21. Review Sessions, Random Numbers, Merge Conflicts

CPSC 120: Introduction to Programming
Kevin A. Wortman ~ CSU Fullerton

Agenda

0. Reminders
   a. Sign-in sheet
   b. Notes Check 3 - random numbers (due Sun May 12)
   c. Final Exam (Mon May 13, Wed May 15)
   d. Lab 12 is final lab
   e. 120L Portfolio (due Fri May 10)
1. Q&A
2. Preparing for Review Sessions
3. Random Numbers
4. (time allowing) Merge Conflicts

1. Q&A

Q&A

Let’s hear your questions about...

- This week’s Lab
- Linux
- Any other issues

Reminder: write these questions in your notebook during lab

2. Preparing for Review Sessions

The Remainder of the Semester

- This week, lecture
    - Today: last slide presentation
    - Wed: review session
- This week, lab
    - Mon sections: lab 12
    - Wed sections: open lab
        - finish lab 12, portfolio, bonus activity
    - Portfolio due out of class Fri
- Finals week
    - Lecture final exam at **scheduled time**
    - No lab activities

Review Sessions

- Q & A
- Instructor will **answer questions that are asked**
- Homework: prepare **specific** questions or topics to review
  - Can be answered in 5-10 minutes each
  - Answer will actually clarify your understanding
- Example specific questions
  - “Which code goes in the .h file, and which in the .cc file?”
  - “Please review what encapsulation means”
- Questions that are hard to answer
  - “Go over classes”
  - “Teach images all over again”

Before a Review Session

- Prepare before the review session
- Read policies in syllabus and study guide
  - Time, mode, format of questions, etc.
- Scan for specific topics or questions in...
  - your notes
  - study guide list of topics
  - learncpp.com table of contents for covered material
- Need clarification on anywhere you lost points?
  - labs
  - reading quizzes
  - midterms
- **Write down** your questions

3. Random Numbers

Randomization

- **Randomization:** when program behavior involves randomness
- Recall: CPU always executes program accurately
- But the program can generate and use **random numbers**
- Applications
  - Games
  - Procedural generation
  - Hash tables *(CPSC 131 - Data Structures)*
  - Password creation *(CPSC 253 - Cybersecurity Foundations and Principles)*
  - Randomized algorithms *(CPSC 335 - Algorithm Engineering)*
  - Statistical sampling *(MATH 338 - Statistics Applied to Natural Sciences)*
  - Machine learning *(CPSC 483 - Introduction to Machine Learning)*

Pseudorandom Numbers and PRNGs

- Programs are **deterministic:** always run the same way
- Cannot create truly random numbers
  - Like dice, roulette wheel
- Pseudo- prefix: appears to be something it isn't
  - Pseudoscience: fake science
  - Pseudonym: fictitious name
- **Pseudo-Random Number Generator (PRNG):** numbers that seem random but are technically deterministic
  - Good enough for most practical purposes
- **True Random number generator (TRNG):** for security-critical applications

Algorithm: PRNG

InitializePRNG(seed_number):

1. Initialize **state** based on seed_number

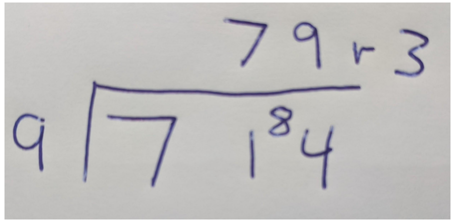
GenerateRandomNumber():

1. Use equation to calculate a pseudo-random **number** from **state**
2. Update **state**
3. Return **number**

Recap: Modulus %

- *Modulus*: remainder of long division (“mod”)
- Example:

  `714 % 9` produces 3
- Only available for integer types
  - `double` gives compile error
- Later: surprisingly, modulo is useful!



Toy PRNG

- **Not** an industrial-grade PRNG
- For demonstration purposes

```
InitializeToyPRNG(seed_number):

  1.   state = seed_number


GenerateRandomNumber():

  1.   number = (13 * state + 1) % 101
  2.   state = number
  3.   return number
```

Toy PRNG Output

```
GenerateRandomNumber():

1.    number = (13 * state + 1) % 101
2.    state = number
3.    return number
```

seed = 1

State	GenerateRandomNumber()
1	(13 * (1) + 1) % 101 = 14
14	(13 * (14) + 1) % 101 = 82
82	(13 * (82) + 1) % 101 = 57
57	(13 * (57) + 1) % 101 = 35
35	(13 * (35) + 1) % 101 = 52

seed = 120

State	GenerateRandomNumber()
120	(13 * (120) + 1) % 101 = 46
46	(13 * (46) + 1) % 101 = 94
94	(13 * (94) + 1) % 101 = 11
11	(13 * (11) + 1) % 101 = 43
43	(13 * (43) + 1) % 101 = 55

PRNGs in <random>

- std::linear_congruential_engine: similar to Toy: $x_{i+1} \leftarrow (ax_i + c) \mod m.$
  - superior choices of *a*, *c*, *m*
- std::mersenne_twister_engine: more complex, higher-quality pseudo-random numbers
  - template parameters to define output type, constants
- std::mt19937: Mersenne Twister that outputs `unsigned int`
  - No required template parameters
- **Best practice:** use Mersenne Twister

Random Number Engine API

Syntax:

*type identifier***{** *seed* **};**

Semantics:

- Declares and initializes *identifier*
- PRNG state is *seed*

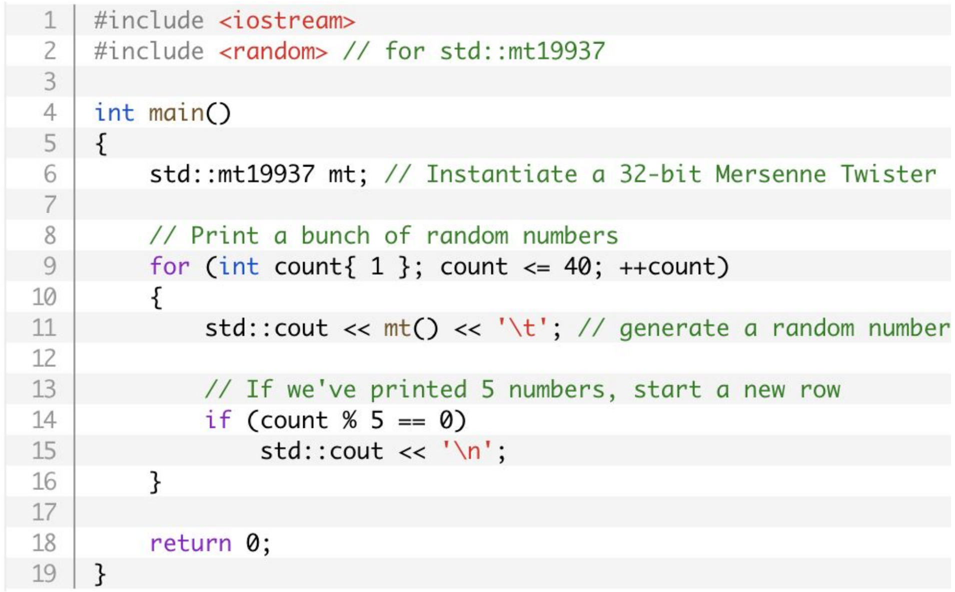
Syntax:

identifier()

