Continuous Learning: Git and Its Discontents

January 19, 2017

REGIONS

Topics of this Lesson

1. Visualizing the branch workflow
2. Merging with Conflicts
3. Homework Assignment

REGIONS

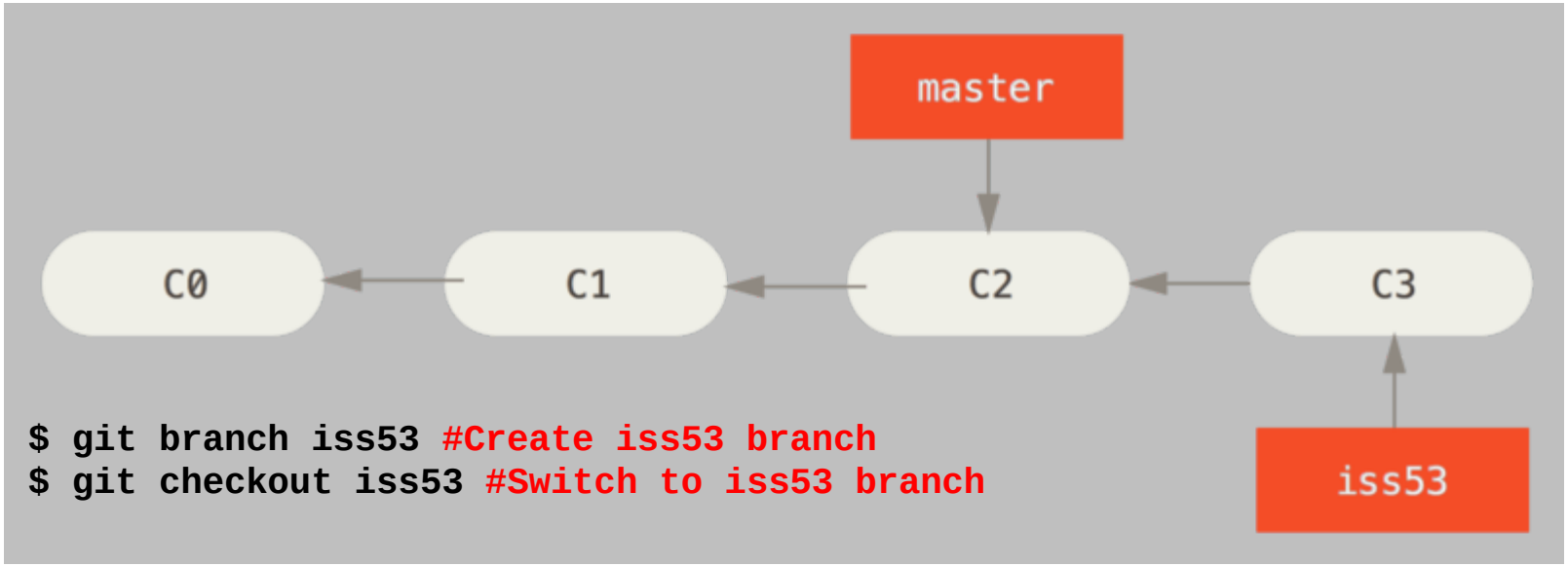
What You Will Learn

- The basic workflow of a single-user with multiple branches.
- How to resolve conflicts when merging a file from two branches.

