orient yourself in files and among files
- Complex find and replace
- Basic file browser / project manager
- Integrated debugging tools
- Links to documentation
- Method / function expansion or suggestion
- Safe execution environment
- Extensible (plug-ins, libraries, …)
- On the fly error detection / hints at common errors

IDEs Are Complex

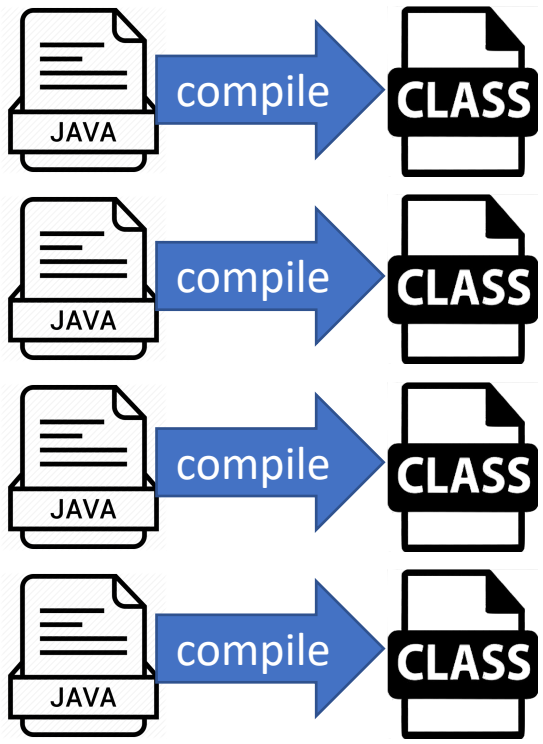
- Modern IDEs have a lot of functionality
- Consequently, they have many multilevel menus and keybindings
- They can be hard to learn!
- But, they allow us to be productive developers
- We will be using both IDEs and command-line for many of our tasks

- You are expected to learn an IDE.
- The standard IDE for this course is IntelliJ
- You can use another IDE, at your own risk

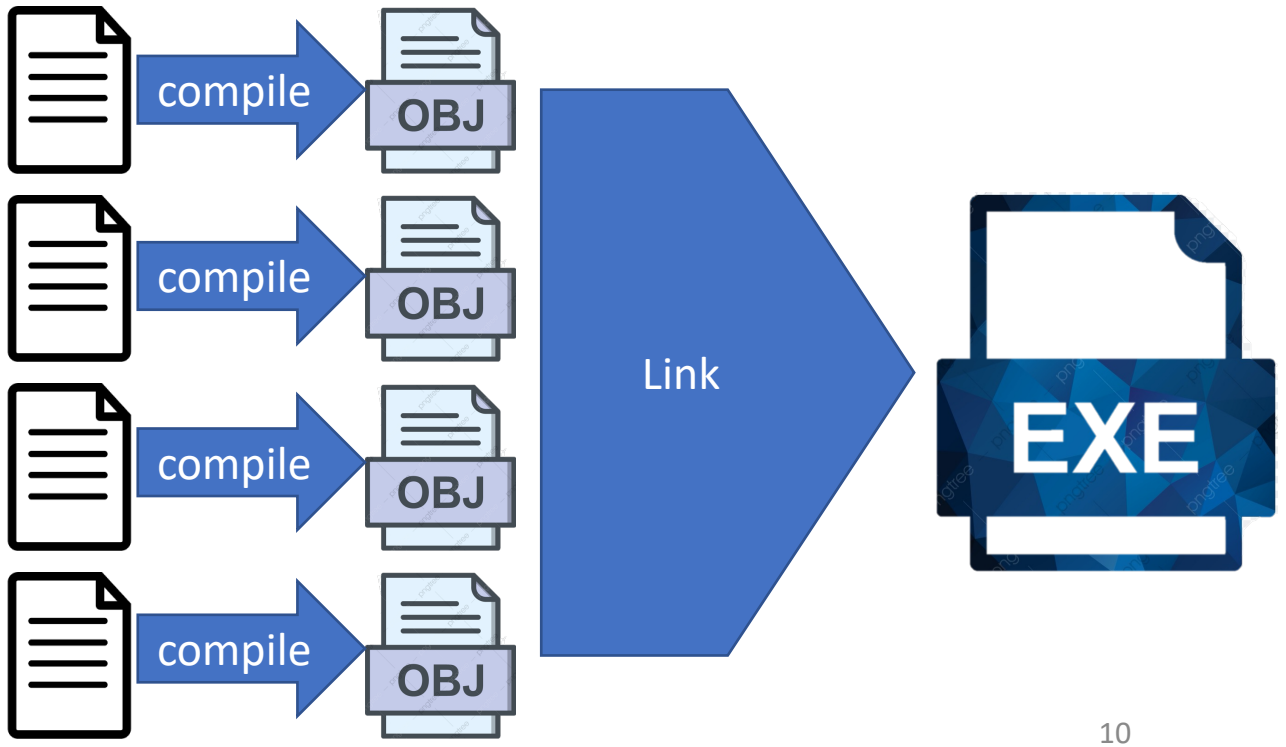
Executable Code Tools: Compilers and Linkers

Compilers and program translators translate human readable code into machine readable code.

