Dealing with Conflicting Updates in Git

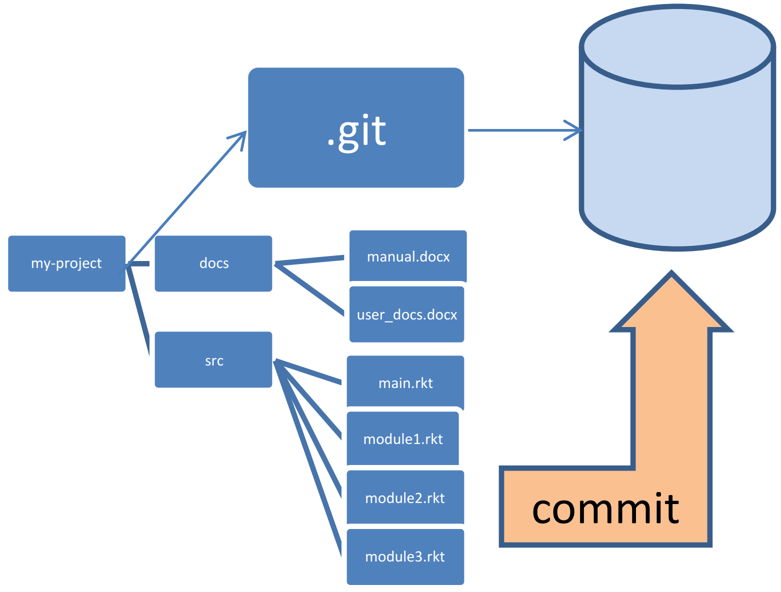
CS 5010 Program Design Paradigms

Lesson 0.6

Learning Objectives

- At the end of this lesson you should be able to:
  - explain what happens when you pull changes from an upstream repository
  - understand what a conflict is
  - resolve a simple merge conflict in a text file (including a .rkt file)