Semantics:

- Function call with no arguments
- Generates and returns next random number
- Updates state

Example: Using mt19937



```cpp
#include <iostream>
#include <random> // for std::mt19937

int main()
{
    std::mt19937 mt; // Instantiate a 32-bit Mersenne Twister

    // Print a bunch of random numbers
    for (int count{ 1 }; count <= 40; ++count)
    {
        std::cout << mt() << '\t'; // generate a random number

        // If we've printed 5 numbers, start a new row
        if (count % 5 == 0)
            std::cout << '\n';
    }

    return 0;
}
```

Output:

```
3499211612      581869302       3890346734      3586334585      545404204

4161255391      3922919429      949333985       2715962298      1323567403

418932835       2350294565      1196140740      809094426       2348838239

4264392720      4112460519      4279768804      4144164697      4156218106

676943009       3117454609      4168664243      4213834039      4111000746

471852626       2084672536      3427838553      3437178460      1275731771

609397212       20544909        1811450929      483031418       3933054126

2747762695      3402504553      3772830893      4120988587      2163214728
```

Problem: PRNGs are Deterministic

- Output of previous program looks random at first glance
- Each successive number is unpredictable
- **But**, the output is identical each time the program runs!
- Unacceptable for PRNG applications
  - Game AI would make same move every time
  - Procedurally-generated image would be identical each time
- Need PRNG output to be unpredictable **from run to run**

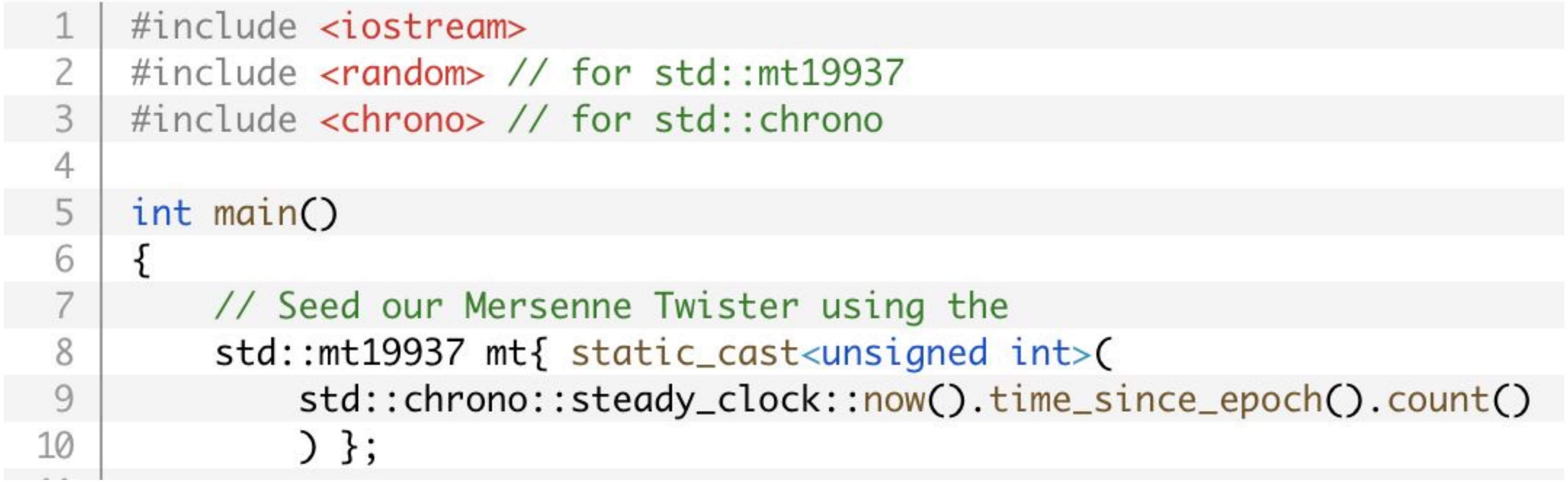
Solution: Seeding a PRNG

- Pass a **seed** argument to the random number engine constructor
- Seed should be **different each time program runs**
- Multiple approaches

Seeding with System Time

- **System time:** number of seconds since the start of the **epoch**
- Epoch (n): an era of history
- Unix epoch: midnight, January 1, 1970, Greenwich Mean Time
- Large integer
- Increments once per second
- So different each time a program runs
  - Assuming runs are at least 1 second apart

Example: Seeding mt19937



```cpp
#include <iostream>
#include <random> // for std::mt19937
#include <chrono> // for std::chrono

int main()
{
    // Seed our Mersenne Twister using the
    std::mt19937 mt{ static_cast<unsigned int>(
        std::chrono::steady_clock::now().time_since_epoch().count()
        ) };
```

Generating Random Numbers in a Range

- `mt19973` generates integers between 0 and 2,147,483,647
  - Ex.: 3499211612, 581869302, 3890346734
- Not directly useful for many applications
- More useful: generate numbers in a **range of our choosing**
- Random **die**: between 1 and 6
- Random **color component**: between 0 and 255
- Random **vector index**: between 0 and *n*-1

std::uniform_int_distribution

- std::uniform_int_distribution: generates random `int`s between *a* and *b*
- You pick *a* and *b*
- Uses a random number engine
  - Ex. `mt19973`
- **Uniform:** each outcome is equally likely
- For a fair die:
  - *a* = 1
  - *b* = 6
  - uniform: 1, 2, 3, 4, 5, 6 are equally likely

Syntax: uniform int distribution declaration

std::uniform_int_distribution *ident*{*a*, *b*};

Syntax: uniform int expression

ident(*prng*)

Semantics:

- *prng* must be a random number engine that returns ints
  - i.e. a `mt19973` object
- returns a random number between *a* and *b*

Example: std::uniform_int_distribution

```cpp
#include <iostream>
#include <random> // for std::mt19937
#include <chrono> // for std::chrono

int main()
{
    // Seed our Mersenne Twister using the
    std::mt19937 mt{ static_cast<unsigned int>(
        std::chrono::steady_clock::now().time_since_epoch().count()
        ) };

    // Create a reusable random number generator that generates uniform numbers between 1 and 6
    std::uniform_int_distribution die6{ 1, 6 }; // for C++14, use std::uniform_int_distribution<> die6{ 1, 6 };

    // Print a bunch of random numbers
    for (int count{ 1 }; count <= 40; ++count)
    {
        std::cout << die6(mt) << '\t'; // generate a roll of the die here

        // If we've printed 10 numbers, start a new row
        if (count % 10 == 0)
            std::cout << '\n';
    }

    return 0;
}
```