Java

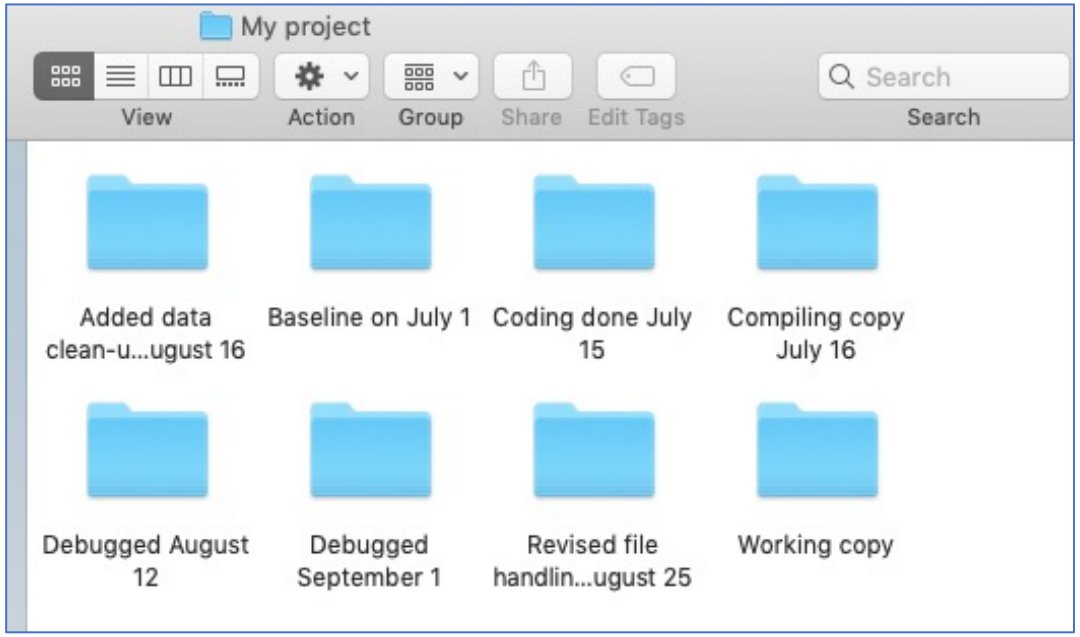


Most compiled languages

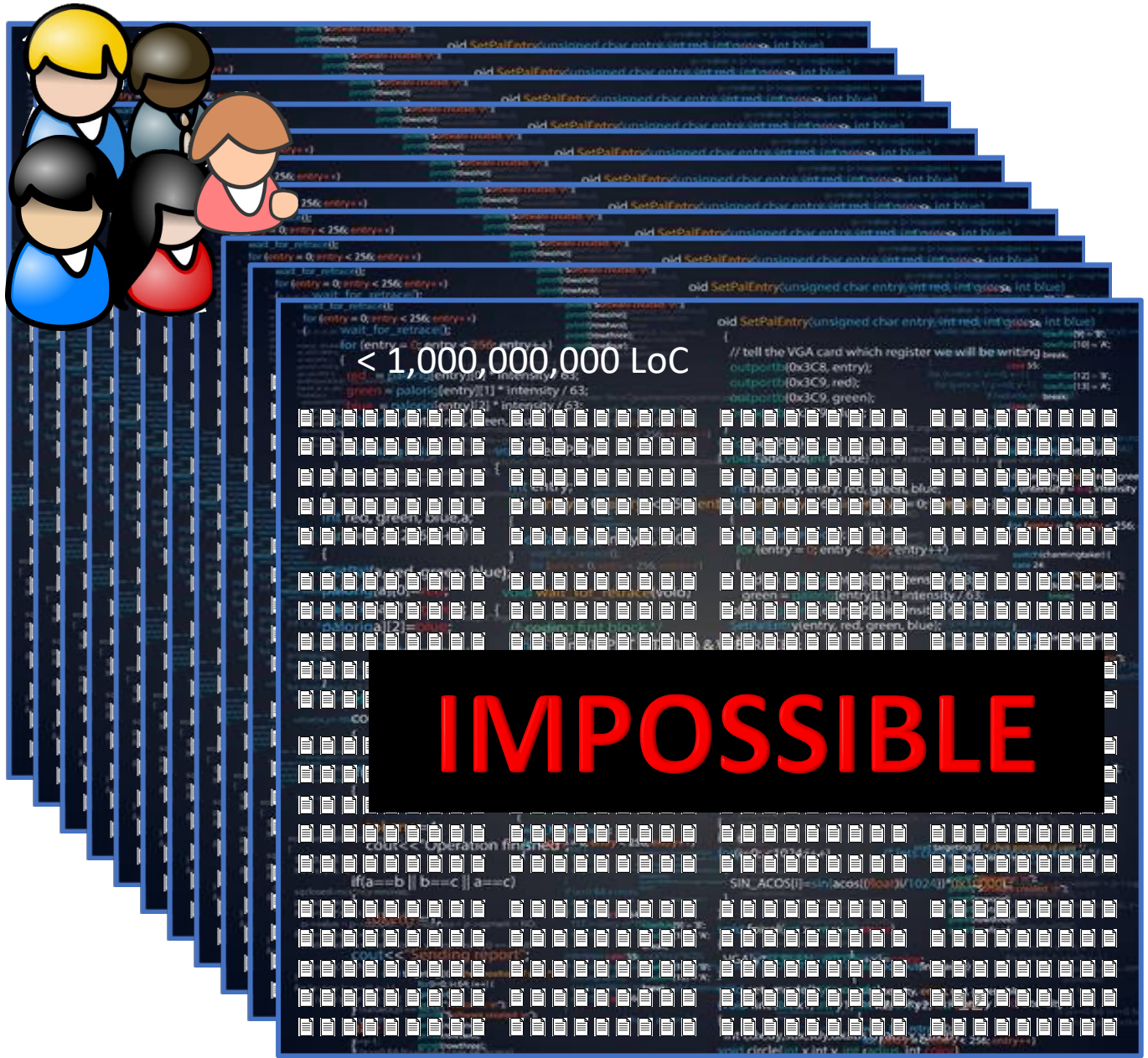


Version Control

- It’s common and useful to keep multiple versions of our code
  - Compare current versions with past versions
  - Revert to past versions
  - Experiment with new additions that may become part of the code
- Past Solution: Make a copy of the code
- Issues to consider:
  - Naming of directories (consistency)
  - Location and backup of copies
  - Tracking what was done where
  - Multiperson collaborations



Version Control by Copying



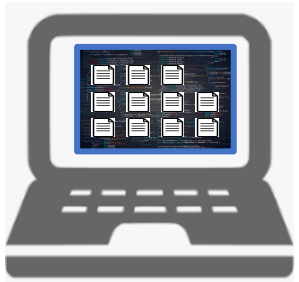
How Do We Manage Large Amounts of Code?

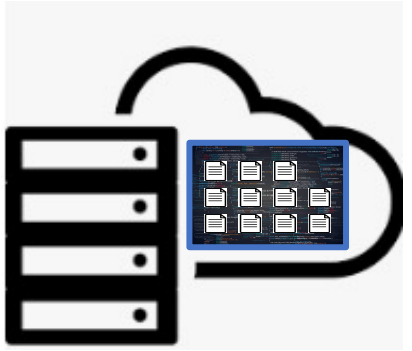
- Question: How do we manage large amounts of money used by many people?
- Answer: We use banks.
- Observation: Banks
  - Facilitate Deposits / withdraws from various accounts by multiple people
  - Keep track of all transactions
  - Make it possible to rewind transactions
- Version Control provides similar services for code

Idea: Use a Version Control System

First, some jargon:

- A **repository** is a collection of files that make up a software projects
- A repository may be
  - **local**: stored on the computer where the coding is being done
  - **remote**: stored on a server different from the current computer
- Every file in a repository has a version number, such as
  - Revision numbers, e.g., 1, 2, 3, 4, 5, …
  - Major/minor version numbers, e.g., 2.1, 2.2, 2.3, …
  - Unique file identifiers, e.g., hash values

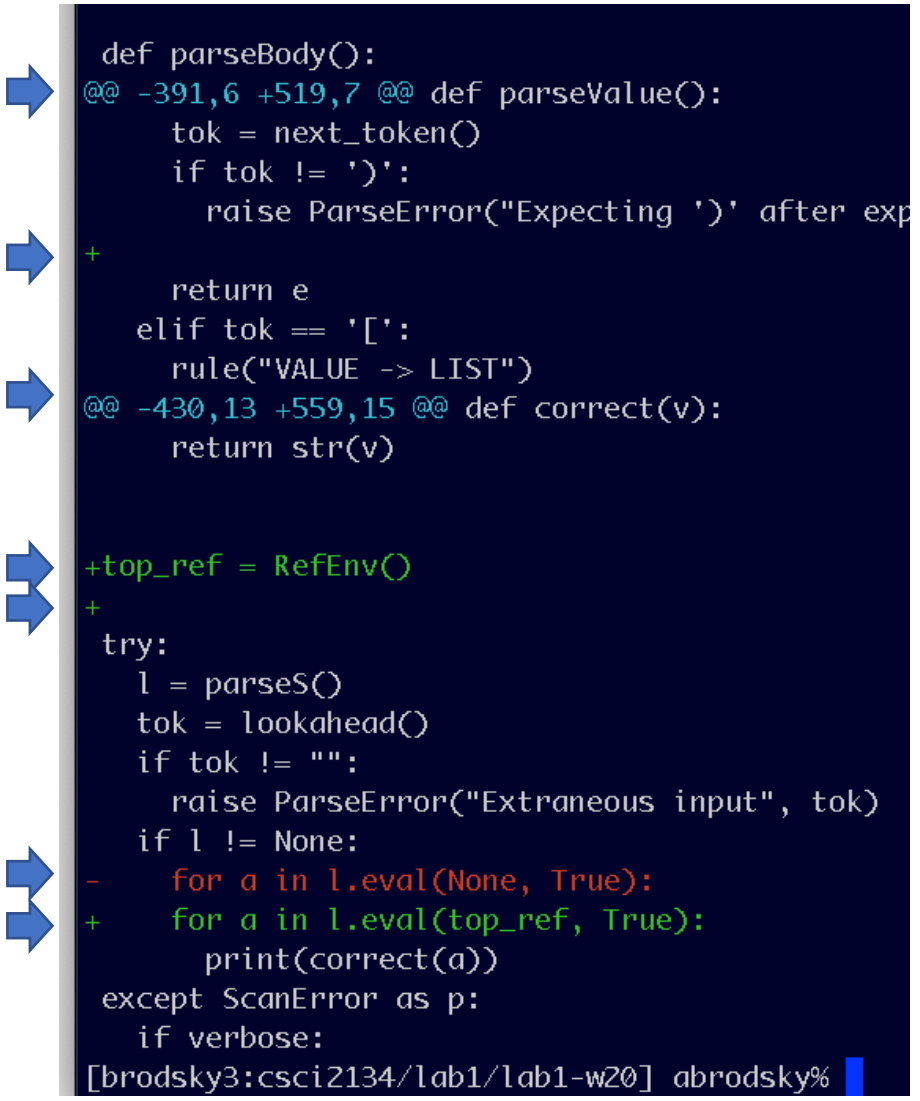




Versions and Version Numbers

- Why do files have version numbers?
- Repositories store all past versions of a file
- Every time a file is changed and committed a new version is added to the repository
- Version numbers allow us to specify precisely which version we wish to access

- Note: version control systems only store the “diff” between two consecutive versions
  - This saves space
  - Any version can be regenerated if needed



Version Control Work-Flow

<u>When starting to work on a new project</u>

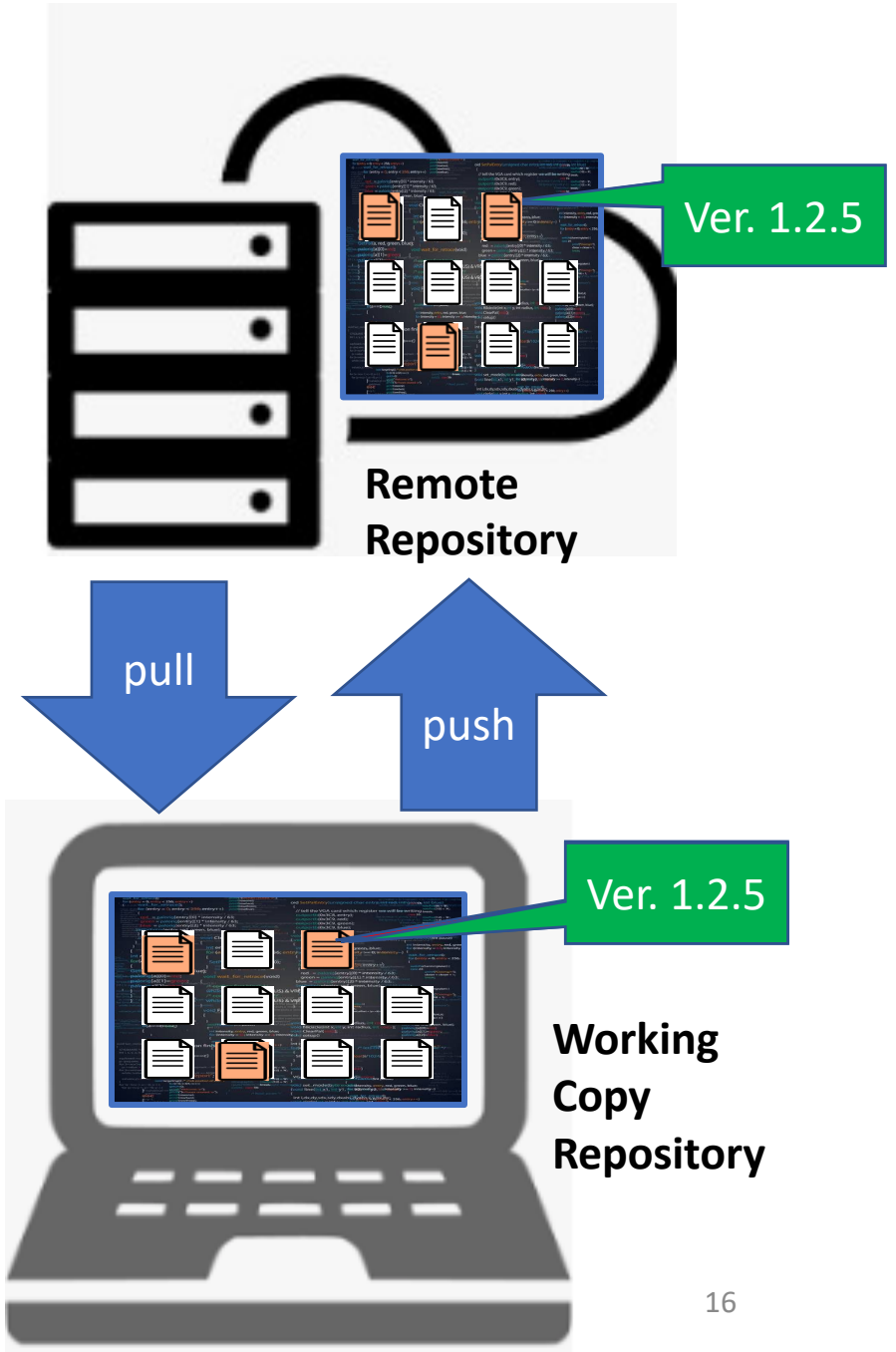
- Create a working copy repository
  This is only done once per project

