Git Workshop

Part 2 of 2

Merge Conflics, Remote Repositories
and
Hosting Services

Workshop Agenda

- Part 1 – Another Deck
  - Installation and Setup
  - Concepts
  - Repository Initialization
  - Clone
  - Basic Lifecycle
  - Logs
  - Introduction to Branches
- Part 2 – This Deck
  - Merge Conflicts
  - Remote Repositories
  - Tags
  - Hosting Services
    - GitHub
    - AWS CodeCommit

This is a "presentation-ification" of the single-page workshop available at

https://github.com/lacounty-isab/workshops/tree/master/git

Merge Conflicts

We are going to introduce some changes in our files on different branches that **conflict**. In this context, it means that different branch change the same line of the same file in different ways. We saw in Part 1 that neither

- changing different files, nor
- changing the same file on different lines

incur a conflict. The **git merge** command will merge these deterministically.

In the following exercise we edit two files: a plain text file and a small Python program.

```
 1 # This file contains mappings.
 2 #
 3 a1 - 20
 4 a2 - 43
 5
 6 b1 - 39
 7 b2 - 34
 8 b3 - 44
 9
10 c1 - 45
11 c2 - 19
```

file1.txt

To keep things distinct from Part 1, we'll create a new directory, `sample3`, with these files and create a new repository.

```
GitWorkshop/samples3$ ls
file1.txt  file2.py
GitWorkshop/samples3$ git init
Initialized empty Git repository in GitWorkshop/samples3/.git/
GitWorkshop/samples3$ git add .
GitWorkshop/samples3$ git commit -m "Initial version."
[master (root-commit) 4fff5a8] Initial version.
 2 files changed, 22 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.py
GitWorkshop/samples3$ git log --oneline
4fff5a8 (HEAD -> master) Initial version.
```

```
 1
 2 def print_usage():
 3     usage = """Usage: addAudit.py [-f] [-v] <filename ...>"
 4      -f - overwrite when duplicate key encountered
 5      -v - verbose
 6      <filename ..> the name of at least one audit file."""
 7
 8     print(usage)
 9
10 print_usage()
```

file2.py

Exercise 8 – Branch B2

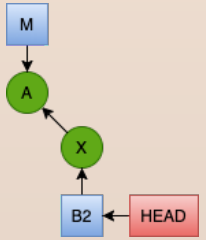
Create a branch **B2** and edit the two files.

1. Create a new branch **B2**.

```
GitWorkshop/samples3$ git checkout -b B2
Switched to a new branch 'B2'
```

2. Edit `file1.txt` as shown on the right.
   a. **Line 6**: Add ",41"
   b. **Line 7**: Change "34" to "36"
   c. **Line 8**: Add a blank space after 44.
   d. **Line 10**: Change "45" to "55"
   e. **Line 11**: Change "19" to "29"
3. Edit `file2.py` by adding a blank space to each line of the `print_usage` function.
4. Check your work with `git diff`.
5. Add the changes to the staging area.
6. Commit with message "B2 changes."

This creates a new commit **X** referenced by branch **B2** as shown in the figure above.




```
 1 # This file contains mappings.
 2 #
 3 a1 - 20
 4 a2 - 43
 5
 6 b1 - 39,41
 7 b2 - 346
 8 b3 - 44
 9
10 c1 - 455
11 c2 - 129
```

file1.txt

```
 1
 2 def print_usage():
 3     usage = """Usage: addAudit.py [-f] [-v] <filename ...>"
 4       -f - overwrite when duplicate key encountered
 5       -v - verbose
 6       <filename ..> the name of at least one audit file."""
 7
 8     print(usage)
 9
10 print_usage()
```

file2.py

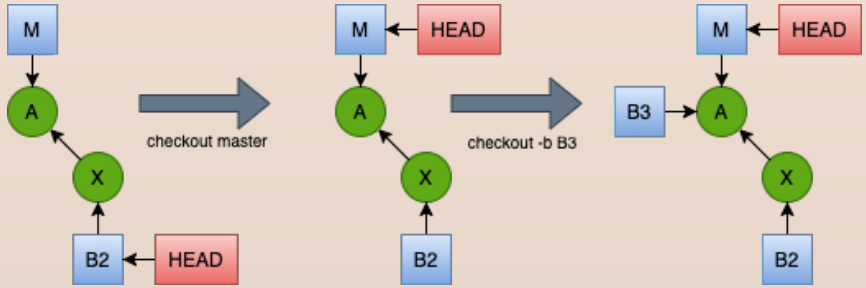
Note that some changes are not as easy to see with the **git diff** command.