4. Merge Conflicts

Review: Git and GitHub

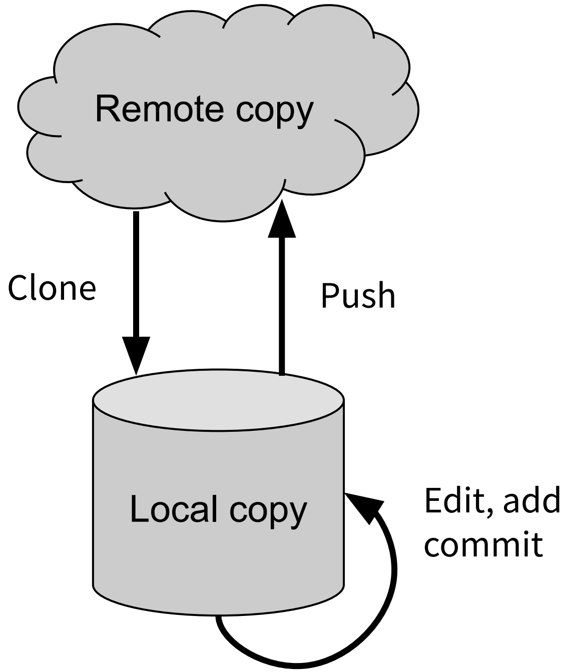
- **Source code control**: tool for programmers to share, track source code
- **git**: popular source code control shell program
- **GitHub**: cloud git service
    - facilitates sharing code with others around the world
- **Repository** (“**repo**”): holds a project
- Example: chromium, chrome_history_client.cc
- Lab 2

Review: GitHub Workflow

git understands that a repo can be copied into multiple places at the same time

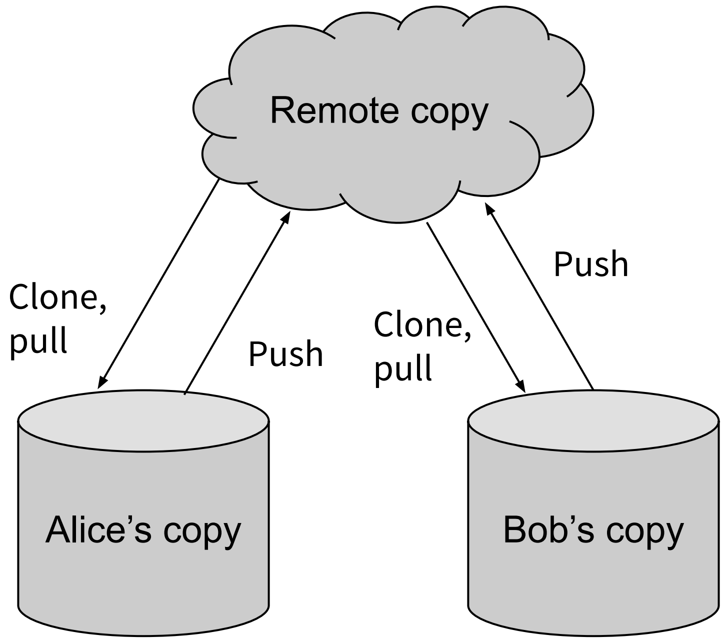
single-developer workflow:

1. Create a **remote copy** repo (lives on github.com)
2. **Clone** a **local copy** onto your computer
3. Edit, save files inside local copy
4. Create **commit(s)** summarizing changes
5. **Push** commits to **remote copy**



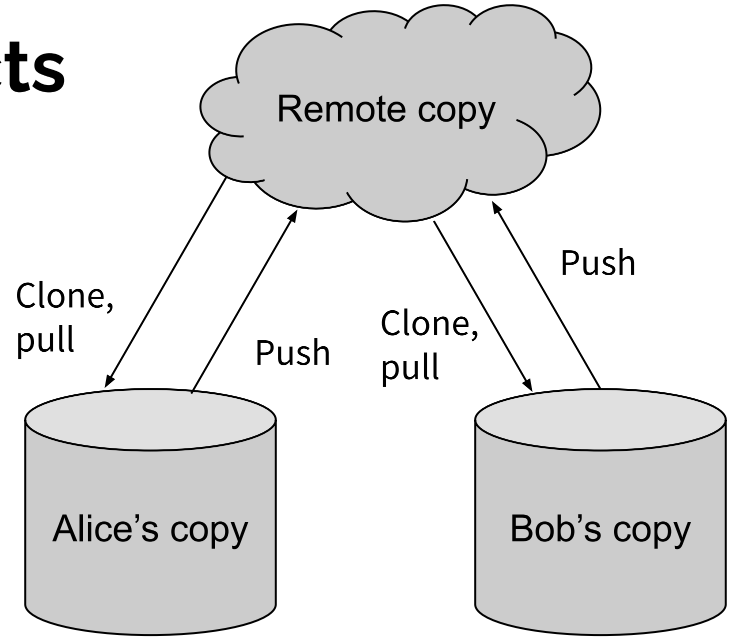
Review: Multi-Developer Sync.

- Git is intended for large teams
- **Synchronization problem:** what if...
- Alice, Bob both clone their own copies
- Alice changes main.cc
- Bob changes main.cc differently
- Alice pushes
- Bob pushes
- **Which version of main.cpp wins, Alice's or Bob's?**
  - A human must decide



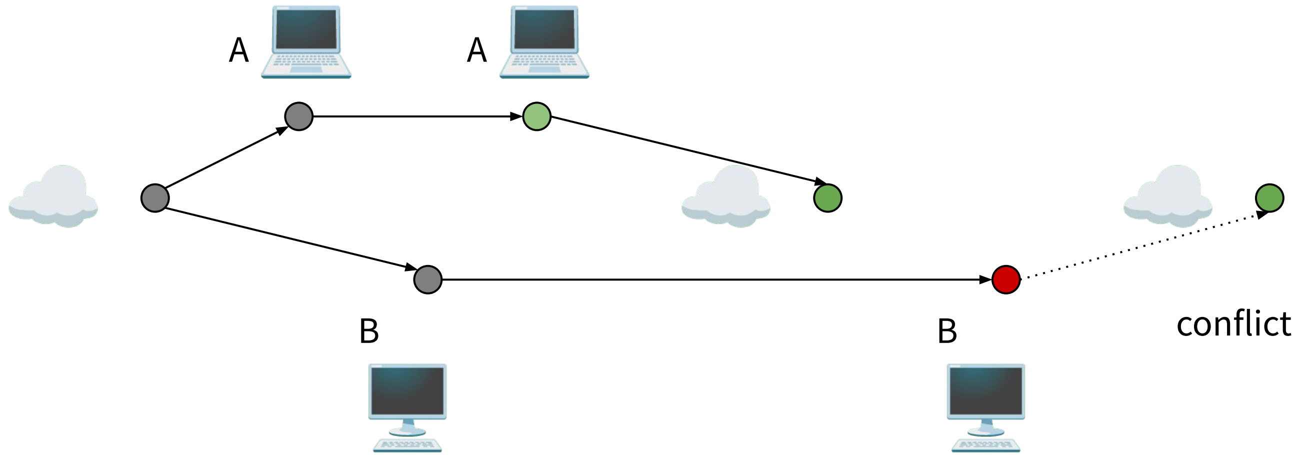
Review: Pull and Merge Conflicts

- Git rule: if your repo is behind, you have to...
- ...**pull** remote changes
- ...resolve conflicts
- ...**push**
- Suggestion: **avoid this** at first
- Use one account and computer at a time