<u>To make modifications</u>

- Pull down current version of files
- Repeat one or more times
  1. Modify the file(s)
  2. Commit the modifications to the working copy repository

<u>After modifications are complete</u>

- Push the changes back to the remote repository



More Jargon

```
 def parseBody():
@@ -391,6 +519,7 @@ def parseValue():
     tok = next_token()
     if tok != ')':
       raise ParseError("Expecting ')' after exp
+
     return e
   elif tok == '[':
     rule("VALUE -> LIST")
@@ -430,13 +559,15 @@ def correct(v):
     return str(v)


+top_ref = RefEnv()
+
 try:
   l = parseS()
   tok = lookahead()
   if tok != "":
     raise ParseError("Extraneous input", tok)
   if l != None:
-    for a in l.eval(None, True):
+    for a in l.eval(top_ref, True):
       print(correct(a))
 except ScanError as p:
   if verbose:
[brodsky3:csci2134/lab1/lab1-w20] abrodsky%
```

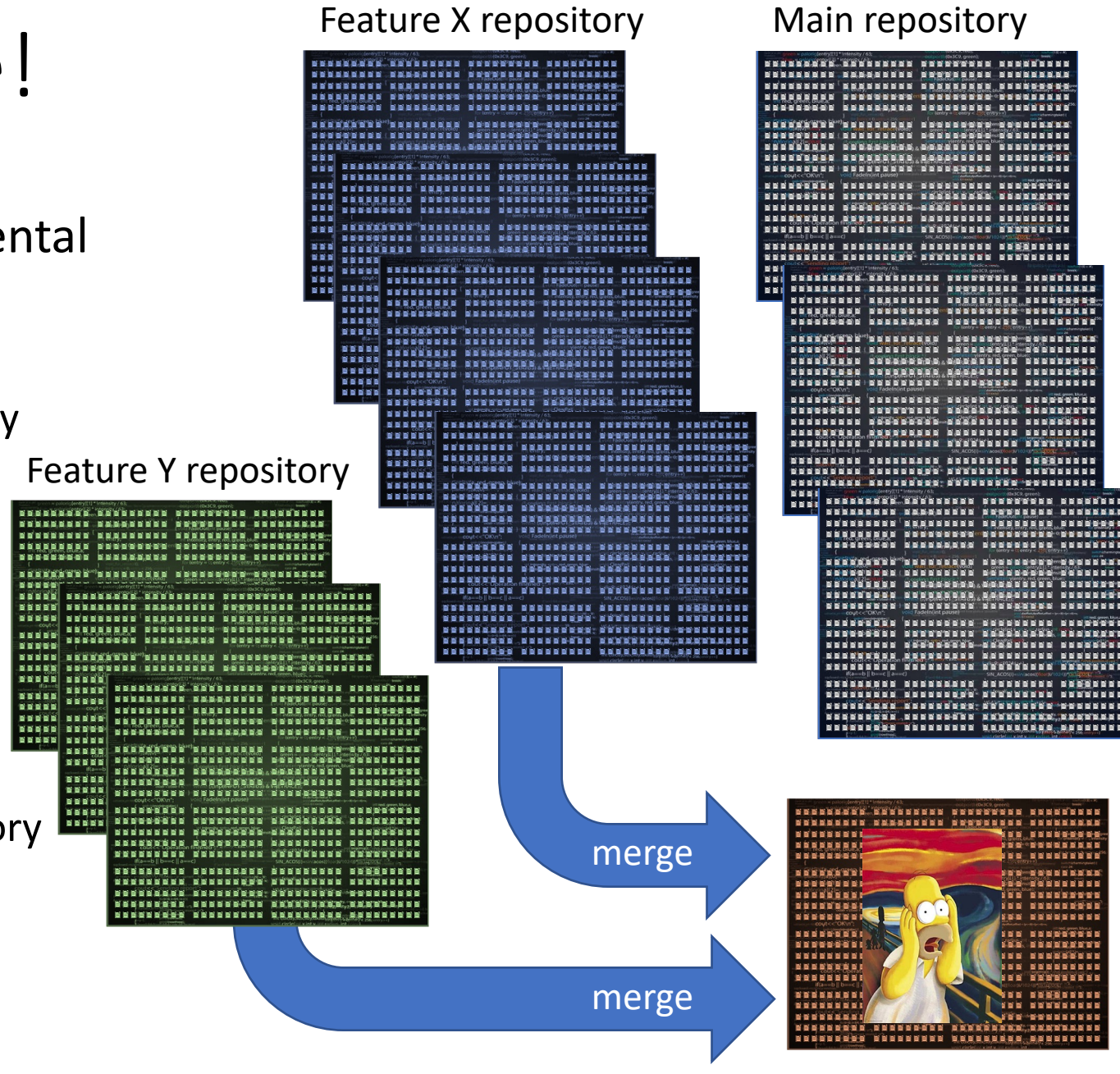
- **Commit**: Save the current modifications that you have made to your local repository (working copy)
  - Modifications are assigned a version number/identifier
  - A commit represents a checkpoint in the code development
  - A commit stores the difference between the current version and the new version, so multiple copies of a file are never stored.
  - How frequently to commits should be done is a topic of great debate
- **Push**: Upload current commits to the remote repository
  - Only modifications that are committed are pushed
  - Any developers who have access to the remote repository will be able to pull the changes
- **Pull**: Download all pushes made to the remote repository up till now
  - Any modifications that were made to the remote repository since the previous pull will be downloaded
  - Your current repository will be synchronized with the remote repository

Contents of a Repository

- Include the originating files or files than cannot be generated, e.g.,
  - Source code
  - Documentation
  - Test cases and test data
  - Scripts
  - Libraries needed to build the project (open to debate)
- Exclude files that can be generated from originating ones, e.g.,
  - Object / class files and executables
  - PDF files (when we have the original source)
  - Configuration files for individual computers (why?)