Extra Credit: The edit to `file2.py` may seem peculiar. What is a common cause for this?

Exercise 8 – Branch B3

7. Now we're going switch to a new branch **B3** starting from commit **A** just like **B2** did.

```
GitWorkshop/samples3$ git checkout master
Switched to branch 'master'
GitWorkshop/samples3$ git checkout -b B3
Switched to a new branch 'B3'
```



8. Edit `file1.txt`.
   a. **Line 3**: change "20" to "30".
   b. **Line 4**: change "43" to "53".
   c. **Line 6**: add ",40".
   d. **Line 7**: change "34" to "35".

```
 1 # This file contains mappings.
 2 #
 3 a1 - 230
 4 a2 - 453
 5
 6 b1 - 39,40
 7 b2 - 345
 8 b3 - 44
 9
10 c1 - 45
11 c2 - 19
```

file1.txt

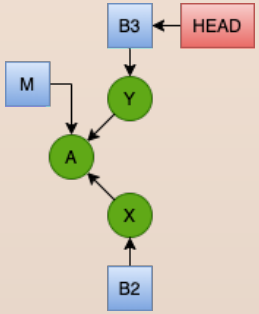
9. Edit `file2.py`. Change the triple-quoted string to a series of print statements.
10. Check your work with git diff.
11. Add the changes to the staging area.
12. Commit with message "B3 changes."

```
1
2 def print_usage():
3     print("Usage: addAudit.py [-f] [-v] <filename ...>")
4     print("  -f - overwrite when duplicate key encountered")
5     print("  -v - verbose")
6     print("  <filename ..> the name of at least one audit file.")
7
8 print_usage()
```

file2.py

Exercise 8 – Branch master

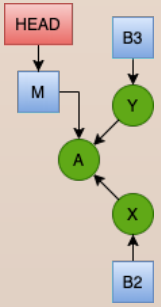
After the B3 commit, we have the branch configuration shown to the right. Since the target of a merge is always the current branch, we're going to change back to the **master** branch. Then merge each of **B2** and **B3**.



13. Change to the **master** branch.

```
git checkout master
```

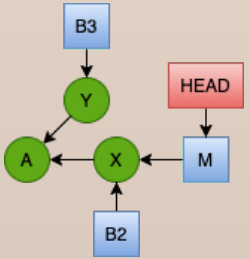
This moves HEAD to the master branch.



14. Merge B2

```
git merge B2
```

This should be a fast-forward merge which simply advances the master branch to B2



15. Finally, issue the command to merge **B3**. This is where the fireworks start.

```
GitWorkshop/samples3$ git merge B3
Auto-merging file2.py
CONFLICT (content): Merge conflict in file2.py
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

What just happened here?

Git auto-merges on a line-by-line basis. When the same line is changed in two different ways, Git places "merge markers" around those lines to indicate two incompatible changes were made. They must be manually resolved.

The changes are easy to spot. They are bounded by the markers

`<<<< HEAD` – **beginning of master**

`=========` – **separates versions**

`>>>>>> B2` – **end of B3 version**

Resolving the merge conflict amounts to choosing one or the other of these versions to keep.