REGIONS

Branch Basics

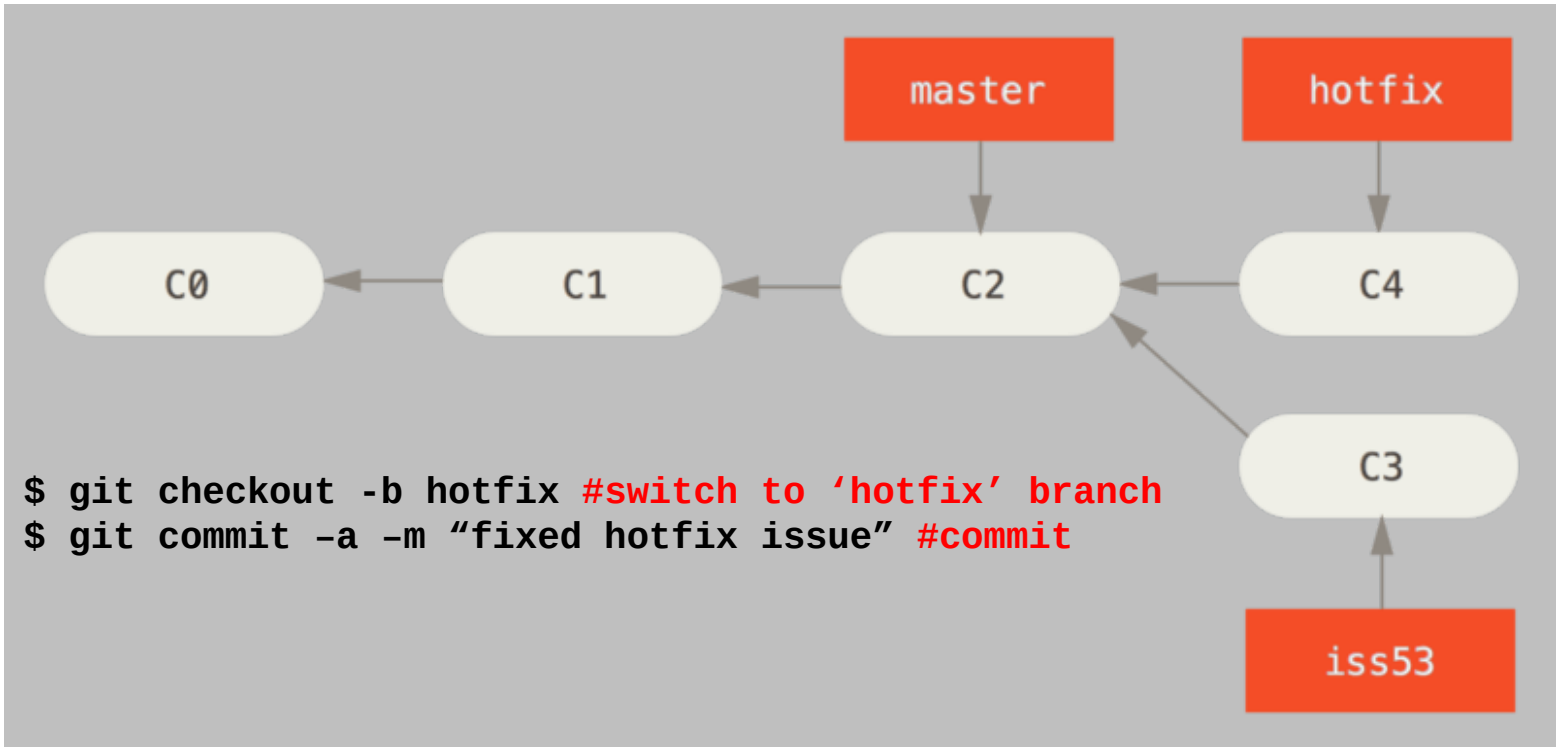
- Assume you have a project with a few commits and production code in the ‘master’ branch with a second branch to address Issue 53 of a project.