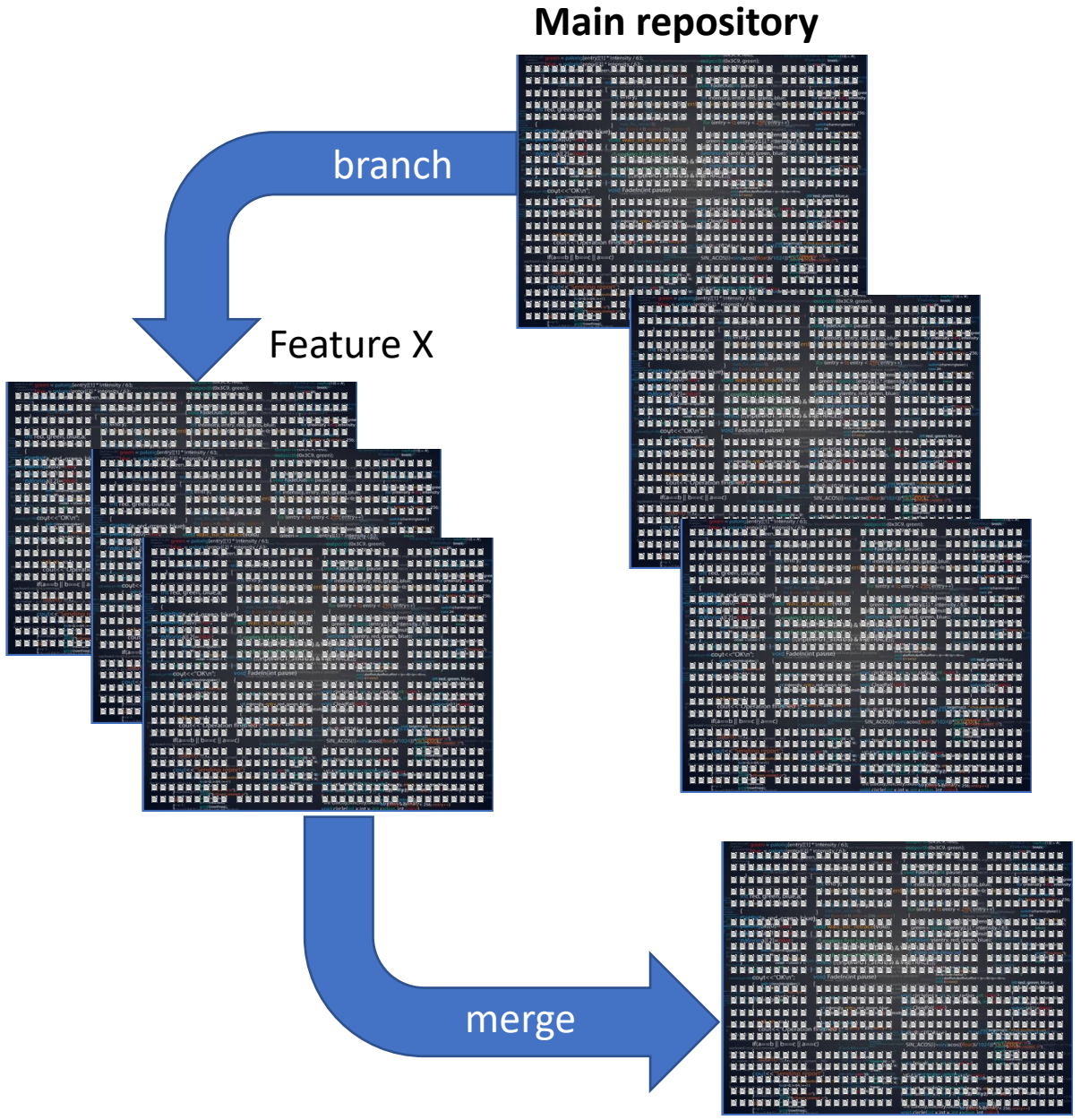
Don’t Break the Code!

- Your team is working on an experimental feature for the company’s software
- Requirements:
  - Version control with a remote repository
  - Cannot affect main/master repository
- Solution:
  - Duplicate the repository
- Issues:
  - Duplication is wasteful
  - Version history becomes disjoint
  - Merging new feature into main repository will be hard
  - More features means more duplication



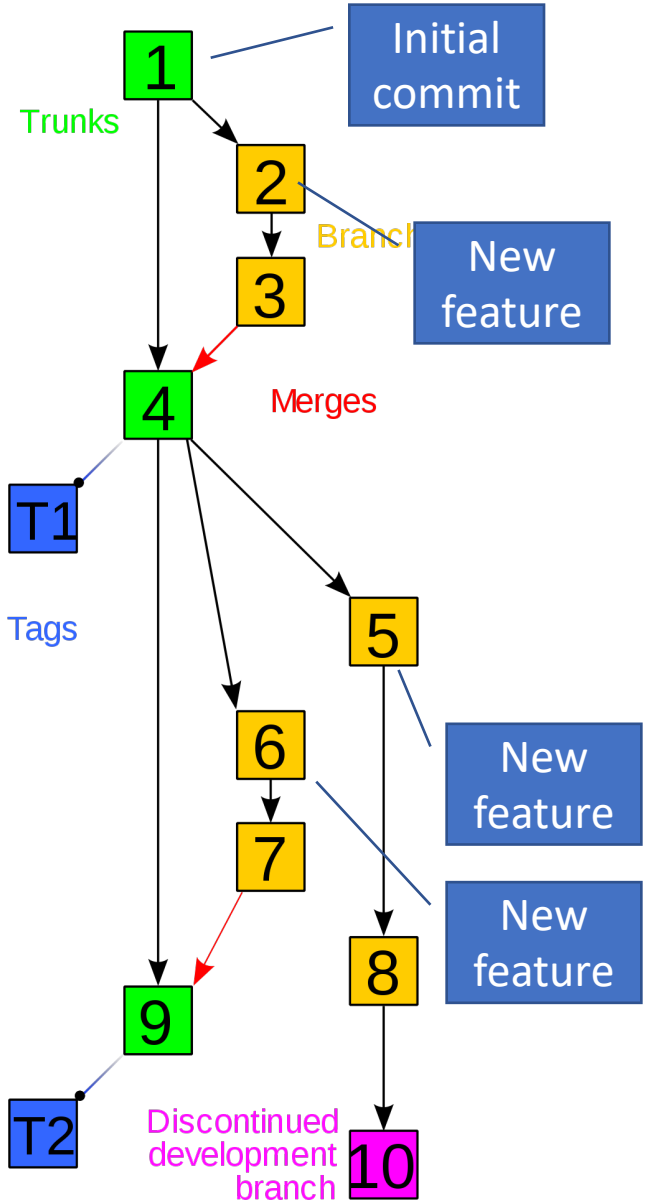
Branching Out

- Branching a repository creates a duplicate, independent code-base from the current repository
- This allows independent development on the repository without affecting the main code-base
- The version control system keeps diffs between the branch and the original



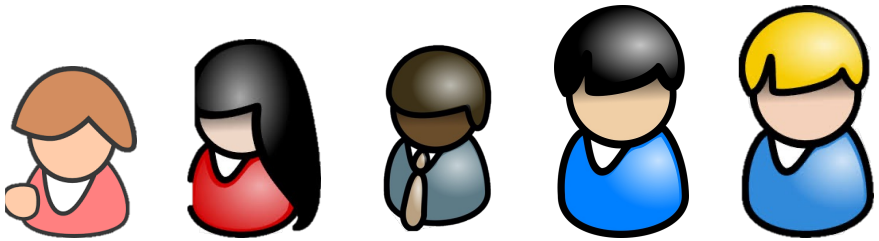
Branches

- Each repository is a tree representing the version history of the files, starting at the initial commit
- The tree has a least one branch, called the **main** or **master** branch
- A **branch** is simply a sequence of commits
- Best practice when starting a new feature on a project:
  - Branch project
  - Develop the new feature on a branch
  - Merge branch with main branch after feature is completed and tested



Multiple Developers – One File

- Suppose multiple developers need to modify the same source file
- What do we do?
- Variant 1: Exclusive Access
  - **Lock** the file(s)
  - Modify the file(s)
  - Commit, push, and unlock the file(s)
- Variant 2: Multiple Access, Merge Later
  - Modify the file(s)
  - Commit the file(s)
  - Push the file(s), manually resolve conflicts if needed
- Variant 1 is safer than variant 2
- Variant 2 is more efficient than variant 1

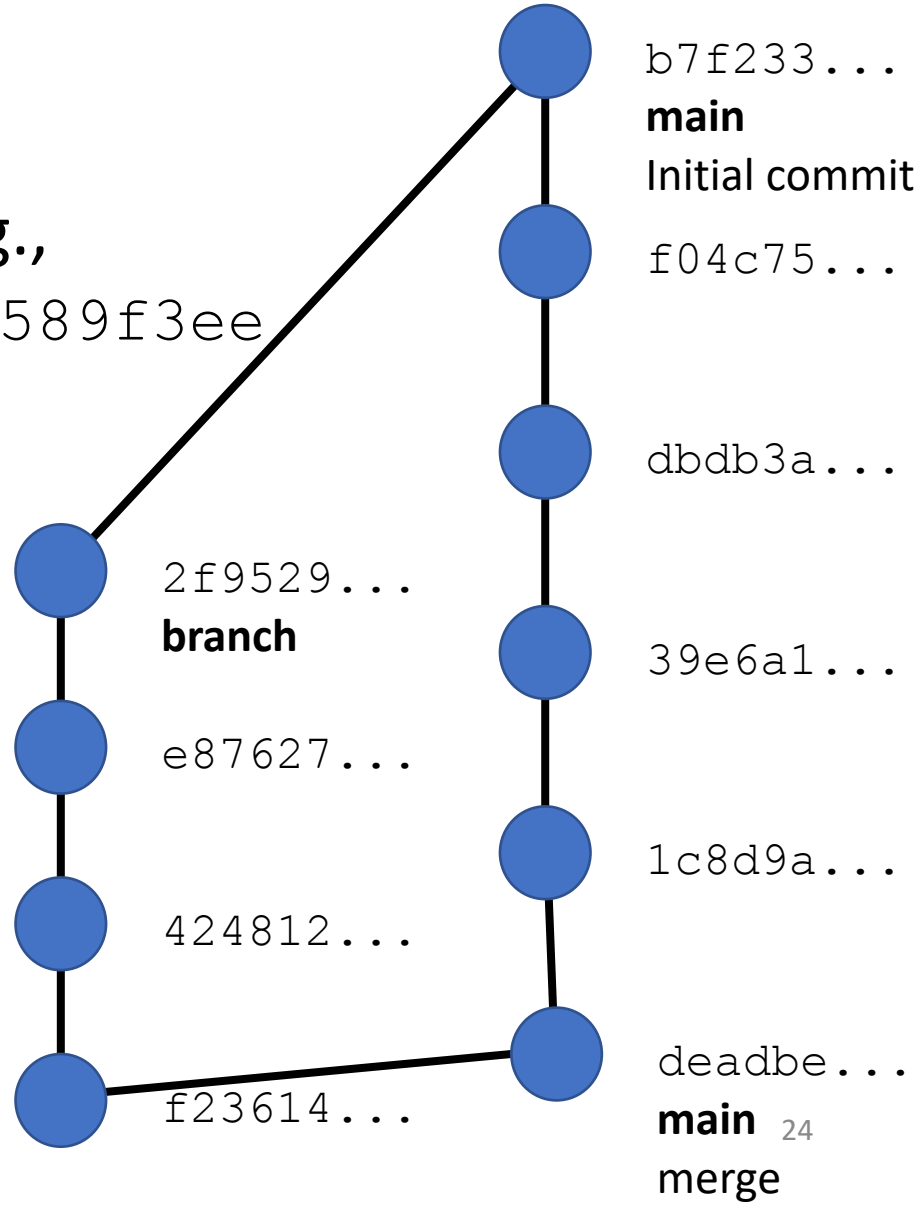


Version Control with Git

- We will be using the Git (https://www.git-scm.com) version control system
- We will be using the Faculty server (https://git.cs.dal.ca) for our course work
- Git
  - Keeps a local copy of the repository
  - Allows multiple developers to work on the same (variant 2)
  - Is free and installable on most platforms
  - Is widely used in the industry