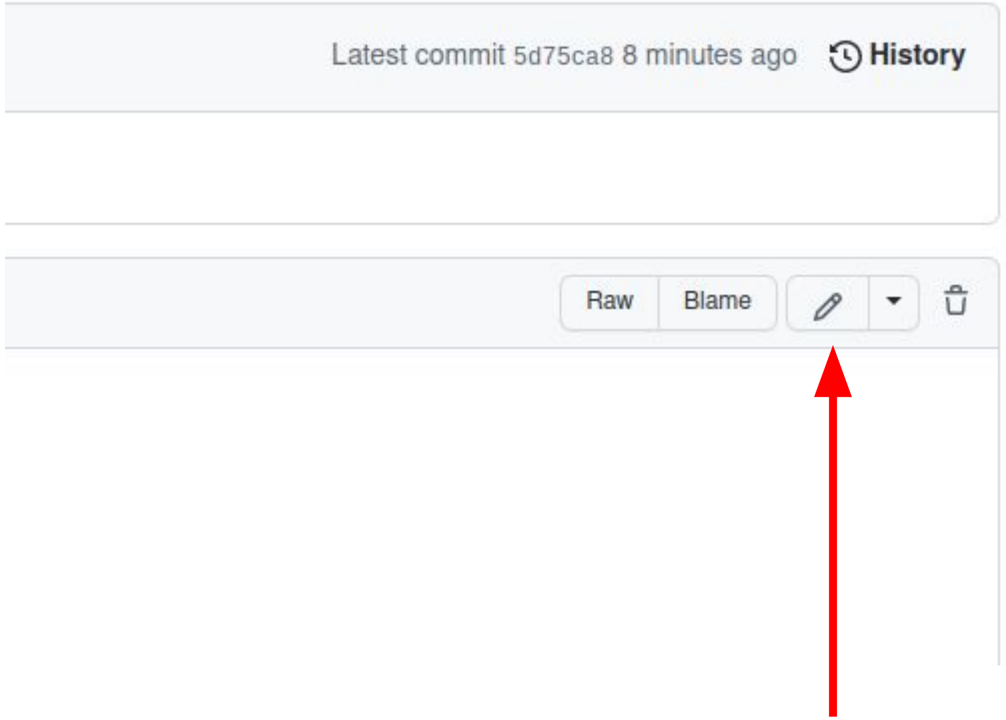
Creating Merge Conflict with Multiple Computers

1. Computer A: clone
2. Computer B: clone
3. Computer A: edit, add, commit, push
4. Computer B: edit same file, add, commit, push
   a. push fails with command error



Creating Merge Conflict with Pencil Tool

1. Computer A: clone, edit
2. github.com pencil too: edit same file
3. Return to computer A, which has conflicts



github.com is effectively Computer B in the previous slide

Alternative Solutions

At least two ways to solve this:

1. Delete the out-of-sync repo and re-clone
   a. (Computer B 🖥️ in previous slides)
   b. Bad: lose all programming work in that repo
   c. Good: simple and fast
2. Resolve conflicts manually
   a. Bad: more source control work
   b. Good: preserves all programming work

Solution 1: Delete Local Repo and Re-Clone

On computer with out-of-sync repo:

1. Identify which repo is out of sync
   a. Execute $ git status
   b. Note which file(s) are modified
2. Confirm that deleting the edits is acceptable
   a. Open modified file(s) in VS Code
   b. Decide: can we live with deleting these edits?
   c. **Next step cannot be undone**
3. Delete local repo
   a. Move outside repo with cd command
   b. Delete entire repo with `$ rm -Rf REPO-PATH`
4. Check that repo was deleted
   a. `ls` command

Solution 2: Resolve Conflicts

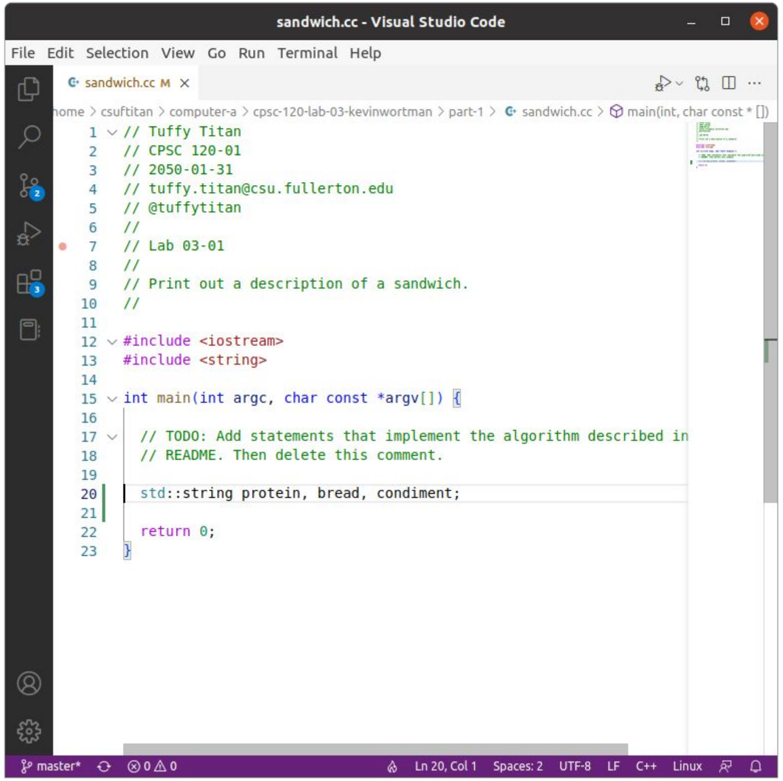
On computer with out-of-sync repo:

1. `git push` prints command error
    a. "error: failed to push some refs..."
    b. "...You may want to first integrate the remote changes hint: (e.g., 'git pull ...') before pushing again..."
    c. Follow advice ↑
2. Execute `$ git pull`
    a. No command errors: done, **stop**
    b. "Automatic merge failed": continue with steps 3-5
3. Execute `$ git status`, note which files are modified
4. Edit files and manually eliminate conflicts; for each modified file
    a. Open file in VS Code
    b. Find each conflict; color coded; marked with `<<<<<<< HEAD`
    c. Click button to keep one version
5. Add, commit, push as usual