REGIONS

Multiple Branches

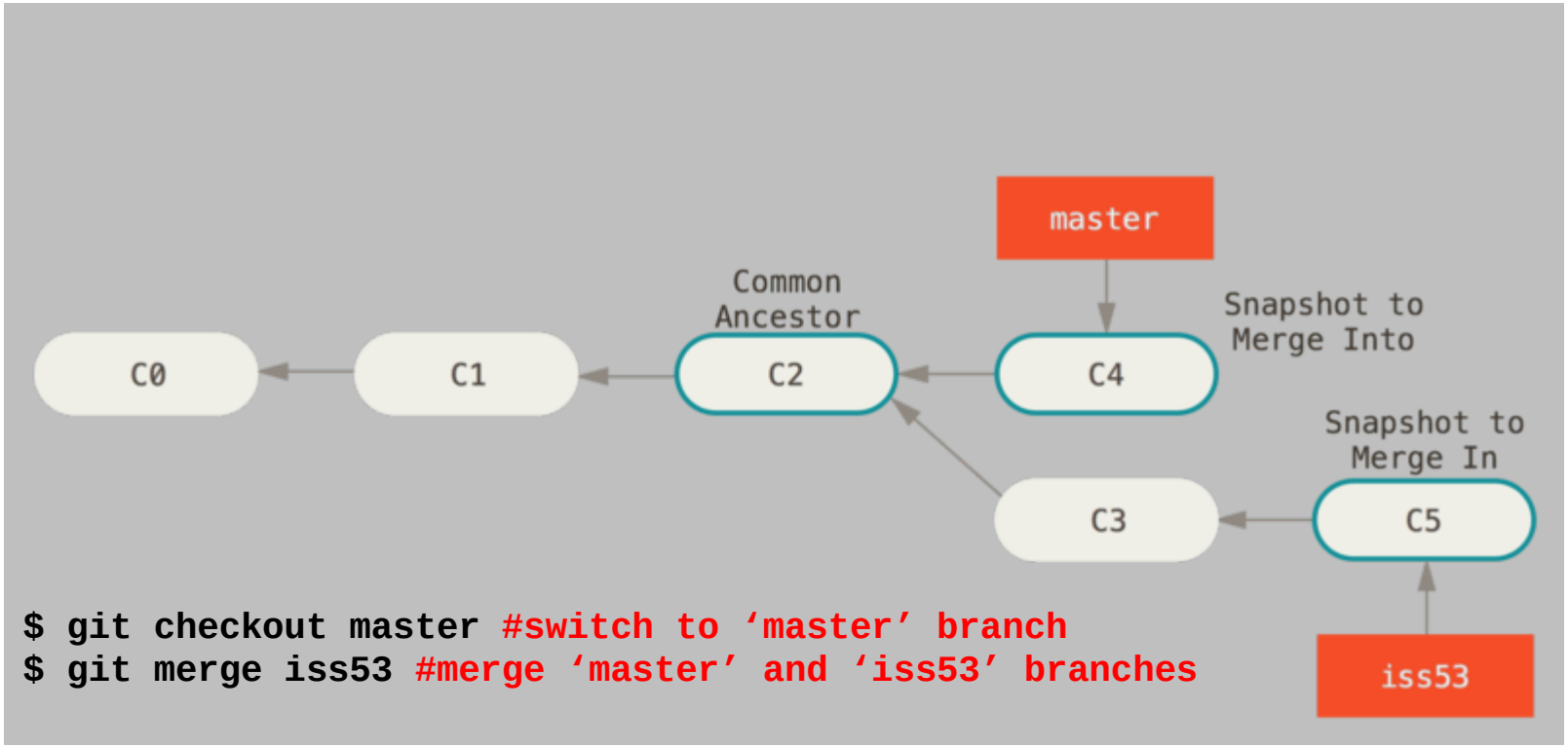
- Now assume you have to perform a quick fix to the code in master branch. You can checkout from iss53 and create a new branch of the production code called ‘hotfix’ to work on a quick fix to the production code.