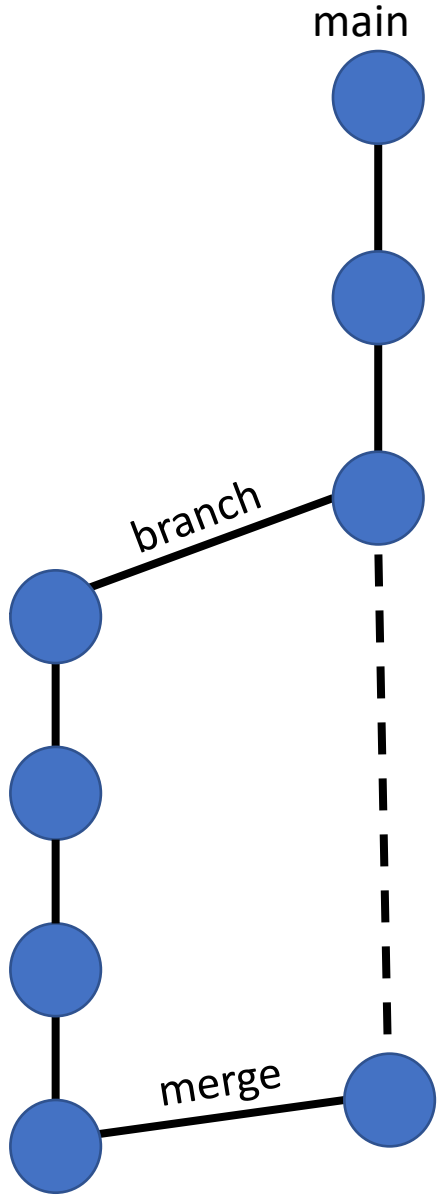
Git Mental Model

- Each version (commit) is a 20-byte hash, e.g.,
  `b7f23353d173902c59f99bf5258ee9051589f3ee`
- Primary branch: **master** or **main**
- Primary remote repository: **origin**
- Future commits are added to the end of a branch
- If a branch is successful, it may be merged back into the main branch



Git Mental Model: Process

- To start on a new or existing project
  - **New project**: Initialize a local git repository
  - **Existing project**: Clone the repository from a remote server
- To start development
  - Ensure your repository is up to date by performing a **pull**
  - Create a branch by performing **branch**
- During development
  - Modify files using appropriate tools
  - Stage the modified files (prepare to commit) by performing an **add**
  - Commit the files (creates a new commit / hash) by performing a **commit**
  - Push commits to the remote repository by performing a **push**
    - Optional
    - Useful as backup
- After development
  - Merge branch with main branch



Setting Up Your Computer to Use Git

- Be sure your CS ID is working. See https://csid.cs.dal.ca if you do not know your CS ID
- Check that you can log into the git server using your CS ID at https://git.cs.dal.ca
- You will need to open
  - Terminal on a Mac
  - Gitbash on Windows, or
  - A shell on linux
- Set your name and email

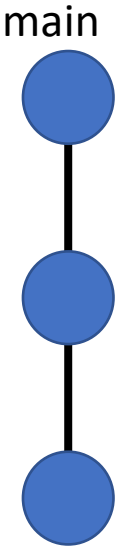
```
git config --global user.name ”Firstname Lastname"
git config --global user.email ”youremail@dal.ca"
```

- Create a **`git`** directory where all your git repositories will reside

  This is only done once the first time you start using Git

  This is a regular directory called **git** made with your file browser or a command like **mkdir**

Setting Up a Project in Git



- If you are creating a brand new project
  - Create a new project directory in the **`git`** directory, e.g. **`MyProject`**
  - Change to this directory and initialize it

    `git init`

    (use Terminal on a Mac or Gitbash on Windows)
  - Set the location of the remote repository, e.g.,

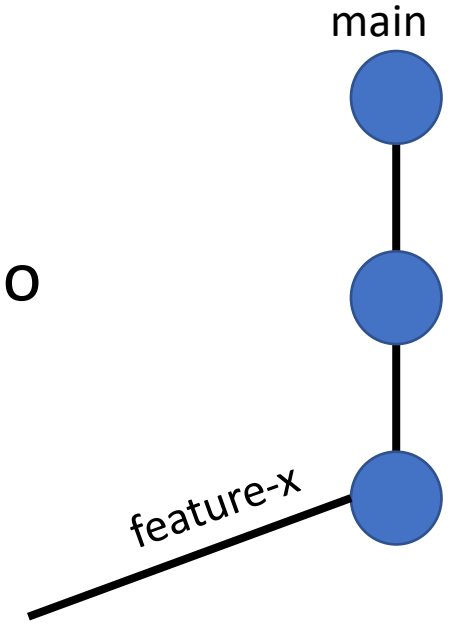
    `git remote add origin https://git.cs.dal.ca/yourcsid/MyProject.git`
  - The URL is above is called a **git slug**.
- If you are going to be working on an existing project, you need to clone it
  - Change to the **`git`** directory
  - Clone from a remote repository, e.g.,

    `git clone https://git.cs.dal.ca/yourcsid/MyProject.git`

Your CS ID

Project Name

Branching in Git



- Create a branch from the main branch on which you will do development, e.g.,

  `git branch feature-x`

  - Must be done in the project directory
- Switch to the branch

  `git checkout feature-x`
- The **`checkout`** command switches to the specified branch

Checkout **`checkout`**

- You can move from one commit to another in Git using the **`checkout`** command
- Initially, the current commit is at last commit of the **main** branch
- To move to the end of a different branch