A Commit

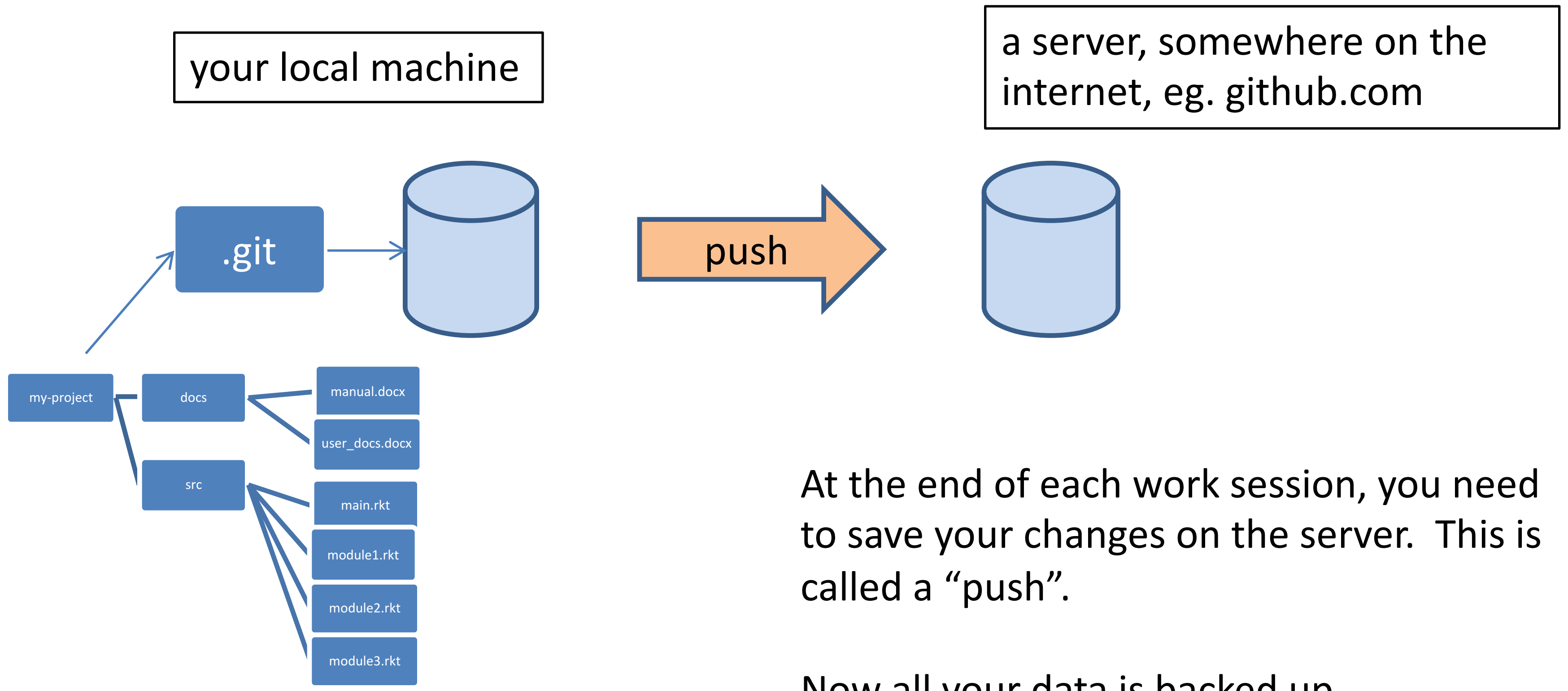


Remember the basic story from the preceding lesson

When you do a "commit", you record all your local changes into the mini-fs.

The mini-fs is "append-only". Nothing is ever over-written there, so everything you ever commit can be recovered.

Synchronizing with the server (1)