REGIONS

Merging Branches Back Together

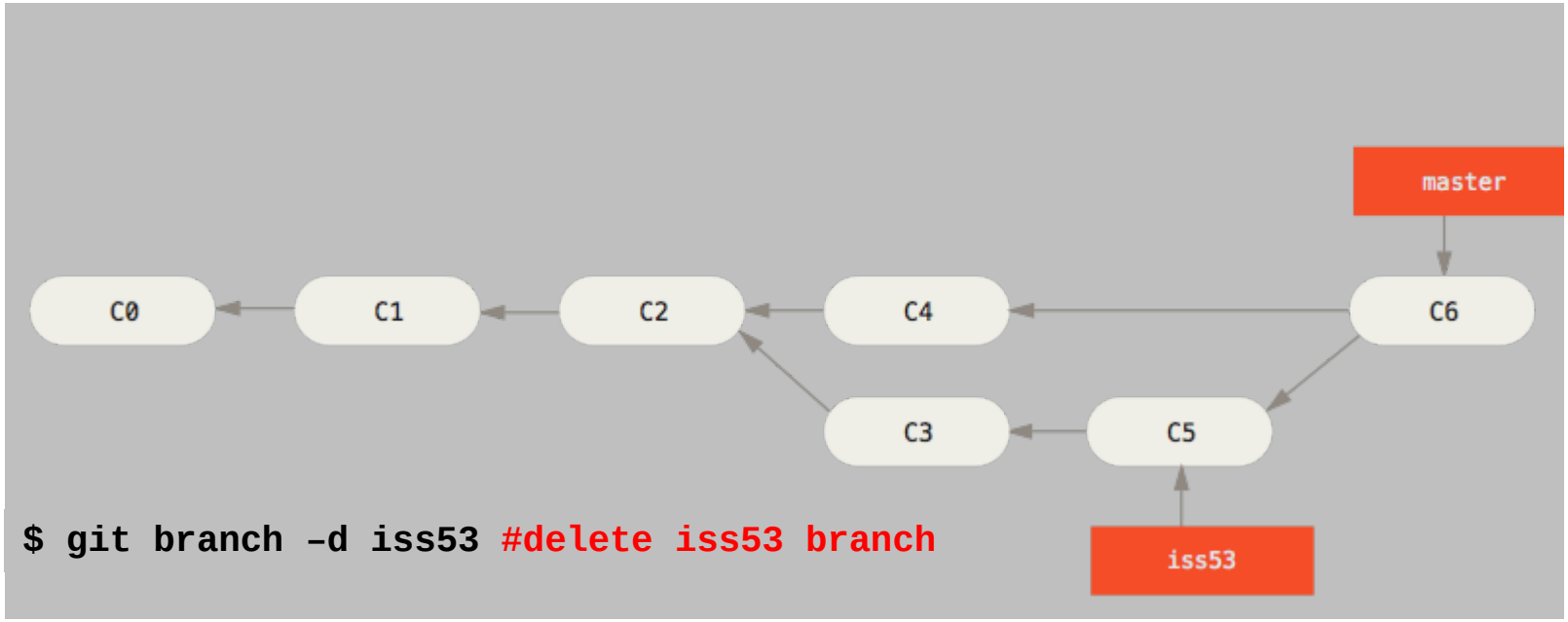
- With the quick fix from the ‘hotfix’ branch deployed, we can return and finish coding Issue 53. Now, we can merge the two branches back together.



REGIONS

Final Workflow after Merge

- With the quick fix from the ‘hotfix’ branch deployed, we can return and finish coding Issue 53. Now, we can merge the two branches back together.



REGIONS

Handling Merge Conflicts

- The `merge` statement earlier in the slides assumes we haven't changed the same part of the same file differently in the two branches. However, sometimes we aren't so lucky when working off of multiple computers or as one contributor to a group project.
- But too often we run across the following:

```
$ git merge dev
Auto-merging code.js
CONFLICT (content): Merge conflict in code.js
Automatic merge failed; fix conflicts and then commit the result.
```

REGIONS

Merge Conflict: Example

`master` branch

```
var gulp = require('gulp'),
  connect - require('gulp-connect');

gulp.src('./client/templates/*.jade')
        .pipe(jade())
        .pipe(gulp.dest('./build/templates'))
        .pipe(minify())
        .pipe(gulp.dest('./build/minifield_templates'));

gulp.task('webserver', function() {

        connect.server({
                livereload: true
        });
});

gulp.task('default', ['webserver']);
~
~
```

`dev` branch

```
var gulp = require('gulp'),
  connect - require('gulp-connect');

gulp.src('client/js/**/*js')
        .pipe(minify())
        .pipe(gulp.dest('build');

gupl.src('client/js/**/*.js', { base: 'client'})
        .pipe(minify())
        .pipe(gulp.dest('build'));

gulp.task('webserver', function() {

        connect.server({
                livereload: true
        });
});

gulp.task('default', ['webserver']);
~
~
```