Example: Both Computers Clone

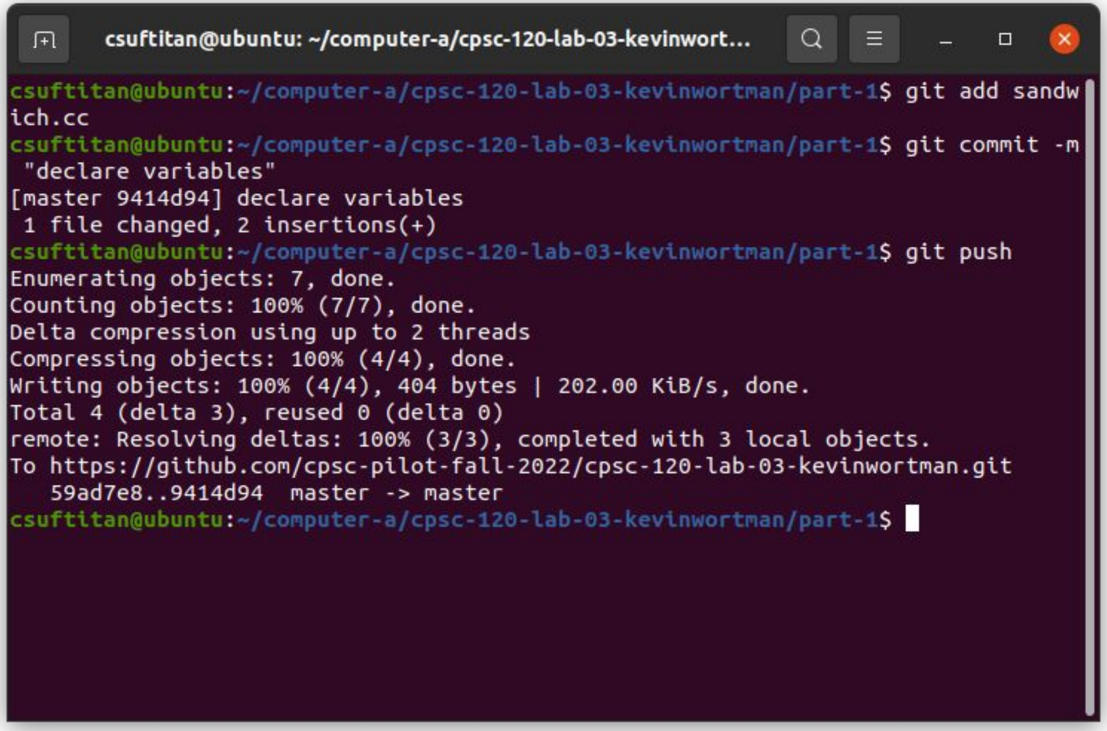
1. Computer A 💻: git clone
2. Computer B 🖥️ : git clone

Example: Computer A 💻 : edit sandwich.cc


sandwich.cc - Visual Studio Code
File Edit Selection View Go Run Terminal Help
sandwich.cc M
ome > csuftitan > computer-a > cpsc-120-lab-03-kevinwortman > part-1 > sandwich.cc > main(int, char const * [])
master* ⊗ 0 ⚠ 0 Ln 20, Col 1 Spaces: 2 UTF-8 LF C++ Linux

```
// Tuffy Titan
// CPSC 120-01
// 2050-01-31
// tuffy.titan@csu.fullerton.edu
// @tuffytitan
//
// Lab 03-01
//
// Print out a description of a sandwich.
//

#include <iostream>
#include <string>

int main(int argc, char const *argv[]) {

  // TODO: Add statements that implement the algorithm described in
  // README. Then delete this comment.

  std::string protein, bread, condiment;

  return 0;
}
```

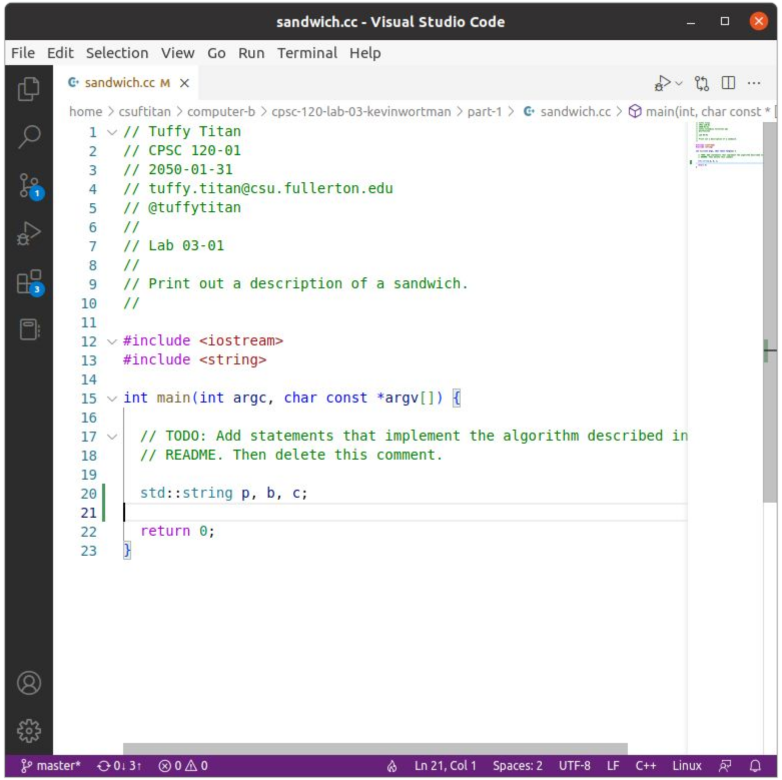
Example: Computer A 💻 : add, commit, merge



csuftitan@ubuntu: ~/computer-a/cpsc-120-lab-03-kevinwort...

```
csuftitan@ubuntu:~/computer-a/cpsc-120-lab-03-kevinwortman/part-1$ git add sandw
ich.cc
csuftitan@ubuntu:~/computer-a/cpsc-120-lab-03-kevinwortman/part-1$ git commit -m
 "declare variables"
[master 9414d94] declare variables
 1 file changed, 2 insertions(+)
csuftitan@ubuntu:~/computer-a/cpsc-120-lab-03-kevinwortman/part-1$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 404 bytes | 202.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/cpsc-pilot-fall-2022/cpsc-120-lab-03-kevinwortman.git
   59ad7e8..9414d94  master -> master
csuftitan@ubuntu:~/computer-a/cpsc-120-lab-03-kevinwortman/part-1$ 
```