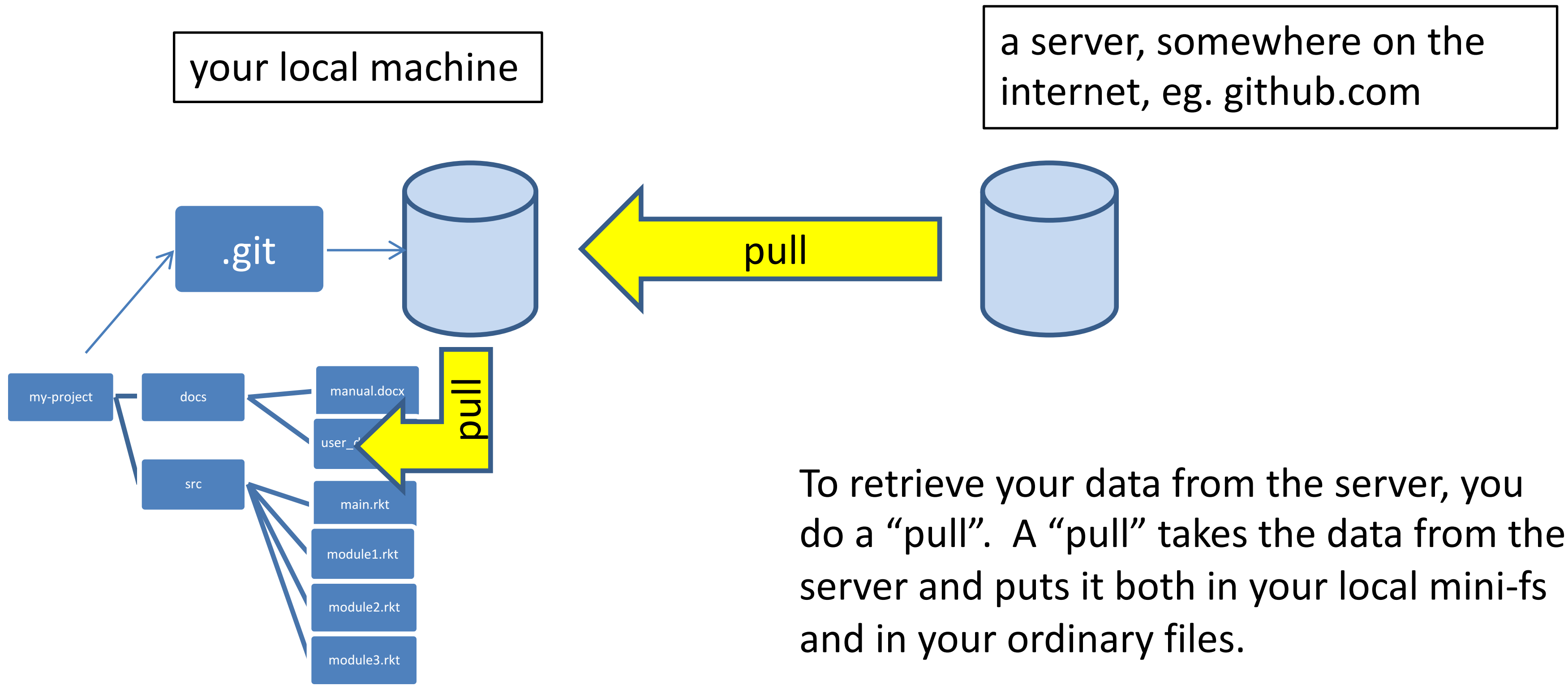


At the end of each work session, you need to save your changes on the server. This is called a "push".

Now all your data is backed up.

- You can retrieve it, on your machine or some other machine.
- We can retrieve it (that's how we collect homework)

Synchronizing with the server (2)