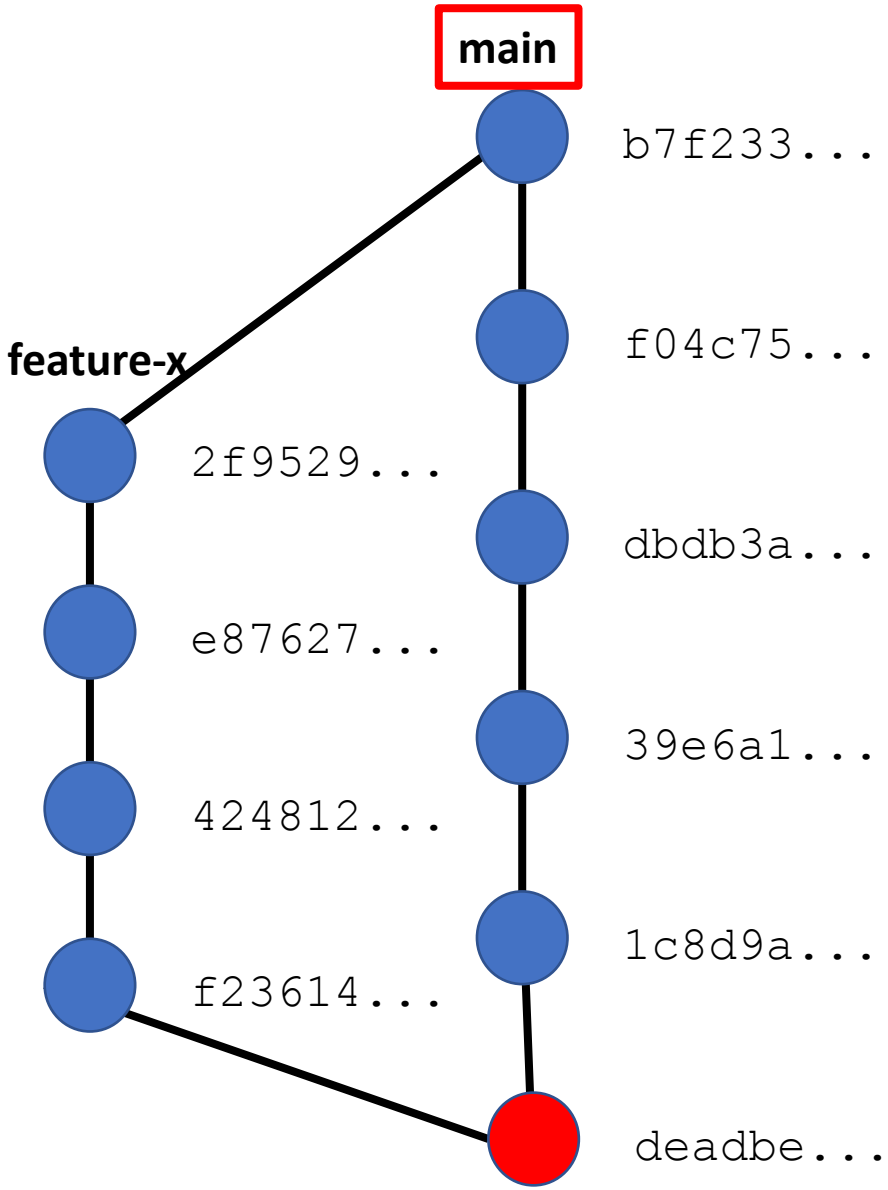
```
git checkout feature-x
```

- To move to a specific commit

```
git checkout e87627
```

- To move to the main branch

```
git checkout main
```



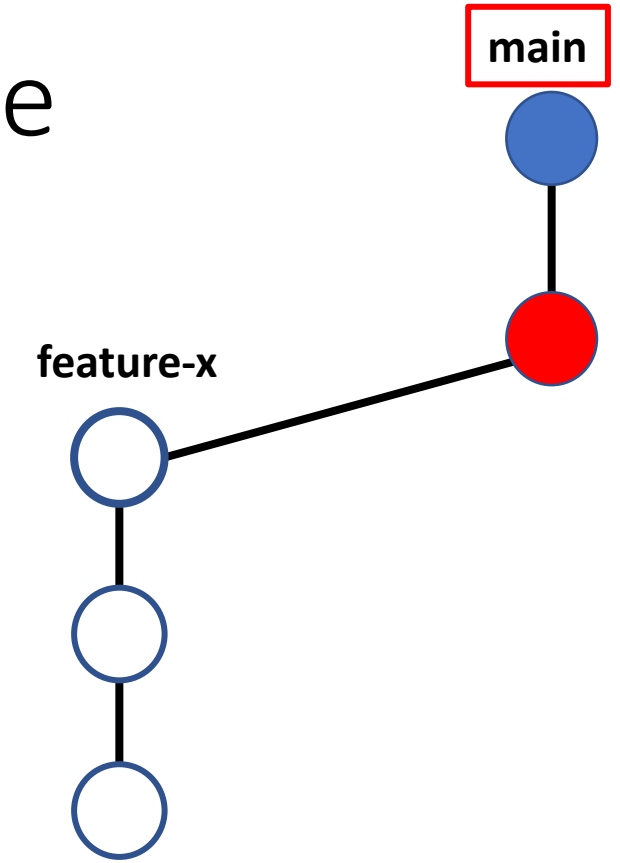
Adding (Staging) and Committing Files

- Git uses a 2-step commit process for committing files to a repository
  1. Add files to be committed (both new files and modified files)
  2. Commit files, resulting in a new commit
- To add a file to be committed use: **`git add <filename>`**
  e.g.,
  `git add Hello.java`
- A couple useful variants:
  - Add all files in specified directory: **`git add <directory>`**
  - Add all files in the project: **`git add -A`**

- Once all the files for a commit have been added, use the command
  `git commit –m "Commit message"`
  e.g.,
  `git commit –m "Created Hello World program"`

A Feature –X Development Example

```
git clone https://git.example.ca/example.git
git branch feature-x
git checkout feature-x
Modify file1 file2
git add file1 file2
git commit -m “modification 1”
Modify file2 file3
git add file2 file3
git commit -m “modification 2”
Modify file1 file2 file3
git add file1 file2 file3
git commit -m “modification 3”
git push –u origin feature-x
```



Are we done?

Merging Branches

- When development is done, the branch can be merged with the **main**
- The command to merge into current branch is:

```
git merge <from branch>
```

- To merge you must be at the head of the destination branch

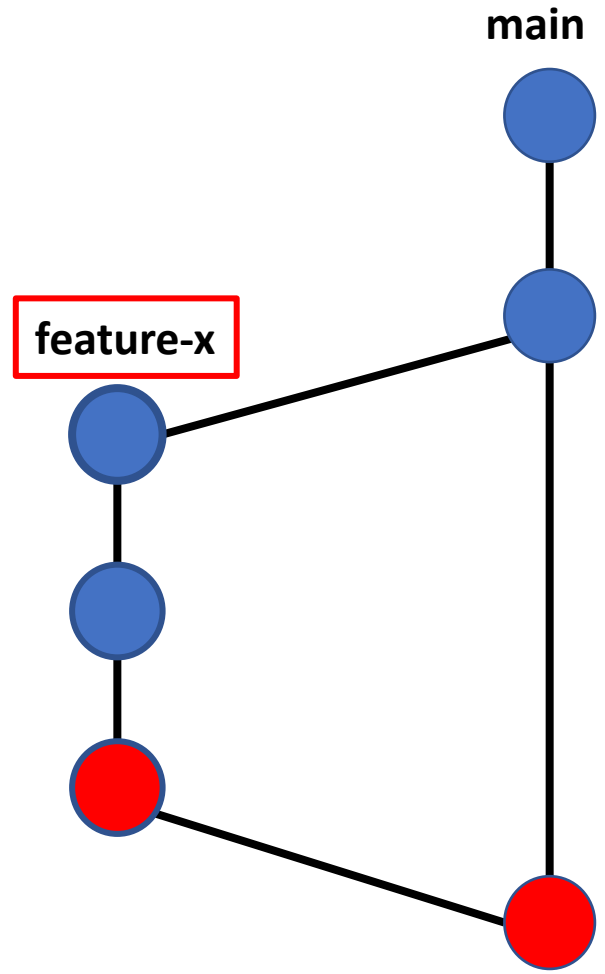
```
git checkout main
```

- And then merge from the desired branch

```
git merge feature-x
```

- Lastly push the merge up to the remote repository

```
git push origin main
```



The Final Push

- To upload the local repository commits to the remote repository use the **push** command
- To push a branch to the remote repository use:

  `git push <remote> <branch>`

  e.g.,

  `git push origin main`
- If you created a new branch that you wish to push use the –u switch to set the local branch to follow the remote branch, e.g.,

  `git push –u origin feature-x`
- To push all branches to the same remote repository use:

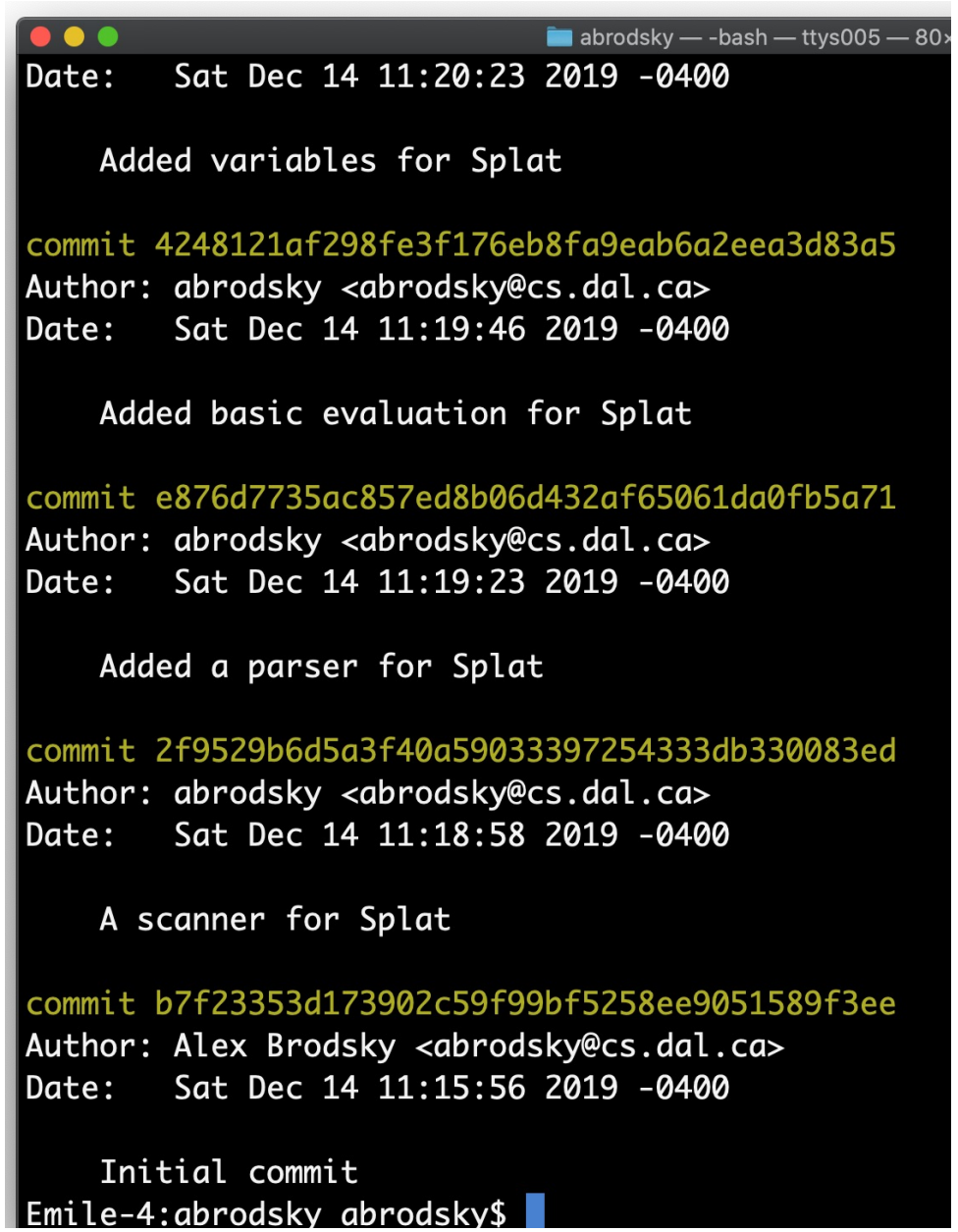
  `git push --all origin`

Log of a Branch

- The list of all commits on a branch is called a **log**
- To view the list of all commits we can use the command