REGIONS

Merge Conflict: Example

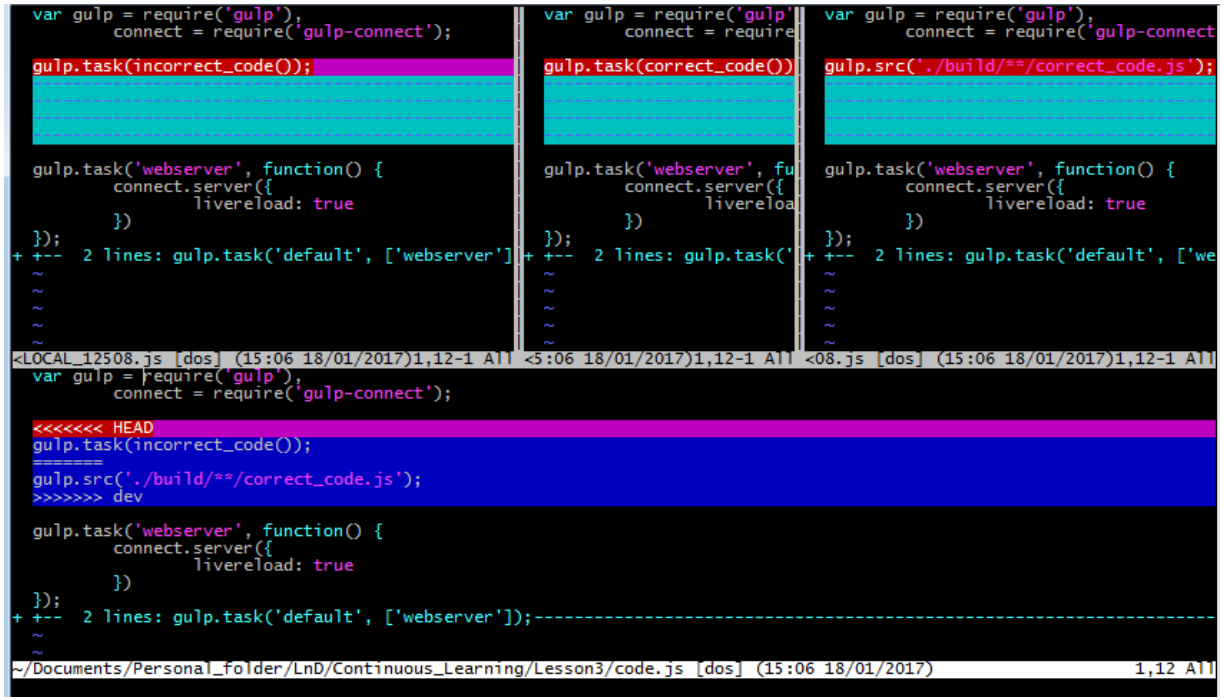
- Better yet, we can use the `git diff -b` command to have git tell us the difference between the two branches.
- Question: Which branch does the ‘HEAD’ code belong to?

```
$ git diff -b
diff --cc code.js
index 0f19dfe,03269a6..0000000
--- a/code.js
+++ b/code.js
@@@ -1,11 -1,13 +1,21 @@@
  var gulp = require('gulp'),
        connect = require('gulp-connect');

++<<<<<<< HEAD
 +glup.src('./client/templates/*.jade')
 +       .pipe(jade())
 +       .pipe(gulp.dest('./build/templates'))
 +       .pipe(minify())
 +       .pipe(gulp.dest('./build/minified_templates'));
++=======
+ gulp.src('client/js/**/*.js')
+        .pipe(minify())
+        .pipe(gulp.dest('build'));
+
+ gulp.src('client/js/**/*.js', {base: 'client'})
+        .pipe(minify())
+        .pipe(gulp.dest('build'));
++>>>>>>> dev

  gulp.task('webserver', function() {
        connect.server({
```

REGIONS

Mergetool: The Go-To Merge Resolutor

- The `mergetool` command opens an interactive session to resolve conflicts among branches in a single file.
- I will be using `vimdiff` to resolve the conflicts in this resolution.



REGIONS

Post-Merge: Tidy Up

- Once we are done merging, we should clean up the `code.js.orig` which is a copy of the original base code.
- Additionally, we should delete the branch dev where we were working so that the workflow is clean again with a single path.

- And that's all she wrote! (for today…)

REGIONS

Homework Assignment

- Take a completed project from Lesson 1 or Lesson 2 and push your results to Github.
- Create a second branch off of master with a change that will cause a conflict.
- Use the mergetool to resolve at least one conflict among branches.

REGIONS

Next Time

- Next time, let’s cover the workflow of git in a group of developers… which we didn’t cover this time because I had already made the slides for the basic workflow…

REGIONS

References

- Lesson code and examples are pulled from the book *Pro Git* found at https://git-scm.com/book/en/v2.
- Specific chapters references are Chapters 3.2. Chapter 7.8 has other options for handling merge conflicts, but I was not able to replicate the example.
- Merge conflict example pulled from https://coderwall.com/p/o1psnw/handle-a-git-merge-conflict.

REGIONS