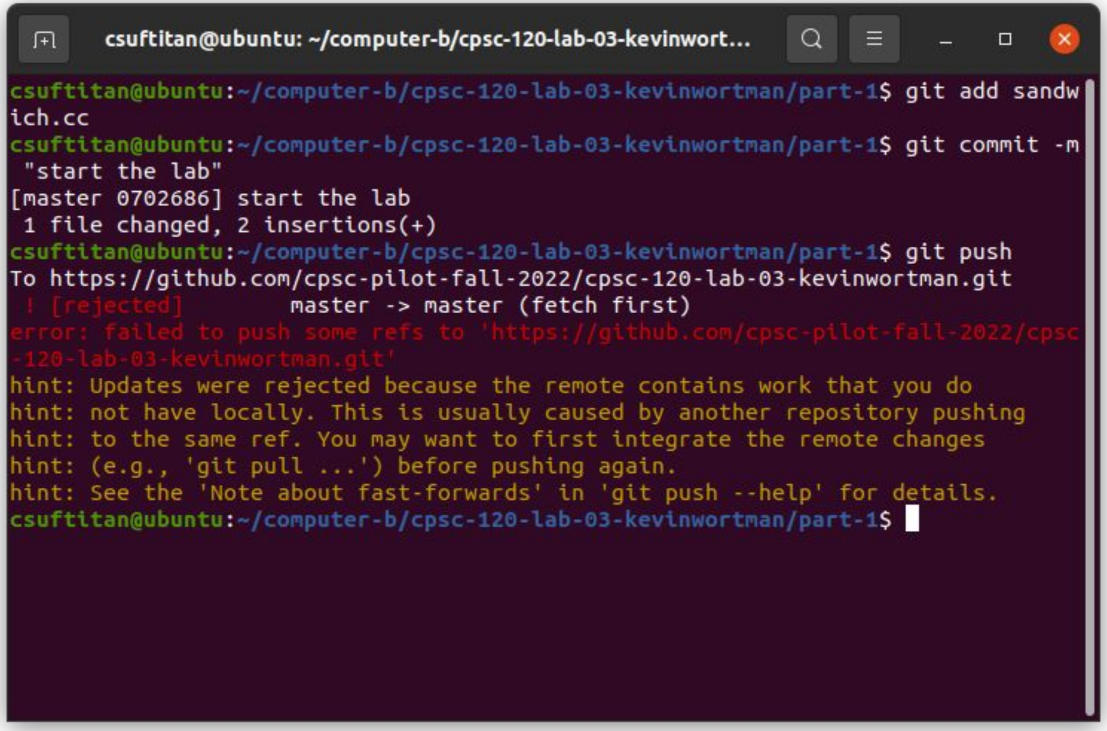
Example: Computer B 🖥️ : edit differently



sandwich.cc - Visual Studio Code
File Edit Selection View Go Run Terminal Help
sandwich.cc M
home > csuftitan > computer-b > cpsc-120-lab-03-kevinwortman > part-1 > sandwich.cc > main(int, char const *
master* 0↓ 3↑ 0 0 Ln 21, Col 1 Spaces: 2 UTF-8 LF C++ Linux

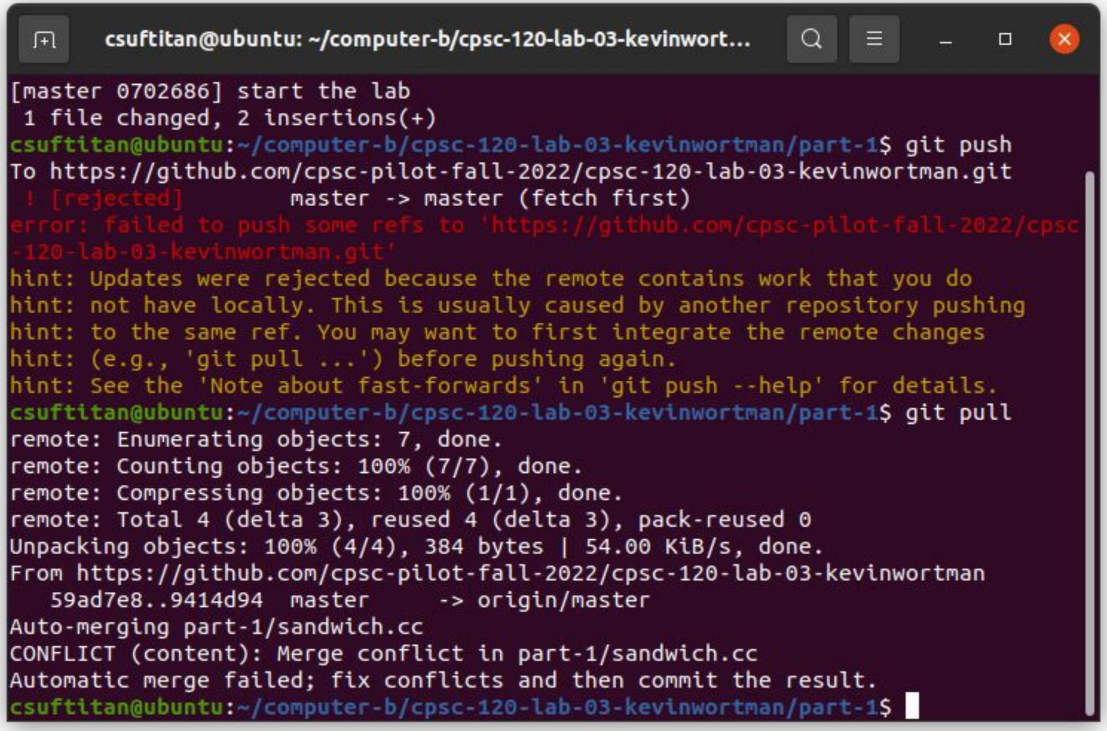
```
// Tuffy Titan
// CPSC 120-01
// 2050-01-31
// tuffy.titan@csu.fullerton.edu
// @tuffytitan
//
// Lab 03-01
//
// Print out a description of a sandwich.
//

#include <iostream>
#include <string>

int main(int argc, char const *argv[]) {

  // TODO: Add statements that implement the algorithm described in
  // README. Then delete this comment.

  std::string p, b, c;

  return 0;
}
```

