CS 458 - Homework 3

Deadline

Due by 11:59 pm on Friday, September 15, 2017

Purpose

To think about some of the Extreme Programming practices, including practice writing a user story, and to play a bit with simple branching and merging using `git`.

How to submit

Submit this homework's files using `~st10/458submit` on **nrs-projects**, with a homework number of `3`.

Instructions for using the tool ~st10/458submit

- If they are not already on nrs-projects, transfer your files to be submitted to a directory on nrs-projects.
- Once all of your files to be submitted are in a directory on nrs-projects, then use `ssh` (or Putty in a campus Windows lab) to connect to `nrs-projects.humboldt.edu` .
- use `cd` to change to the directory containing the files to be submitted -- for example,

  ```
  cd 458hw03
  ```

- type the command:

  ```
  ~st10/458submit
  ```

  ...and give the number of the homework being submitted when asked, and answer that, `y`, you do want to submit all of the files with of-interest-to-458 suffixes in the current directory. (It is fine if a few extraneous files get submitted as well -- I'd rather receive too many files that too few, and typing in all of the file names for each assignment is just too error-prone!)
- you are expected to carefully check the list of files that the tool believes have been submitted, and make sure all of the files you hoped to submit were indeed submitted! (The most common error is to try to run `~st10/458submit` while in a different directory than where your files are.)

Problem 1

In a file `458hw3-1.txt`, first put your name and the last modified date. Then, answer each of the following questions, preceding each by the problem part it is the answer for.

Consider the following list of some Extreme Programming practices:

- Pair programming
- Release planning/the planning game
- Unit testing
- Test-infected development/Test-driven development/Test-first development (assume this includes the

concept of writing unit tests BEFORE writing the code to be tested, and assume that, once written, code cannot be checked-in to the current project version unless it passes its unit tests)

- Acceptance testing/customer testing
- Frequent releases/Small releases
- Refactoring
- Simple design/YAGNI (You Aren't Going to Need It)/"do the simplest thing that could possibly work"
- Collective code ownership
- Continuous integration
- On-site customer/"extreme" customer involvement
- 40-hour week/sustainable pace
- Coding standards
- Common vocabulary/system metaphor
- User stories
- Stand-up meetings
- CRC cards - Class, Responsibilities, and Collaboration - for object-oriented programming
- Spike solutions

1 part a

You hopefully have noticed that some of these practices influence, or even depend, on some of the other practices.

Consider the practice of "Collective code ownership". List at least three other practices from the list above that make "Collective code ownership" more practical/feasible.

1 part b

CRC - Class, Responsibilities, and Collaboration - cards are used for brainstorming, for design, and for communication. They do assume an object-oriented approach. You'll find a number of templates on the web, but this is one simple version, based on the description and examples at:

http://www.agilemodeling.com/artifacts/crcModel.htm

An index card has the name of a class on the top of the card. On the left-hand side of the card, you list that class's responsibilities -- something that class knows or does. On the right-hand side of the card, you list collaborator(s) -- other classes that this class interacts with to fulfill its responsibilities.

As an example from the above web page, you might have a card with **Customer** at the top, with responsibilities listed on the left-hand-side of:

- places orders
- knows name
- knows address

- knows customer number
- knows order history

On the right-hand-side, it might have a collaborator-class listed of:

- Order

It is considered a benefit that index cards are small, and easy to move around on a desk (or perhaps a computer desktop) -- you can put cards for classes that collaborate with each other near each other, and cards that don't further apart. You can also easily throw away one or more cards and start over if one approach isn't working!

Consider a Monopoly game program. You might have a `Player` class in such a game.

Give an example of at least one **responsibility** that a `Player` class might have.

Give an example of at least one **collaborator-class** that a `Player` class might have to interact with to fulfill its responsibilities.

1 part c

Consider the example user stories from Week 4 Lecture 2 -- recall, one of these was:

EXAMPLE 1 of a user story

AS AN internet banking customer,
I WANT to see a rolling balance for my everyday account
SO THAT I know the balance of my account after transaction is applied

estimate: 5 hours

Example acceptance criteria:
* The rolling balance is displayed
* The rolling balance is calculated for each transaction
* The balance is displayed for every transaction for the full period of time transactions are available
* The balance is not displayed if a filter has been applied

Now, consider a Monopoly game program. Develop a user story for this program, being sure to state the user story in the style shown above, including an implementation-time estimate, and including an attempt at example acceptance criteria.

Submit your resulting file `458hw3-1.txt`

Problem 2 - stage 1

(some references I used in kluging this together:

- http://pcottle.github.io/learnGitBranching/
- https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging
- https://git-scm.com/book/en/v2/Git-Branching-Branch-Management

- https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows

)

You are going to use GitHub team `cs458`'s repository `458hw02` for several problems, along with `git` on nrs-projects. The goal is to give you a little taste/experience with Git branching and merging, so that you can hopefully make use of these while working on your team project.

(Remember that if you are prompted by these commands for your password, that should be your **GitHub** password...)

