CS 520
In-class exercise 3
Debugging

Due: **Tuesday, November 5, 2019, 9:00 AM EST** via Moodle. This in-class exercise is a group submission. This means that **each group only needs to submit their solution once** and also that every student in a group will get the same grade. You will work with students within your group, but not with students from other groups. Multiple groups' submissions may not be created jointly. Late assignments will not be accepted without **prior** permission.

Overview and goal

The high-level goal of this exercise is to learn about systematic debugging and to identify and minimize a defect-inducing commit in the version control history of a project.

What to do

Forming groups

1. Team up in groups of size 4. (If you cannot find a 4th member, raise your hand and ask the instructor.)
2. Create a new group on Moodle (see "In-class exercise 3: group selection"), and add all group members.

Set up

1. Make sure that you have Git, Apache Ant, and Java installed.

   Git: `https://git-scm.com/`

   Apache Ant: `http://ant.apache.org/`

   Java: `https://www.oracle.com/technetwork/java/javase/downloads/jdk13-downloads-5672538.html`

2. Clone the `basic-stats` git repository (if you have a clone from a previous assignment, delete it, or clone to a new location):

   ```
   git clone https://github.com/LASER-UMASS/basic-stats basic-stats
   ```

   Read that repository's `README.md` file:

   ```
   cat basic-stats/README.md
   ```

3. Test your set up by compiling, running, and testing the application.

   Note that **one test failure is expected on the `HEAD` revision**.

   ```
   cd basic-stats
   ant compile
   ant -lib lib/ test
   ant clean
   ```

4. Compile and run the application using the `main` method in the `BasicStats.java` file. This will invoke a GUI. Play with it and see whether it works correctly.

   ```
   cd basic-stats
   ant clean
   ant compile
   java -cp bin BasicStats
   ```

   USEFUL HINT: Try running the test case that failed above.

5. Check out the version `v1.0.0` and compile and test the application again. Note that **all tests are expected to pass** on version `v1.0.0`.

   ```
   cd basic-stats
   git checkout v1.0.0
   ant clean
   ant compile
   ant -lib lib/ test
   ```

Background

The developers of this application followed reasonable development practices and used a test-driven approach until they published the first release. Everything just went haywire from there — a lot of late-night, sleep-deprived, pizza-and-drinks hacking.

Unfortunately, the developers introduced a defect at some point **after the first release (v1.0.0)**, which has existed in the code since. While the developers thought about and implemented some automated testing, their automated testing infrastructure does not catch the defect. And since they were so excited about coding, they never bothered to manually check a single test run.

The testing infrastructure that the developers used is a nightly run cron (https://en.wikipedia.org/wiki/Cron) job, which executes the following command:

`ant clean test || reportBug`.

This command executes `ant` to build the application from scratch and to run the tests. If `ant` returns an error (exit code $\neq 0$), then the `reportBug` command sends an email to the developers. Otherwise nothing happens.

Now what?

1. Checkout the `HEAD` revision (`git checkout master`) and run `ant clean test`. Note the test failure and figure out why the testing infrastructure described above can't catch this failure.

   USEFUL HINT: Think about exit codes and take a look at the build file, specifically the used options in the JUnit task that runs the tests.

2. Familiarize yourself with the version control history and determine the number of commits between `v1.0.0` and the `HEAD` revision (including `v1.0.0` and `HEAD`). Note the command(s) that you used.

   USEFUL HINT: Look up the difference in the outputs of the following two commands:

   ```
   git log branch1..branch2
   git log branch1..branch2^
   ```

3. Brainstorm possible strategies for identifying the commit that introduced the defect. Imagine that you are looking at several thousand commits.

4. Familiarize yourself with the `git bisect` command and use it to identify the commit that introduced the defect between version `v1.0.0` and the `HEAD` revision. Note the commit hash and log message of the defect-inducing commit. Verify that you correctly identified the defect-inducing commit.

   USEFUL RESOURCE: https://git-scm.com/docs/git-bisect

5. Isolate the actual defect in the defect-inducing commit, using a delta-debugging-like approach. You'll see delta-debugging on homework 2. Essentially, think about iteratively removing irrelevant changes until only the defect remains. Note that you can employ `git` for this purpose: locally commit the source code after every successful iteration so that you can easily revert back if the next iteration fails.

Questions

Using your notes and results, answer the following questions:

1. Why does the described automated testing infrastructure not catch the defect?
2. How could the developers improve the build file or the testing infrastructure to notice test failures in the future?
3. How many commits exist between `v1.0.0` and the `HEAD` revisions (including `v1.0.0` and `HEAD`)? What command(s) did you use to determine the number?
4. Which commit (commit hash and log message) introduced the defect? How did you verify that this commit indeed introduced the defect?
5. After how many steps (`git bisect` calls) did you identify the defect-inducing commit?
6. Which `git` command can you use to undo the defect-inducing commit? Briefly explain what problem may occur when undoing a commit and what best practice generally mitigates this problem.
7. Write a command ⟨cmd⟩ such that the following two calls of `git bisect` automatically determine the defect-inducing commit:

   `git bisect start HEAD v1.0.0`

   `git bisect run` ⟨cmd⟩

   Note: You may write and script (instead of a single command) and call it as your command. If you do, please include the content of your script in your answer.

Deliverables

Your submission, via Moodle, must be a single (one per group) archive (`.zip` or `.tar.gz`) file with name `<group name>-inclass3.<zip/tar.gz>`, containing:

1. The isolated defect. That is, the minimal change that introduces the defect, when applied to the source code of the last good commit (i.e., the last commit that passes all tests). You may submit a diff (take a look at the `diff` or `git diff` commands) or the buggy source code. The difference between your buggy source code and the last good commit should be only the defect and nothing else.
2. `answers.txt`: A plain-text file with your answers to the above 7 questions. *List all group members on top of this file.*