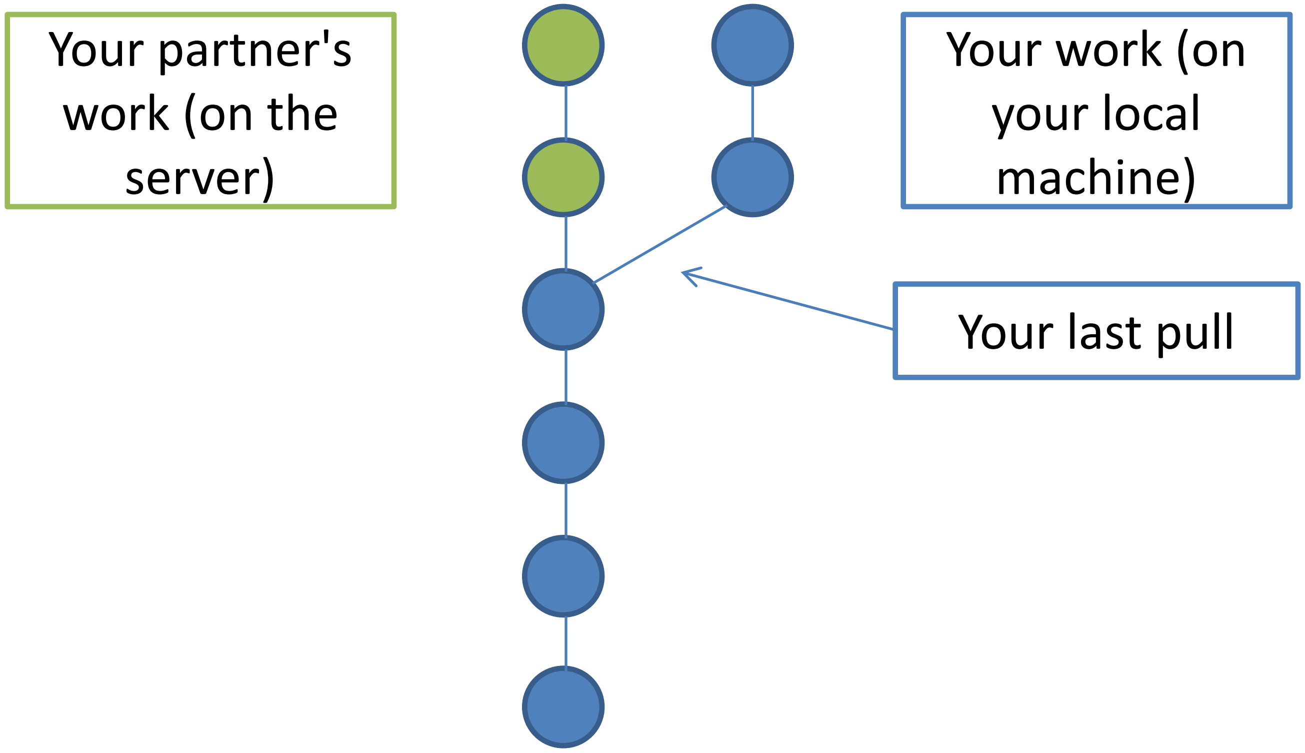


To retrieve your data from the server, you do a "pull". A "pull" takes the data from the server and puts it both in your local mini-fs and in your ordinary files.

~~If your local file has changed, git will merge~~ the changes if possible. If it can't figure out how to the merge, you will get an error message. ~~Dealing with this is beyond the scope of this tutorial~~ ☹

Q: When might you need to merge?

A: When your partner committed some changes to the server, which you don't have.

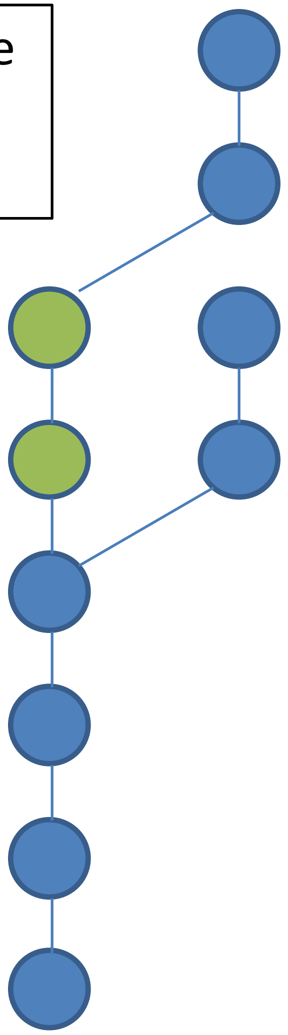


Your partner's work (on the server)
Your work (on your local machine)
Your last pull

Result of Syncing

Your changes are applied to the latest version on the server. This is called "rebasing"

Combined work now lives on both the server and your local machine.