Example: Computer B 🖥️ : add, commit, push, error

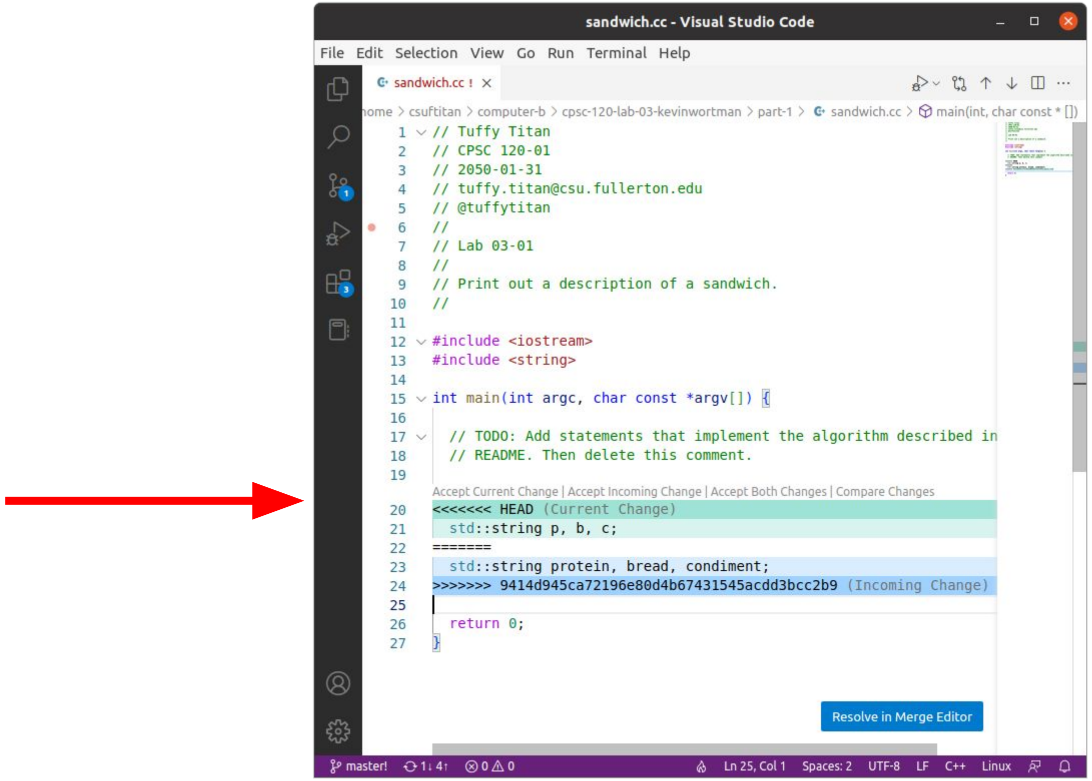


```
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git add sandw
ich.cc
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git commit -m
 "start the lab"
[master 0702686] start the lab
 1 file changed, 2 insertions(+)
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git push
To https://github.com/cpsc-pilot-fall-2022/cpsc-120-lab-03-kevinwortman.git
 ! [rejected]          master -> master (fetch first)
error: failed to push some refs to 'https://github.com/cpsc-pilot-fall-2022/cpsc
-120-lab-03-kevinwortman.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ 
```

Example: Computer B 🖥️ : `git pull`

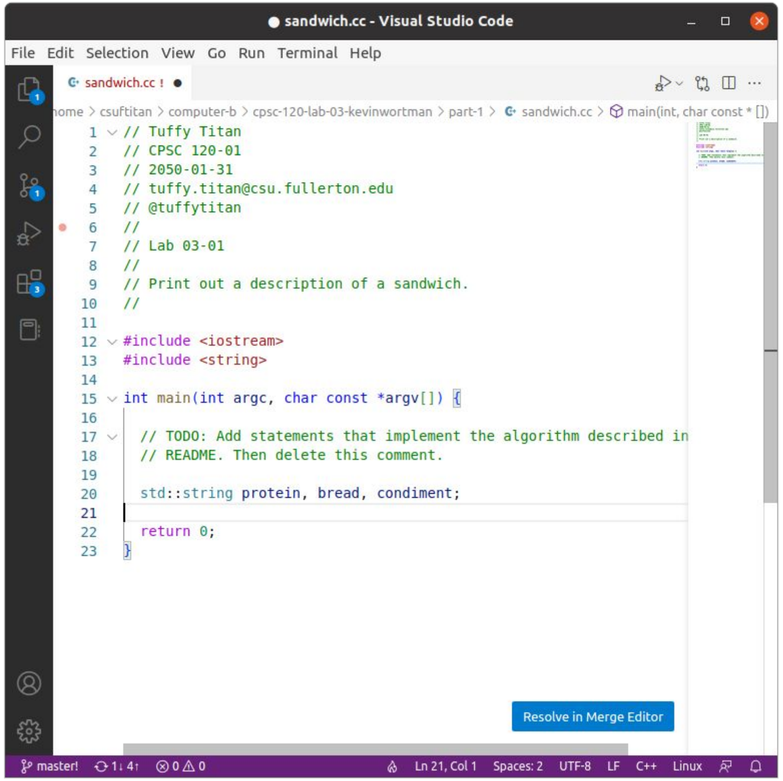

```
[master 0702686] start the lab
 1 file changed, 2 insertions(+)
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git push
To https://github.com/cpsc-pilot-fall-2022/cpsc-120-lab-03-kevinwortman.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/cpsc-pilot-fall-2022/cpsc
-120-lab-03-kevinwortman.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 4 (delta 3), reused 4 (delta 3), pack-reused 0
Unpacking objects: 100% (4/4), 384 bytes | 54.00 KiB/s, done.
From https://github.com/cpsc-pilot-fall-2022/cpsc-120-lab-03-kevinwortman
   59ad7e8..9414d94  master     -> origin/master
Auto-merging part-1/sandwich.cc
CONFLICT (content): Merge conflict in part-1/sandwich.cc
Automatic merge failed; fix conflicts and then commit the result.
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ 
```

Example: Computer B 🖥️ : VS Code merge conflict


sandwich.cc - Visual Studio Code
File Edit Selection View Go Run Terminal Help
sandwich.cc !
ome > csuftitan > computer-b > cpsc-120-lab-03-kevinwortman > part-1 > sandwich.cc > main(int, char const * [])
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
Resolve in Merge Editor
master! 1↓ 4↑ 0 0 Ln 25, Col 1 Spaces: 2 UTF-8 LF C++ Linux

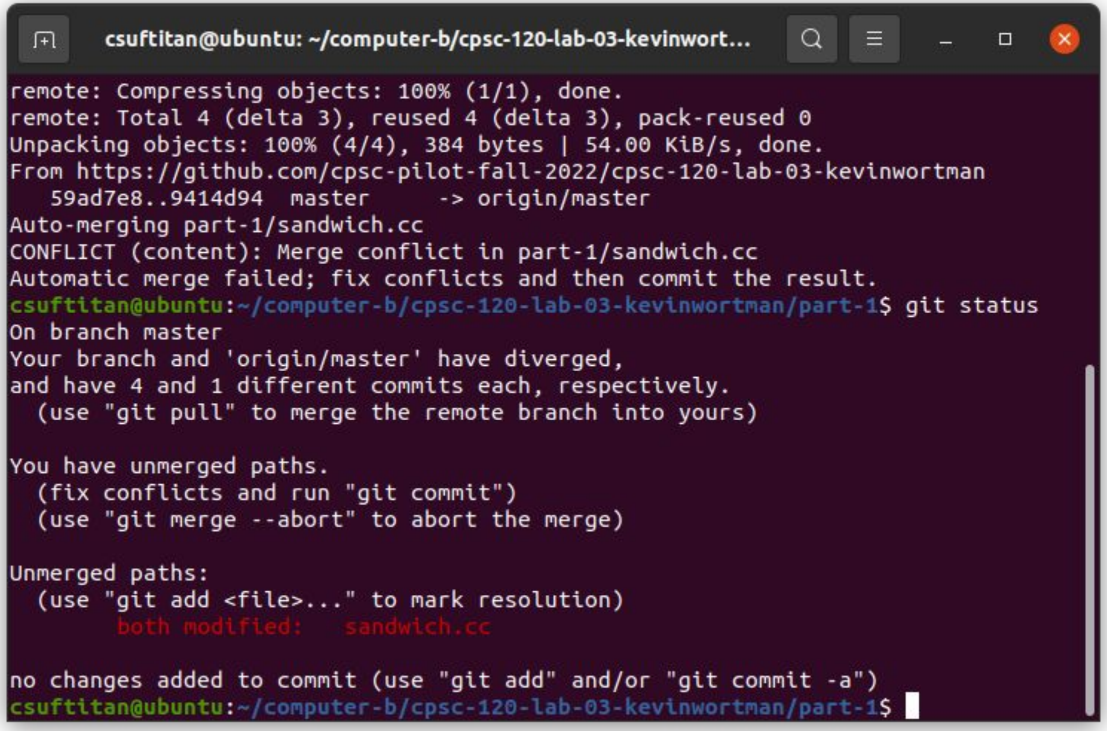
```
// Tuffy Titan
// CPSC 120-01
// 2050-01-31
// tuffy.titan@csu.fullerton.edu
// @tuffytitan
//
// Lab 03-01
//
// Print out a description of a sandwich.
//

#include <iostream>
#include <string>

int main(int argc, char const *argv[]) {

  // TODO: Add statements that implement the algorithm described in
  // README. Then delete this comment.

<<<<<<< HEAD (Current Change)
  std::string p, b, c;
=======
  std::string protein, bread, condiment;
>>>>>>> 9414d945ca72196e80d4b67431545acdd3bcc2b9 (Incoming Change)

  return 0;
}
```