  `git log`

- The log includes
  - Commit hash
  - Commit author
  - Commit date
  - Commit message
- The log gives a good overview of the development of a given branch

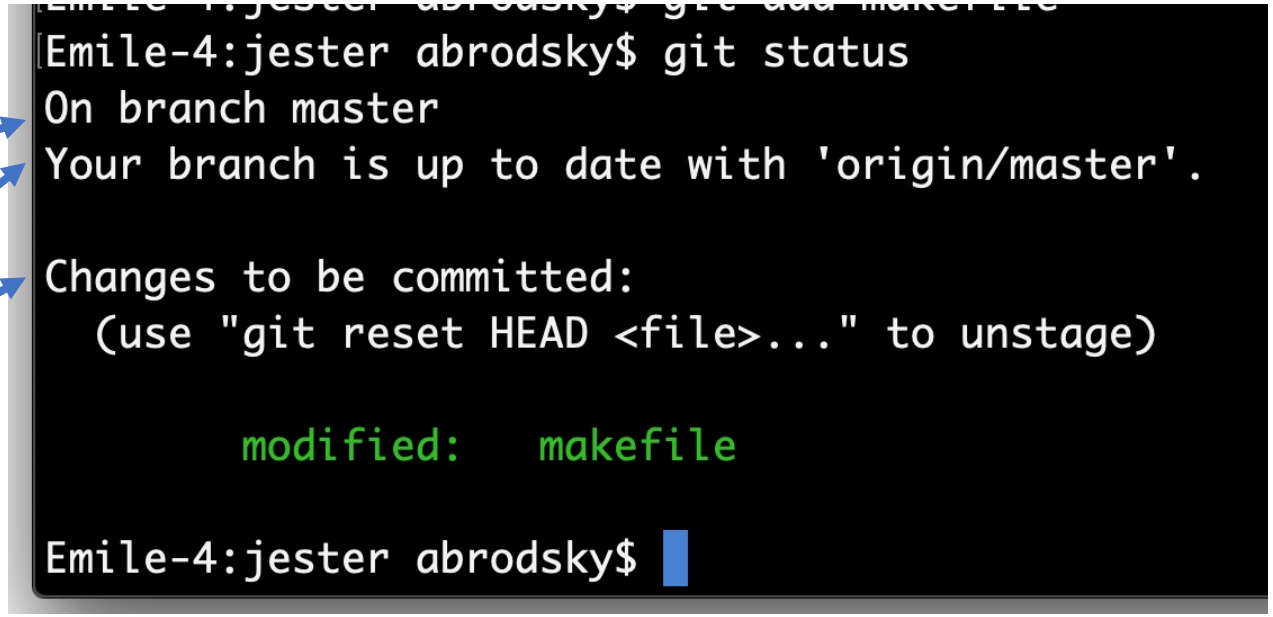


Current Status

- One of the most often used commands is the **status** command

  `git status`

- This command lets you know
  - Current branch
  - If the branch is up to date
  - Files to be committed
- This command should be used after using **checkout**, **add**, etc..



Differences Between Versions

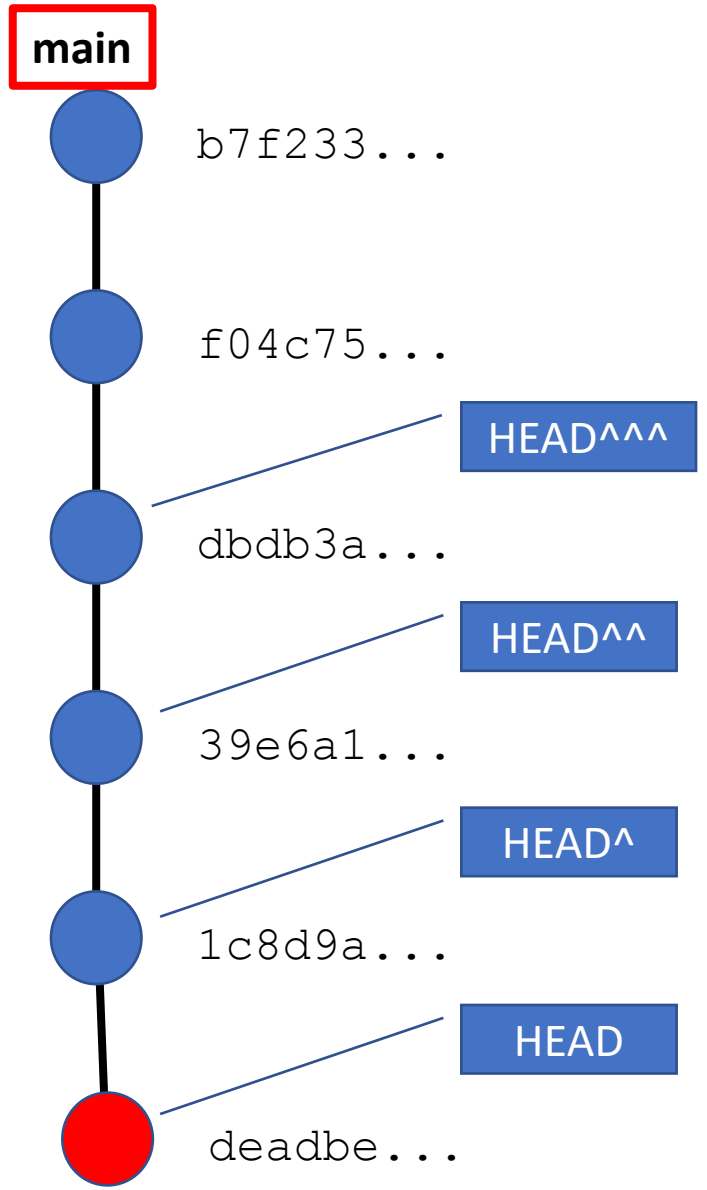
- To determine the difference between two version of a file use the **diff** command:

  `git diff <version> [version] <filename>`

- The diff displayed is similar to what is stored for each commit
- The version can be
  - A hash, e.g.,

    `git diff 80f3f 5e643 incremental/splat.py`
  - Relative to the current commit

    `git diff HEAD HEAD^ incremental/splat.py`
- The name **HEAD** refers to the current version
- **HEAD^** refers to the previous commit and **HEAD^^** refers to the commit before **HEAD^**



Removing Files

- Just like adding (staging) a new or modified file, a file needs to be explicitly removed from the repository to be deleted
- To remove a file use the command:

  `git rm <filename>`

  E.g.,

  `git rm hello.c`

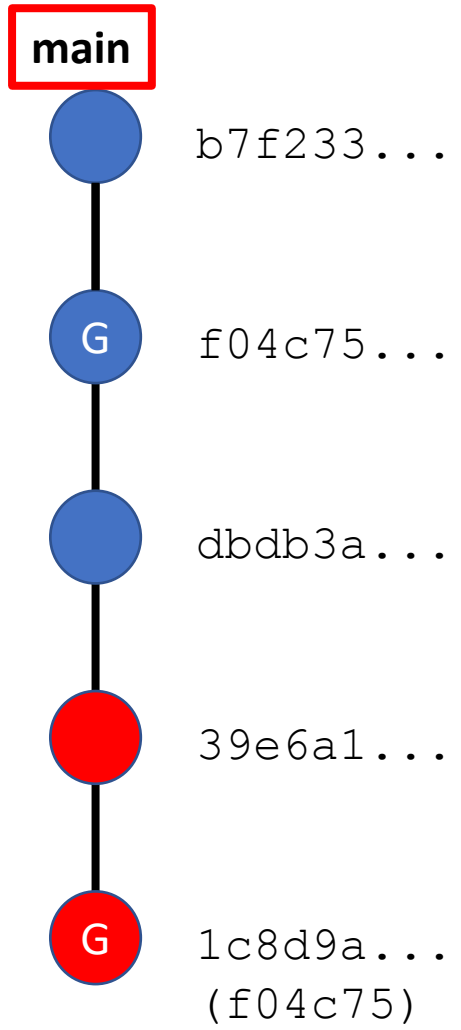
Going Back in Time

Revert

- To revert to a previous commit use the command
  `git revert <version>`
  e.g.,
  `git revert f04c75`
- This will add a new commit that is the same as the specified one
- This does not throw-away the previous commits

Reset (use with care)

- To reset to a previous commit (throw away all succeeding commits) use the command
  `git reset --HARD <version>`
  e.g.,
  `git reset --HARD f04c75`

