**Example Using Git:**

The first assignment asks students to three things...

1. **Lecture: First Day**
   - Cover the syllabus for the course. The syllabus should cover basic information about
     (a) the instructor including contact information, office location, webpage links,
     (b) general course content description,
     (c) what is expected in terms of homework,
     (d) exams and exam dates,
     (e) general USU, Department, and instructor policies, and
     (f) how assignments and exams will be graded.
   - Students must be familiar with command windows or terminals for command line interaction with Unix, Linux, or Cygwin Unix emulation to be successful in the course. We will spend some time introducing these ideas using Cygwin. Interactive Develement Environments (IDEs) will not do the work in this course. Note that for USU students, a version of Cygwin has been installed and is available in the Engineering Lab on the third floor of the building.
   - Students must choose a programming language for use in the course: It will be best to choose a computer programming language and stick with the language throughout the semester. This means you should be proficient with at least one native programming languages. Students should use one of the following:
     (a) C, You can use gcc
     (b) C++, You can use g++
     (c) Fortran, You can use gfortran
     (d) Python - Use python for Cygwin/Linux
     (e) Java - Download and use the Java SDK from Oracle.
   - Installing/working with git: Git is a Version Control System (VCS) that was first developed and used to distribute the work of building and maintaining the kernel for Unix/Linux systems. Note that git can easily be obtained as a package along with Cygwin.
   - Get on Github: We will use Github for this course. This means you will need to set up an account on Github. This is where you can share work done in the course.
   - Compiling an example code: create helloworld.exe from a text file. This will be done to emphasize how students are to complete homework problems for the material.
   - Computing Machine Precision: compute the machine $\epsilon$ in single or double precision - first ideas on sources of errors in computation. This example will be completed by students and turned in at the next course.
   - Compilation, development of reusable code, and starting a software manual on Github. The software manual is a writing project that will tie the algorithms together.
   - If time permits, examples of how to set up a laptop or tablet for the course will be presented. For example, how one can install a platform like Cygwin. Also, discussion of other options like Linux or Bash on Ubuntu for Windows. Another issue involves use of text editors like **vi** or **vim**.

2. **Lecture: Repositories, Sharing Libraries, and Test Codes**
   - Questions from the previous lecture.
   - Using "git" to complete homework assignments. Some time in this lecture will be spent going over how to use **git** to create a repository and how to build files and other features in a repository. We will go through some specific examples of git commands and how to add, delete, modify, and commit changes to a repository. Students will need to master the use of git to turn in homework.

- As an example of using git, students will compile an example code: helloworld.exe from a text file. Students will learn how to include the files and output from the file into a repository.
- The second example of using git will involve computing machine precision. How to compute the machine epsilon in any precision - first ideas on sources of errors in computation.
- Compilation, reuse of code, and starting a shared library of object modules and how to document work via a software manual.
- If time permits at the end of class, more examples of how to set up laptops and compile code will be discussed. Examples of compiling codes in C, C++, Fortran, Java, and Python, although Python does not need compiling.

3. **Lecture: Accuracy Using the Difference Quotient for the Derivative**
   - Questions from the previous lecture.
   - A mathematical example of errors in the determination of an approximate formula for the derivative of a function in one independent variable.
   - Definition of truncation error when using approximations of analytic formulas. The definition of the derivative will be reviewed and used as a means to obtain an approximation of the derivative.
   - Analysis of the difference quotient as an approximation of the first derivative using Taylor series expansions. We will briefly review Taylor series normally presented in the second semester of an engineering calculus sequence. At USU, the prerequisite course is Math 1220 Calculus II.
   - Coding up the approximation of the finite difference approximation and testing the code. Note that the code will produce a numerical value and the results will be "verified" using a known derivative. A general discussion of the computation of the error will start with this example.
   - The end of class will be used to go over any details from previous classes.

4. **Lecture: Truncation Error versus Roundoff Error**
   - Questions from the previous lecture.
   - Formally define truncation error and roundoff error related to machine precision.
   - Define the terms absolute error and relative (or percent) error. This part of the lecture will include examples involving sensitivity of errors to small changes or perturbations in the numerical computations.
   - How roundoff errors can be accumulate through computationally intensive algorithms. One example involves the approximate solution of linear systems of equations or systems of differential equations.
   - Discussion of the two sources of error in the context of a finite difference approximation from the previous lecture. Students will see how to compute both absolute error and relative error for this example.
   - Interval Analysis and the accumulation of roundoff error. Interval analysis allows a computational mathematicians to determine bounds on how errors accumulate over a large number of computations.
   - Another related topic involves selecting "stable"algorithms for solution of real problems. Stable algorithms are less sensitive to accumulation of errors.
   - Determining the efficiency of an algorithm. An effective means for determining efficiency is to count the number of Floating Point OPerations (flops) it takes to complete the algorithm.
   - Tradeoffs between accuracy of an approximation versus efficiency of the algorithm used to compute the approximation will be discussed in terms of the examples already presented to this point.
   - Ideas on practical methods to efficiently compute the number of flops needed to complete an algorithm.