- Make a new `458hw03` directory on nrs-projects, protect it, and change to it:

  ```
  mkdir 458hw03
  chmod 700 458hw03
  cd 458hw03
  ```

- And now, clone the `458hw02` repository that GitHub team `cs458` has access to into this directory on nrs-projects:

  ```
  git clone https://github.com/hsu-cs458-f17/458hw02.git
  ```

  ...and enter your GitHub username and password when prompted. (Yes, you will now have a repo named `458hw02` within your directory `458hw03`...!)

- Demonstrate that you have reached this stage by running the following commands (STARTING in the nrs-projects directory in which you ran the above command, which should be the one now **containing** your cloned repository directory) (note that >> appends to a file...)

  ```
  echo yourName >> 458hw03-stage1.txt
  pwd >> 458hw03-stage1.txt
  ls 458hw02/* >> 458hw03-stage1.txt
  ```

Submit your resulting file `458hw03-stage1.txt`

Problem 2 - stage 2

- **NOW** do:

  ```
  cd 458hw02
  ```

  ...so you are IN your cloned GitHub repo. And go to subdirectory `class-greetings`:

  ```
  cd class-greetings
  ```

Reminders...

Basic idea - you can use a branch to explore an idea, or work on adding a particular user story, or work on fixing a particular bug, etc.

But there are two parts of this -- **creating** the branch, and then **checking out** that branch. If you create a new branch and don't check it out, you'll keep working in whatever branch you WERE in -- any changes won't be made in the new branch!

So, commands `git branch` and `git checkout` can do this (yes, there are shorthands, also, BUT we'll do these "longhand" for these problems to hopefully make it clearer what you are doing). That is:

```
git branch myNewBranch  # create a new branch myNewBranch
git checkout myNewBranch # now you have checked out branch myNewBranch,
                         #    and changes you make and stage and commit
                         #    will be part of this branch (until you
                         #    check out another branch)
```

And, you can use the command `git branch` with no arguments to list your current branches -- and the output will show an asterisk `*` next to the current branch.

Continuing with Problem 2 - Stage 2

- SO, now: create and checkout a branch named `458hw03-branch`:

  ```
  git branch 458hw03-branch
  git checkout 458hw03-branch
  ```

- Run the following commands to see your current branches, and then to record in a file to submit that you have indeed created and checked out this branch (remembering that, right now, you should be in the `class-greetings` directory of your `458hw02` repo/directory):

  ```
  git branch
  echo yourName >> ../../458hw03-stage2.txt
  git branch >> ../../458hw03-stage2.txt
  ```

You're now going to make some specific changes in this branch `458hw03-branch`.

- As part of Homework 2, you created a file in `class-greetings` named *yourGitHubUsername*`.txt` that contained a greeting and your real name.

  Now EDIT this file, **adding** a message saying this is part of Homework 3 - Problem 2 - Stage 2 and was added in branch `458hw03-branch`. (Leave your Homework 2 contents, do not remove them!)

- Stage and commit your modified file:

  ```
  git add yourGitHubUsername.txt
  git commit -m "doing HW 3 problem 2 stage 2"
  ```

Also recall that the command `git branch -v` shows the last commit on each branch.

- Run the following commands to see your current status and last commit on each branch, and also record them for homework problem submission:

  ```
  git status
  git status >> ../../458hw03-stage2.txt
  git branch -v
  git branch -v >> ../../458hw03-stage2.txt
  ```

Submit your resulting file `458hw03-stage2.txt`

Problem 2 - stage 3

Now -- you are "called away" to fix ANOTHER problem, and you need/want to go back to the `master`

branch and come back to `458hw03-branch` later.

My understanding thus far is: as long as you commit your changes in your current branch, it should be no problem to switch to/check out another branch, and then come back to this one later.

Conveniently, you just happened to commit your branch `458hw03-branch` at the end of the previous stage.

- So, attempt to switch back to your `master` branch, running:

  `git checkout master`

- Show that you have switched:

  `echo` *yourName* `>> ../../458hw03-stage3.txt`

  `git branch`

  `git branch >> ../../458hw03-stage3.txt`

- Now -- you are IN the `master` branch. Another way you can tell: look at the contents of the file *yourGitHubUsername*`.txt` -- they should, indeed, be the original contents! Run the commands:

  `more` *yourGitHubUsername*`.txt`

  `more` *yourGitHubUsername*`.txt >> ../../458hw03-stage3.txt`

- Create and checkout a branch named `458hw03-aside`:

  `git branch 458hw03-aside`

  `git checkout 458hw03-aside`

- ...and see/show that you have done so:

  `git branch`

  `git branch >> ../../458hw03-stage3.txt`

- Edit this branch's version of *yourGitHubUsername*`.txt`, carefully adding a message saying this is for Homework 3 - Problem 2 - Stage 3's aside.

- Stage and commit your modified file:

  `git add` *yourGitHubUsername*`.txt`

  `git commit -m "doing HW 3 problem 2 stage 3"`

- Run the following commands to see your current status and last commit on each branch, and also record them for homework problem submission:

  `git status`

  `git status >> ../../458hw03-stage3.txt`

  `git branch -v`

  `git branch -v >> ../../458hw03-stage3.txt`

Submit your resulting file `458hw03-stage3.txt`.