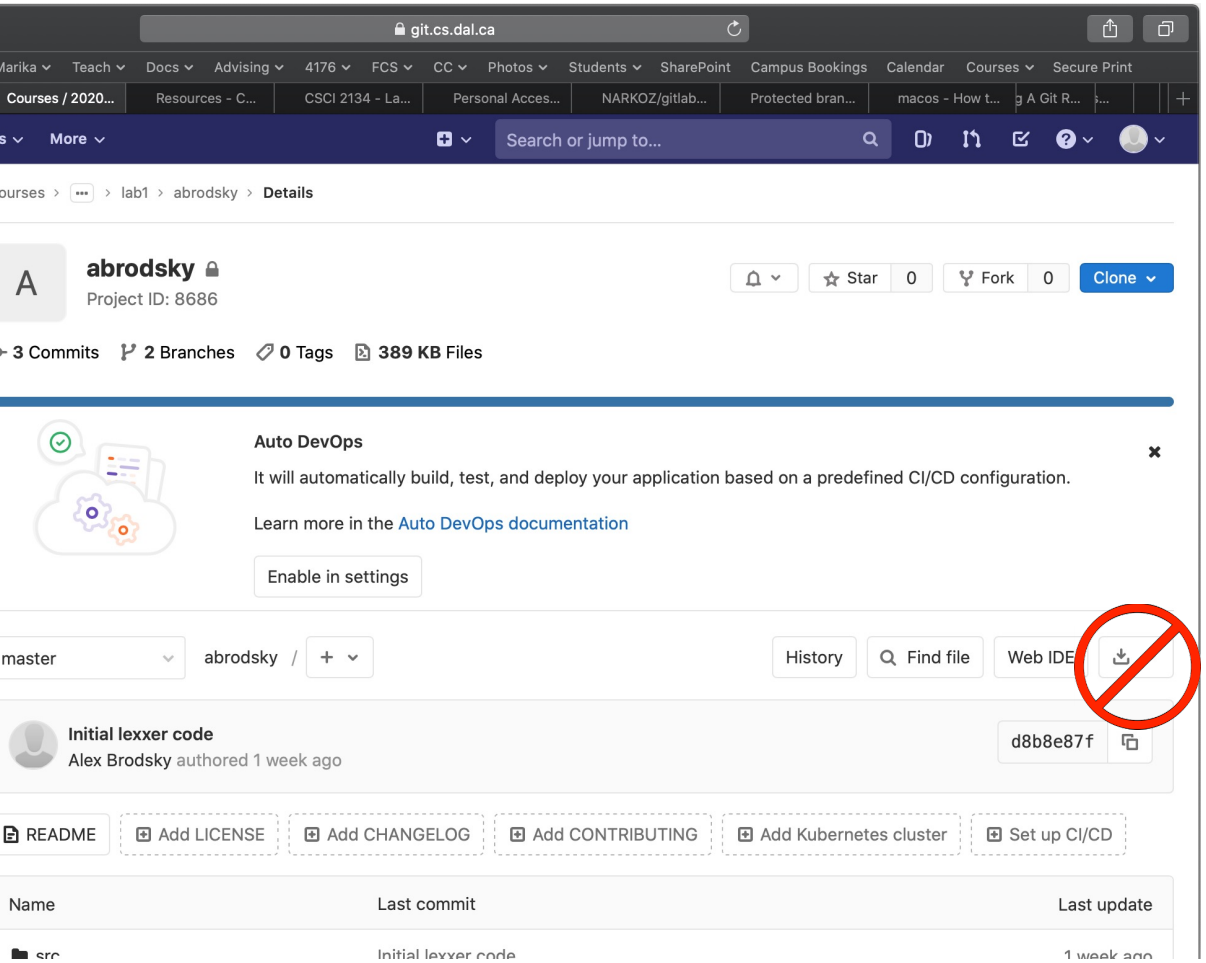


Avoid this unless you really know what you are doing

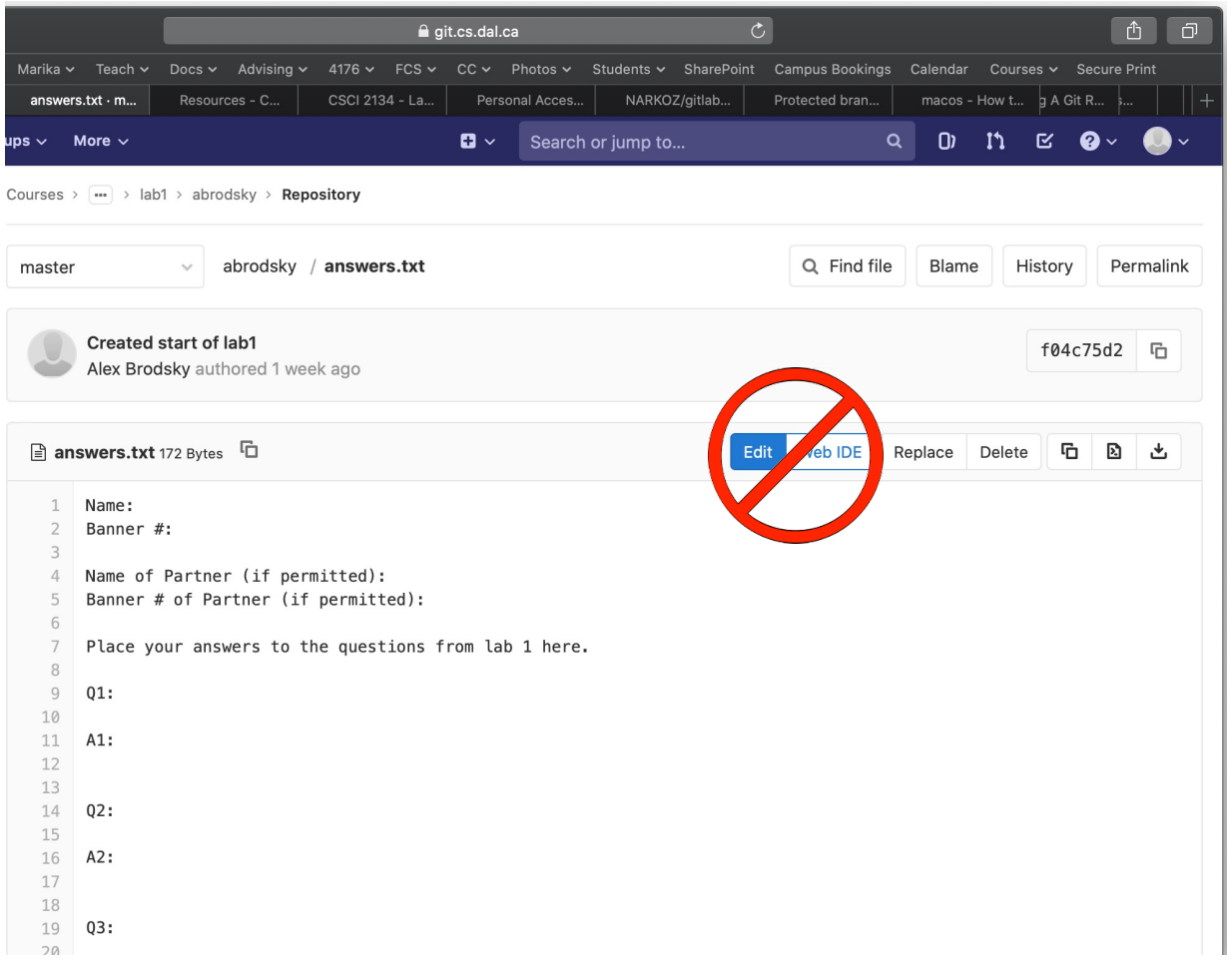
Best Practices using Git

- Store all the repositories in a single **`git`** directory on your computer
- Never download code from a git repository (use clone or fetch)
- Never edit code using the gitlab/github web interface
- Be very careful. It is very easy to mess up a repository
- Create a branch every time you start a new feature
- Commit often to create checkpoints in your development
- Use **git status** often to confirm the current branch and confirm what is being committed

Never Download, Only Clone



Never Edit in GitLab



Key 

Points

- As developers we need an assortment of different tools to help us develop software
- An integrated development environment (IDE) combines tools for editing, building, debugging, testing, and managing our source code
- A version control system is used to manage source code and is especially useful when working on large projects, with multiple versions, and multiple people
- Version control systems typically represent versions (commits) of the code in the repository as nodes in a tree, where a branch represents independent development on the same code base
- Git is a version control system that encourages developers to branch whenever they are working on a new feature and commit often

Image References

Retrieved December 19, 2019

- http://pengetouristboard.co.uk/vote-best-takeaway-se20/
- https://media.istockphoto.com/vectors/sledgehammer-to-crack-a-nut-vector-id465746244?k=6&m=465746244&s=612x612&w=0&h=jMa4sJQsKl6DMgm6NCSr-WPog5j82pDY3I3QbxmETw4=
- https://previews.123rf.com/images/artnataliia/artnataliia1801/artnataliia180100009/93384948-big-set-of-house-repair-tools-including-hammer-sledgehammer-spatula-brush-nail-screw-nut-wrench-and-.jpg
- https://cdn0.iconfinder.com/data/icons/file-names-vol-8-1/512/03-2-512.png
- https://c7.uihere.com/files/893/429/157/bmp-file-format-bitmap-others-thumb.jpg
- https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Revision_controlled_project_visualization-2010-24-02.svg/800px-Revision_controlled_project_visualization-2010-24-02.svg.png
- https://images-na.ssl-images-amazon.com/images/I/61q0wrsXejL._AC_SY741_.jpg
- https://publicdomainvectors.org/en/free-clipart/Do-Not-sign-vector-clip-art/30078.html