Example: Computer B 🖥️ : VS Code merge conflict


sandwich.cc - Visual Studio Code
File Edit Selection View Go Run Terminal Help
sandwich.cc !
ome > csuftitan > computer-b > cpsc-120-lab-03-kevinwortman > part-1 > sandwich.cc > main(int, char const * [])
Resolve in Merge Editor
master! 1↓ 4↑ 0 0 Ln 21, Col 1 Spaces: 2 UTF-8 LF C++ Linux

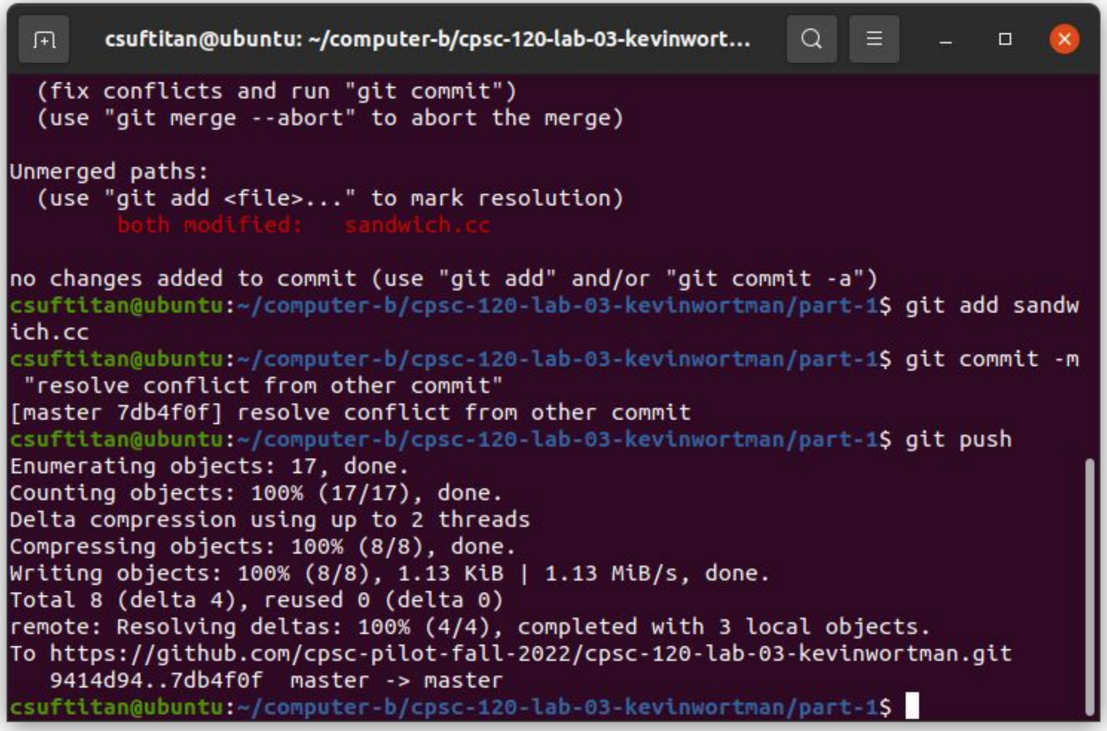
```
// Tuffy Titan
// CPSC 120-01
// 2050-01-31
// tuffy.titan@csu.fullerton.edu
// @tuffytitan
//
// Lab 03-01
//
// Print out a description of a sandwich.
//

#include <iostream>
#include <string>

int main(int argc, char const *argv[]) {

  // TODO: Add statements that implement the algorithm described in
  // README. Then delete this comment.

  std::string protein, bread, condiment;

  return 0;
}
```

Example: Computer B 🖥️ : `git status`



```
remote: Compressing objects: 100% (1/1), done.
remote: Total 4 (delta 3), reused 4 (delta 3), pack-reused 0
Unpacking objects: 100% (4/4), 384 bytes | 54.00 KiB/s, done.
From https://github.com/cpsc-pilot-fall-2022/cpsc-120-lab-03-kevinwortman
   59ad7e8..9414d94  master     -> origin/master
Auto-merging part-1/sandwich.cc
CONFLICT (content): Merge conflict in part-1/sandwich.cc
Automatic merge failed; fix conflicts and then commit the result.
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 4 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   sandwich.cc

no changes added to commit (use "git add" and/or "git commit -a")
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$
```

Example: Computer B 🖥️ : Add, Commit, Push



```
csuftitan@ubuntu: ~/computer-b/cpsc-120-lab-03-kevinwort...
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   sandwich.cc

no changes added to commit (use "git add" and/or "git commit -a")
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git add sandw
ich.cc
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git commit -m
 "resolve conflict from other commit"
[master 7db4f0f] resolve conflict from other commit
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$ git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 2 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 1.13 KiB | 1.13 MiB/s, done.
Total 8 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To https://github.com/cpsc-pilot-fall-2022/cpsc-120-lab-03-kevinwortman.git
   9414d94..7db4f0f  master -> master
csuftitan@ubuntu:~/computer-b/cpsc-120-lab-03-kevinwortman/part-1$
```

Delete or Merge?

- Delete out-of-sync repo: lose programming work (edits)
- Merge: saves work, but need to do merge work
- Choose **whichever is less of a setback**

Preventing Merge Conflicts

- “An ounce of prevention is worth a pound of cure”
  - Better to avoid merge conflicts than spend time fixing them
- Only work on one computer at a time
  - before leaving a computer: git add, commit, push
  - arriving at a different computer: git clone or git pull
- Do not use github.com pencil tool