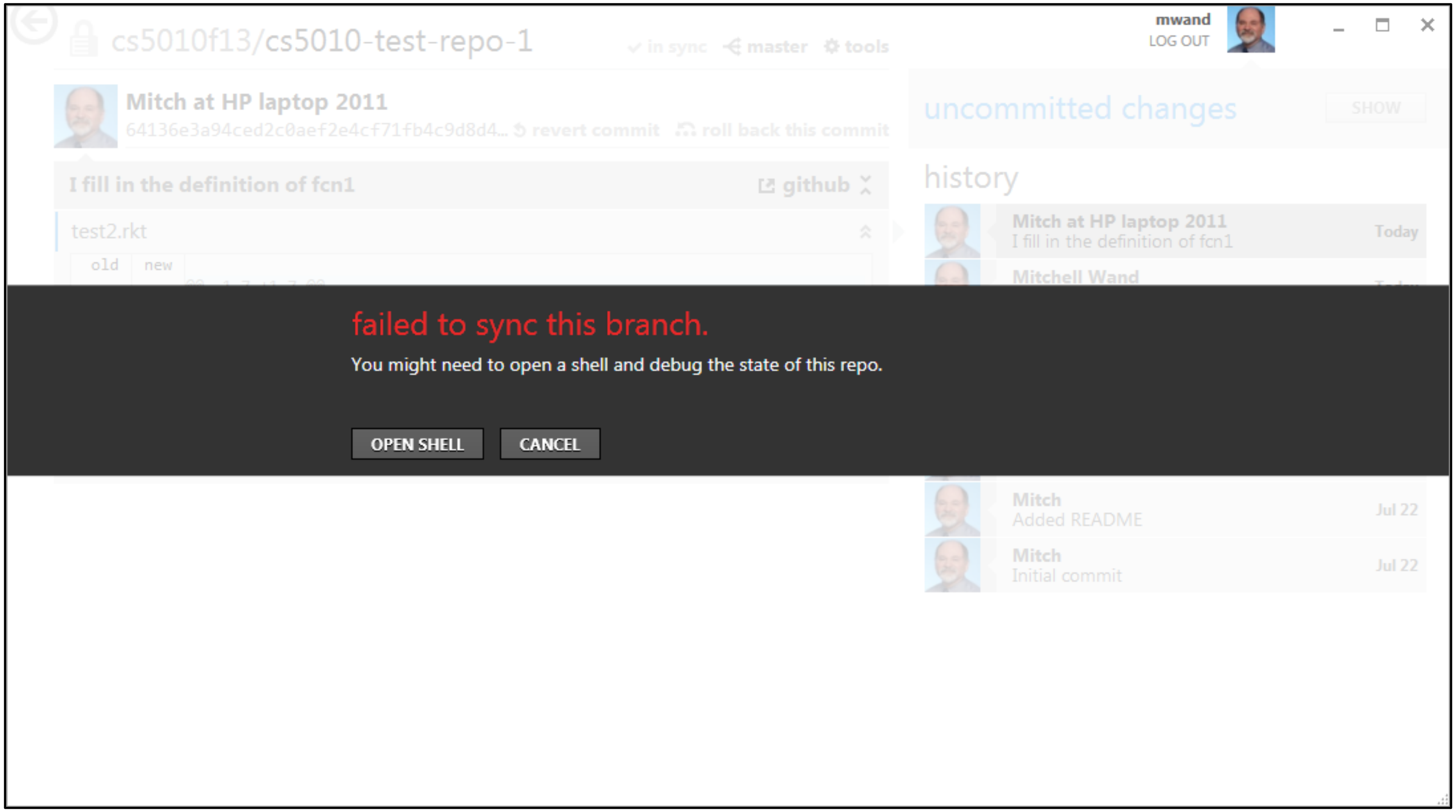
Your partner's work (on the server)

Your work (on your local machine)

Most of the time, this works well

- So long as you and your partner are working on separate parts of the file, this works fine.
- Both sets of changes get made, and the history on the server stays linear.
- But what happens if you and your partner commit incompatible changes?

