20875 Software Engineering

Tutorial 5

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 a Python program that prints the truth table of a Boolean formula in CNF.

1. Alice starts the project by writing a function that parses command-line arguments. Furthermore, she starts the design of a Python class "`Formula`", although the code inside its methods is not written yet. She envisioned three methods: `parse()`, `truth_table()`, `preprocess()`. Clone Alice's repository `https://www.poirrier.ca/courses/softeng/ex/alice.git` and read her code.

2. Bob emails you. He implemented a CNF file parser in the method `parse()`. He asks you to fetch the commit `5ef767` in his repository `https://www.poirrier.ca/courses/softeng/ex/bob.git`. Review the code introduced by Bob.

3. Carol creates a branch `carol_branch` in which she implements printing the truth table (the method `print_table()`). She reports that it takes 50 seconds to print the ones in the truth table of `ag24_15.cnf`. Fetch her branch `carol_branch` from her repository `https://www.poirrier.ca/courses/softeng/ex/carol.git` into a local branch. For simplicity, we will call this local branch `carol_branch` as well. Switch to `carol_branch`, review Carol's code.

4. You realize that, for a given x, as soon as one clause has value False, we know the value of the formula (it is False). Therefore, it would be advantageous to first check the clauses that are most likely False. Since each clause is a disjunction ("or"), the more literals it has, the more likely it is to be True. Therefore, you decide to sort the clauses by increasing number of literals (in the "`preprocess()`" method). Now the ones of `ag24_15.cnf` take 3 seconds. Create a commit with your improvement, still on `carol_branch`.

5. Dan has a similar idea. He thinks that if a clause was False for a previous value of x, it has higher chances to be false as well for a subsequent value of x. He implements code that remembers the last False clause, and subsequently checks it first, before the others. He reports that on `ag24_15.cnf`, printing the ones takes 1.8 seconds. Fetch his branch, `dan_branch`, which was based on Carol's code at `https://www.poirrier.ca/courses/softeng/ex/dan.git`. Inspect Dan's code.

6. You want to combine your approach with Dan's. Rebase your commit on Dan's branch. Still for `ag24_15.cnf`, the code now needs 1.5 seconds. Now, merge this new combined code into `carol_branch`.

7. At the same time you and Dan were working on processing fewer clauses, Eve worked on processing clauses faster. She reimplemented the clause evaluation code using integer bit strings instead of lists. Her code was also based on Carol's, and compared to her initial 50 seconds on `ag24_15.cnf`, Eve's version takes 25 seconds. Fetch `eve_branch` at `https://www.poirrier.ca/courses/softeng/ex/eve.git`. Review Eve's contribution.

8. Merge `eve_branch` onto `carol_branch`. Resolve merge conflicts. Printing the ones of `ag24_15.cnf` now takes 0.8 seconds.