```
 1 # This file contains mappings.
 2 #
 3 a1 - 30
 4 a2 - 53
 5
 6 <<<<<<< HEAD
 7 b1 - 39,41
 8 b2 - 36
 9 b3 - 44
10 =======
11 b1 - 39,40
12 b2 - 35
13 b3 - 44
14 >>>>>>> B3
15
16 c1 - 55
17 c2 - 29
```

This is the HEAD (i.e. master) version of the lines (originally B2)

This is the B3 (i.e. incoming) version of the lines

Exercise 8 – Resolve Conflict 1

We have to decide, based on a larger perspective, how to resolve the conflicts. We could

- pick the left side (**master** branch version)
- pick the right side (**B3** branch version)
- choose something completely different from either side based on some knowledge we might have.

These decisions are carried out in the following steps.

a. **Edit the lines** within the merge markers based on your decisions

b. **Delete the merge marker** lines and save the file.

c. **Add the file** to the Git staging area.

This last step is how Git knows when we've completed the merge activity for the file. We repeat steps a, b and c for each file in which a merge conflict occurred.

Looking at `file1.txt` in the previous slide, things aren't that bad. The top (only changed on **B2**) and the bottom (only changed on **B3**) were auto-merged. Only the middle third, where both **B2** and **B3** changed lines, requires resolution.

16. Edit the lines of `file1.txt`.

a) For entry `b1`, branch **B2** added `41` while branch **B3** added `40` to the value. Let's make the decision to add both so that line 7 has `b1 - 39,40,41`. Notice how we're implicitly making lines 7 – 9 our "definitive copy."

b) For entry `b2`, branch **B2** changed the value to `36` which branch **B3** changed it to `35`. Let's decide to keep `36` so that line 8 remains unchanged.

c) Entry `b3` is tricky. It looks the same in both lines 9 and 13. Recall that branch **B2** erroneously added a space at the end of the line. In this case, we wish to accept the **B3** line 13 which left the line unchanged. Remove the last space on line 9.

```
 6 <<<<<<<•HEAD␍␊
 7 b1•-•39,41␍␊
 8 b2•-•36␍␊
 9 b3•-•44•␍␊
10 =======␍␊
11 b1•-•39,40␍␊
12 b2•-•35␍␊
13 b3•-•44␍␊
14 >>>>>>>•B3␍␊
```

```
 6 <<<<<<<•HEAD␍␊
 7 b1•-•39,40,41␍␊
 8 b2•-•36␍␊
 9 b3•-•44␍␊
10 =======␍␊
11 b1•-•39,40␍␊
12 b2•-•35␍␊
13 b3•-•44␍␊
14 >>>>>>>•B3␍␊
```

Exercise 8 – Resolve Conflicts 3

17. Delete the merge markers. Lines 7 – 9 are now in the form we want to keep. We can delete lines 11 – 13 as well as the merge markers on lines 6, 10 and 14. Then save the file.

```
 6 <<<<<<< HEAD
 7 b1 - 39,40,41
 8 b2 - 36
 9 b3 - 44
10 =======
11 b1 - 39,40
12 b2 - 35
13 b3 - 44
14 >>>>>>> B3
```

```
 6 b1 - 39,40,41
 7 b2 - 36
 8 b3 - 44
```

18. Add `file1.txt` to the Git staging area.

```
git add file1.txt
```

19. Perform the same merge steps a, b and c for `file2.py`. This is a simpler case where we wish to only accept the **B3** version. The **B2** version had simply added an extra space on each line, which is not uncommon for some editors. Simply delete lines 3 – 10 and line 15. Then save the file and add it to the staging area.

```
 1
 2 def print_usage():
 3 <<<<<<< HEAD
 4     usage = """Usage: addAudit.py [-f] [-v] <filename ...>"
 5      -f - overwrite when duplicate key encountered
 6      -v - verbose
 7      <filename ..> the name of at least one audit file."""
 8
 9     print(usage)
10 =======
11     print("Usage: addAudit.py [-f] [-v] <filename ...>")
12     print("  -f - overwrite when duplicate key encountered")
13     print("  -v - verbose")
14     print("  <filename ..> the name of at least one audit
file.")
15 >>>>>>> B3
16
17 print_usage()
```

```
 1
 2 def print_usage():
 3     print("Usage: addAudit.py [-f] [-v] <filename ...>")
 4     print("  -f - overwrite when duplicate key encountered")
 5     print("  -v - verbose")
 6     print("  <filename ..> the name of at least one audit
file.")
 7
 8 print_usage()
```

Exercise 8 - Commit

21. With both file conflicts resolved and added to the Git staging area, we can now create the merge commit. It should be like any other commit; the hard work is over.

```
git commit -m "Merge branch B3"
```

22. Verify the branch activity using **git log**.

```
GitWorkshop/samples4$ git log --oneline --graph
*   a8a9c39 (HEAD -> master) Merge branch 'B3'
|\
| * c76d480 (origin/HEAD, origin/B3, B3) B3 changes.
* | eaf393f (origin/B2, B2) B2 changes.
|/
* 4fff5a8 (origin/master) Initial version.
```