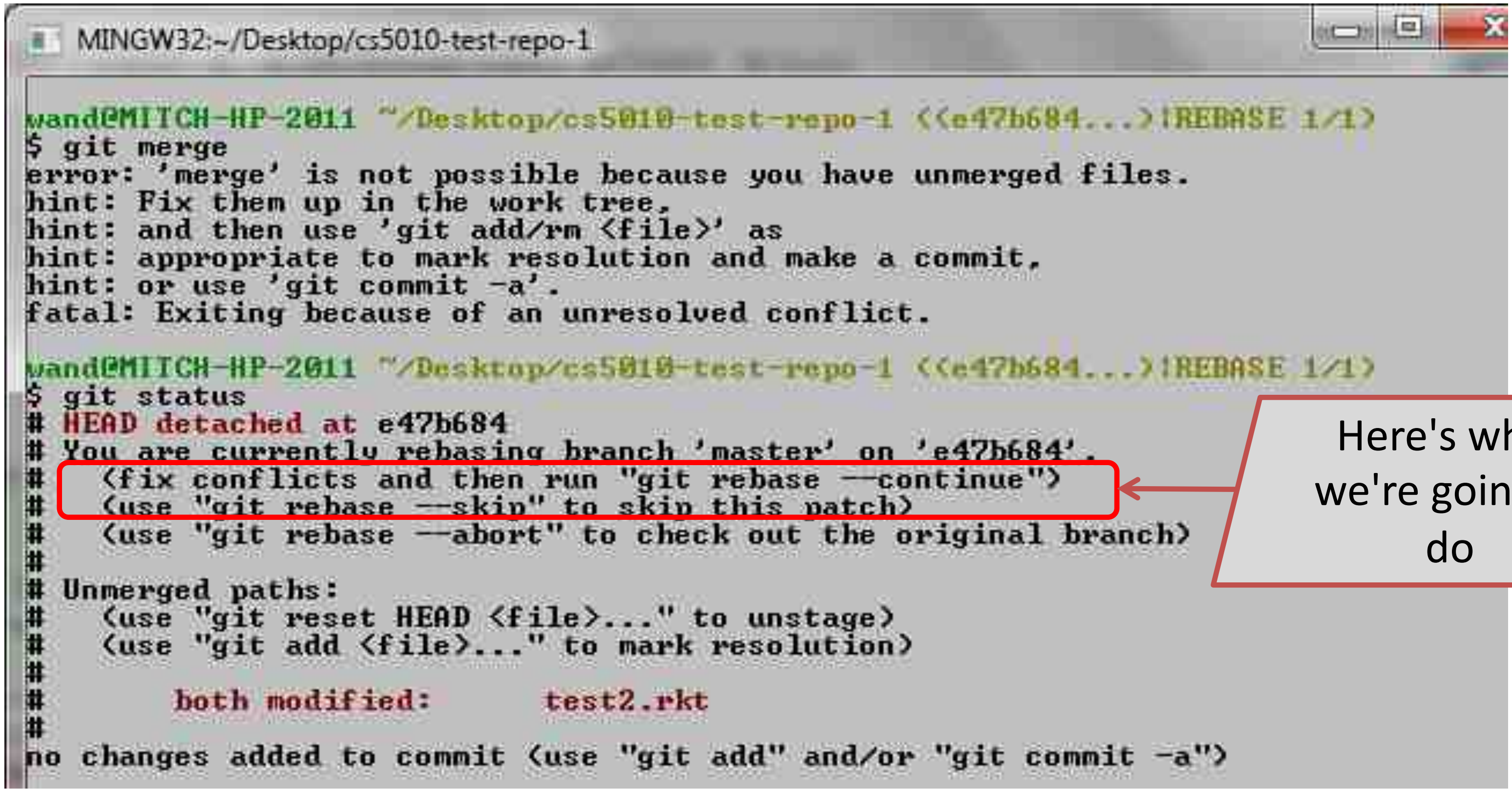
Here's what you'll see



failed to sync this branch.

You might need to open a shell and debug the state of this repo.