5. **Lecture: Solution of Linear Systems of Equations**
   - Questions from the previous lecture.

- Gaussian elimination and implementation of row reduction operations.

6. **Lecture: Solution of Linear Systems of Equations**
   - Questions from the previous lecture.
   - Back-substitution and the solution of linear systems of equations.
   - How to measure errors in the solution of linear equations in terms of norms.
   - Standard norms in computational mathematics.
7. **Lecture: Solution of Linear Systems of Equations**
   - Questions from the previous lecture.
   - LU factorization of square matrices.
   - Rewrite of a linear system of equations using LU factorization.
   - Forward substitution for intermediate solution of linear systems.
   - LU factorization and solution of linear systems of equations.
8. **Lecture: Solution of Linear Systems of Equations**
   - Questions from the previous lecture.
   - Comparisons for Gaussian elimination and LU factorization for solving linear systems of equations.
   - Number of Floating Point Operations (flops) by introducing counters and the size of the linear system is increases.
9. **Lecture: Solution of Linear Systems of Equations**
   - Questions from the previous lecture.
   - Reusable codes and putting together shared libraries from codes that can easily be used by others.
   - Building examples of linear system of equations and using benchmarks to test codes. ORNL/nanet
10. **Lecture: Matrix Operations**
    - Questions from the previous lecture.
    - Computation of the sum of vectors.
    - Computation of inner products of vectors.
    - Computation of cross products of vectors.
    - Computation of matrix-vector products.
    - Computation of matrix-matrix products.
11. **Lecture: Stationary Iterative Method for Solving Linear Systems**
    - Questions from the previous lecture.
    - Functional Iteration for System Equations.
    - Jacobi Iteration.
    - Gauss-Seidel Iteration.
    - Analysis of these methods.
12. **Lecture: Stationary Iterative Method for Solving Linear Systems**
    - Questions from the previous lecture.
    - Gauss-Seidel Iteration.
    - Recursion versus explicit evaluation of iteration methods.
    - Analysis of Gauss-Seidel
13. **Lecture: Parallel Algorithms for Solving Linear Systems**
    - Questions from the previous lecture.

- Compiler options for optimization of a code.
- Hello world using OpenMP.
- Using OpenMp to parallelize Jacobi Iteration.

14. **Lecture: Finding roots of nonlinear functions of one variable**
    - Questions from the previous lecture.
    - The root finding problem.
    - Functional Iteration for Root Finding.
    - Convergence of Functional Iteration Via Fixed Point Theorems.
15. **Lecture: Finding roots of nonlinear functions of one variable**
    - Questions from the previous lecture.
    - The Bisection Method.
    - The Intermediate Theorem for Continuous Functions on a Closed Interval.
    - Accuracy versus Number of Iterations in the Bisection Method.
16. **Lecture: Estimating Eigenvalues of a Square Matrix**
    - Questions from the previous lecture.
    - The Rayleigh Quotient for Eigenpairs of a Square Matrix.
    - The Power Method for estimating the largest eigenvalue of a matrix.
    - Why the Power Method Works.
17. **Lecture: Inverse Iteration for the Smallest Eigenvalue**
    - Questions from the previous lecture.
    - Properties of Eigenvalues of Matrices Using Shifting
    - The Inverse Iteration Method for estimating the smallest of a matrix.
    - Using Shifting, Parallel Implementation to Search for Eigenvalues
18. **Lecture: The QR Factorization of a Matrix**
    - Questions from the previous lecture.
    - QR factorization of a matrix.
    - Gram-Schmidt Orthogonalization of Vectors.
    - Modified Gram-Schmift Orthogonalization for added stability of the computational algorithm.
19. **Lecture: The QR Factorization for Solving Systems of Linear Equations**
    - Questions from the previous lecture.
    - The matrix operations needed to solve a linear system.
20. **Lecture: The QR Factorization for Solving Least Squares Problems**
    - Questions from the previous lecture.
    - Full Column Rank Problems and the thin QR.
21. **Lecture: The QR Factorization for Eigenvalue Problems**
    - Questions from the previous lecture.
    - The QR algorithm.
    - Why the QR algorithm works.
22. **Lecture: Quantum Computing - IBMQ**
    - Questions from the previous lecture.
    - What is quantum computing?
    - Accessing IBMQ.