20875 Software Engineering

Tutorial 4

First git project

1. If not already done, configure the name and email to be used in your commits. Configure your preferred editor to be used by git as well.
2. Create a directory called "`test_project`".
3. Inside the directory "`test_project`", create a git repository.
4. Add a file called "`main.c`" containing:

   ```
   #include <stdio.h>

   int main()
   {
           printf("Hello, world!\n");
           return 0;
   }
   ```

   and a "`Makefile`" (make sure to use TAB characters for indentation):

   ```
   main: main.o
           clang -Wall -O3 -o $(@) $(^)

   main.o: main.c
           clang -Wall -O3 -c -o $(@) $(<)
   ```

5. Build the executable "`main`". Create a "`.gitignore`" file that ignores that executable and any file ending in "`.o`".
6. Stage the source files ("`main.c`", "`Makefile`", "`.gitignore`") for a commit. Check your staged area with "`git status`".
7. Create the first commit for "`test_project`".

Second git project

You and the rest of your team (Alice, Bob, Carol, Dan, Eve) decide to write an interpreter for a little language aimed at describing Boolean formulas and printing their truth tables.

1. Alice creates a git repository, then writes a tokenizer and a parser for the language. Subsequently, Alice, leaves the project. Clone Alice's repository `https://www.poirrier.ca/git/alice.git` and read her code.

2. Bob emails you. On top of Alice's parser, he implemented the `eval()` and `run()` methods, which allow us to print the appropriate truth tables. For the input file `ag24_15.txt`, Bob's code takes more than 4 hours. He asks you to fetch the commit `a97f91` in his repository `https://www.poirrier.ca/git/bob.git`. Review the code introduced by Bob.

3. After some discussion with Carol and Dan, the team concludes that Bob's code could be made faster by using short-circuit evaluation for the operators `and` and `or`. Carol and Dan each set out to propose their own implementation of short-circuit evaluation. Carol finishes first. She created a branch `carol_branch` for her code. She reports that `ag24_15.txt` now takes 46 seconds. Fetch `carol_branch` from her repository `https://www.poirrier.ca/git/carol.git` into a local branch. For simplicity, we will call this local branch `carol_branch` as well. Switch to it, review Carol's code. Then, merge that code into your `main` branch.

4. You realize that there is a small problem with the code's output. Bob decided to print a header above tables for clarity, but forgot that this header must be preceded by a "#" sign in order to conform to the output specification. Fix the issue, then create a commit for this code change, still on the `main` branch.

5. Eve achieved massive performance improvement on top of Carol's code. First, she implemented constant propagation (e.g., changing `(x and True)` into `(x)`, or changing `(x or True)` into `(True)`). Then, she implemented *probing*. The idea of probing is to first arbitrarily fix the value of a variable (e.g., `y = True`), then perform constant propagation. If an expression becomes the constant `False`, then we know that we cannot have `y = True` in a `True` row of the truth table for that expression. Therefore, we can fix `y = False` for any `show_ones` for that expression. With this improvement, `ag24_15.txt` now runs in 0.135 second. Fetch Eve's code on her branch `carol_branch` at `https://www.poirrier.ca/git/eve.git`. Review Eve's contribution. Rebase it on top of your commit on the `main` branch, then merge it.

6. Dan finally implemented his approach to short-circuit evaluation. Contrary to Carol's use of `any()` and `all()` with generator expressions, Dan simply used a loop. Because of how CPython is implemented, his implementation is faster: 30 seconds compared to Carol's 45 seconds. Fetch Dan's `dan_branch` at `https://www.poirrier.ca/git/dan.git`. Merge it onto the `main` branch, combining Dan's short-circuit code with Eve's probing and your bugfix. Resolve any conflicts that result.