Problem 2 - stage 4

Now you are going to merge the change you just made in branch `458hw03-aside` into your `master` branch.

To do this, you'll switch branches back to your `master` branch, and then use the `git merge` command with the name of the branch whose changes you'd like to attempt to merge with the `master` branch.

- First, go back to the `master` branch, and show that you have done so:

```
git checkout master
echo yourName >> ../../458hw03-stage4.txt
git branch -v
git branch -v >> ../../458hw03-stage4.txt
more yourGitHubUsername.txt
more yourGitHubUsername.txt >> ../../458hw03-stage4.txt
```

- ...and now merge your work from branch `458hw03-aside` into the current `master` branch:

```
git merge 458hw03-aside
git branch -v
git branch -v >> ../../458hw03-stage4.txt
more yourGitHubUsername.txt
more yourGitHubUsername.txt >> ../../458hw03-stage4.txt
```

SINCE you've finished with the `458hw03-aside` branch now, you can delete it, using the `git branch` command's `-d` option.

- Run the commands:

```
git branch -d 458hw03-aside
git branch -v
git branch -v >> ../../458hw03-stage4.txt
```

Submit your resulting file `458hw03-stage4.txt`.

Problem 2 - stage 5

Now you are going to go back to your branch `458hw03-branch` and continue working on it.

- Run these commands to go back to branch `458hw03-branch` and show that you have done so:

```
echo yourName >> ../../458hw03-stage5.txt
git checkout 458hw03-branch
git branch -v
git branch -v >> ../../458hw03-stage5.txt
more yourGitHubUsername.txt
```

```
more yourGitHubUsername.txt >> ../../458hw03-stage5.txt
```

- And now "complete" your work on this branch, first by adding another line to *yourGitHubUsername*`.txt`, with a message saying you've now FINISHED branch `458hw03-branch`.
- Stage and commit your modified file:

```
git add yourGitHubUsername.txt
git commit -m "doing HW 3 problem 2 stage 5"
```

- Run the following commands to see your current status and last commit on each branch, and also record them for homework problem submission:

```
git status
git status >> ../../458hw03-stage5.txt
git branch -v
git branch -v >> ../../458hw03-stage5.txt
```

Submit your resulting file `458hw03-stage5.txt`.

Problem 2 - stage 6

And, now I decide I'm happy with the work in branch `458hw03-branch`, and want to merge it into my `master` branch.

Again, I'll use `git checkout` to switch to the branch I want to merge INTO, and then use `git merge` with the name of the branch whose changes I want to bring in.

- First, go back to the `master` branch, and show that you have done so:

```
git checkout master
echo yourName >> ../../458hw03-stage6.txt
git branch -v
git branch -v >> ../../458hw03-stage6.txt
more yourGitHubUsername.txt
more yourGitHubUsername.txt >> ../../458hw03-stage6.txt
```

- ...and now attempt to merge your work from branch `458hw03-branch` into the current `master` branch (which, by the way, **should fail**, complaining about a **merge conflict**.)

```
git merge 458hw03-branch
git status
git status >> ../../458hw03-stage6.txt
```

- Files with merge conflicts now have additional lines added, called **conflict resolution markers**.

  Check these out, and show you have them:

```
more yourGitHubUsername.txt
more yourGitHubUsername.txt >> ../../458hw03-stage6.txt
```

- For this homework's purposes, it turns out that you want to keep all of the lines added in the various stages, SO:

  Carefully edit your *yourGitHubUsername*`.txt`, removing **JUST** the conflict resolution marker lines:

  ```
  <<<<<<< HEAD
  =======
  >>>>>>> 458hw03-branch
  ```

- After you have resolved each of the sections with conflicts in each conflicted file, you stage its commits to mark it as resolved. You only have one such file and you've handled its conflicts, so do this staging:

  `git add` *yourGitHubUsername*`.txt`

  ...and commit it:

  ```
  git commit -m "doing HW 3 problem 2 stage 6"
  ```

- Run the following commands to see your current status and last commit on each branch, and also record them for homework problem submission:

  `more` *yourGitHubUsername*`.txt`

  ```
  more yourGitHubUsername.txt >> ../../458hw03-stage6.txt
  git status
  git status >> ../../458hw03-stage6.txt
  git branch -v
  git branch -v >> ../../458hw03-stage6.txt
  ```

SINCE you've finished with the `458hw03-branch` branch now, you can delete it, using the `git branch` command's `-d` option.

- Run the commands:

  ```
  git branch -d 458hw03-branch
  git branch -v
  git branch -v >> ../../458hw03-stage6.txt
  ```

- Finally, cross your fingers and try pushing your updated `458hw02` repository back up to GitHub:

  ```
  git push origin master
  ```

  (I am hoping very hard that, since each of us should only be changing "our" file in class-greetings, that this MIGHT just work...!)

Submit your resulting file `458hw03-stage6.txt`.