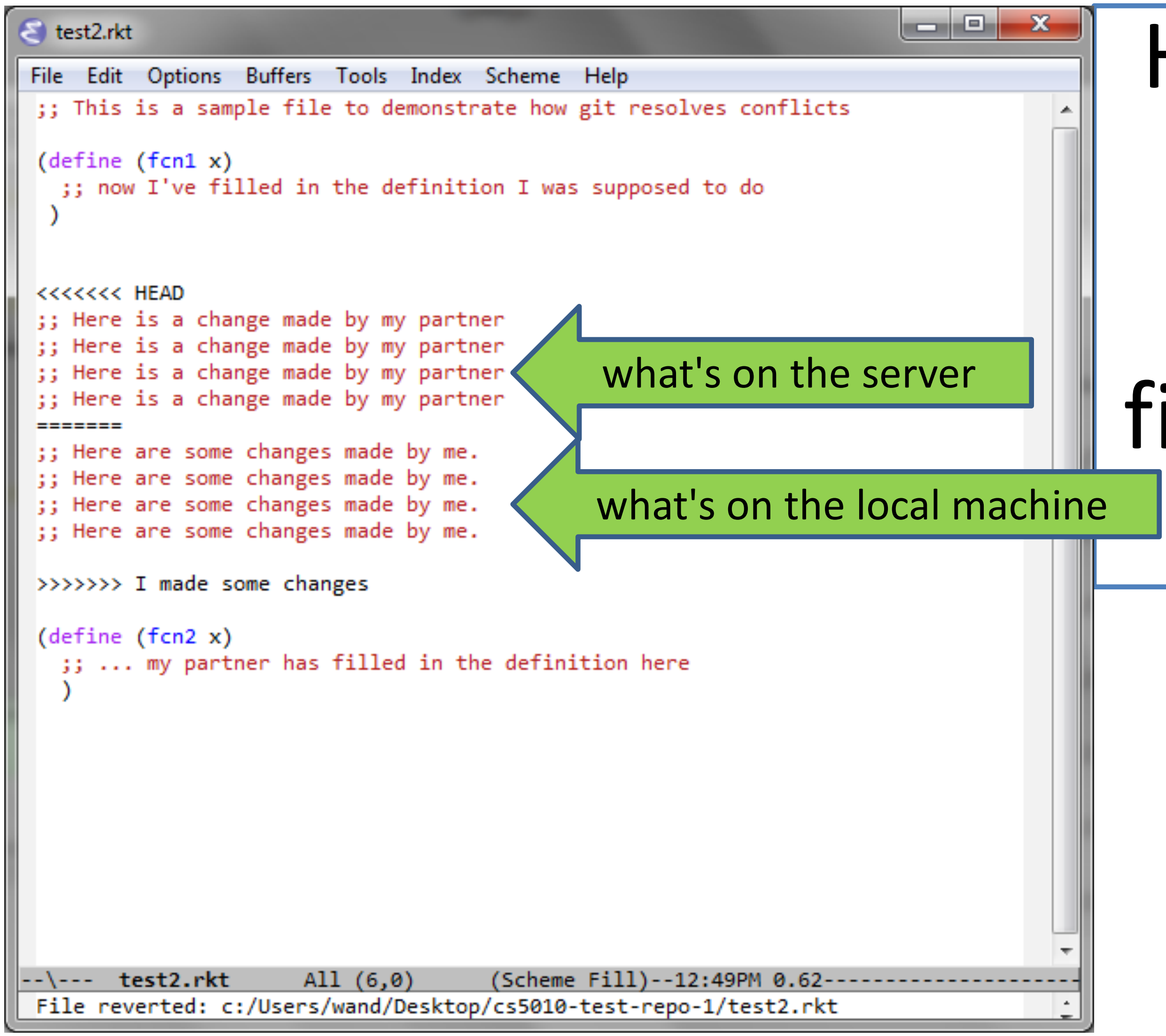
So, click on tools and open a shell



```
wand@MITCH-HP-2011 ~/Desktop/cs5010-test-repo-1 ((e47b684...)|REBASE 1/1)
$ git merge
error: 'merge' is not possible because you have unmerged files.
hint: Fix them up in the work tree,
hint: and then use 'git add/rm <file>' as
hint: appropriate to mark resolution and make a commit,
hint: or use 'git commit -a'.
fatal: Exiting because of an unresolved conflict.

wand@MITCH-HP-2011 ~/Desktop/cs5010-test-repo-1 ((e47b684...)|REBASE 1/1)
$ git status
# HEAD detached at e47b684
# You are currently rebasing branch 'master' on 'e47b684'.
#   (fix conflicts and then run "git rebase --continue")
#   (use "git rebase --skip" to skip this patch)
#   (use "git rebase --abort" to check out the original branch)
#
# Unmerged paths:
#   (use "git reset HEAD <file>..." to unstage)
#   (use "git add <file>..." to mark resolution)
#
#       both modified:      test2.rkt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Here's what we're going to do

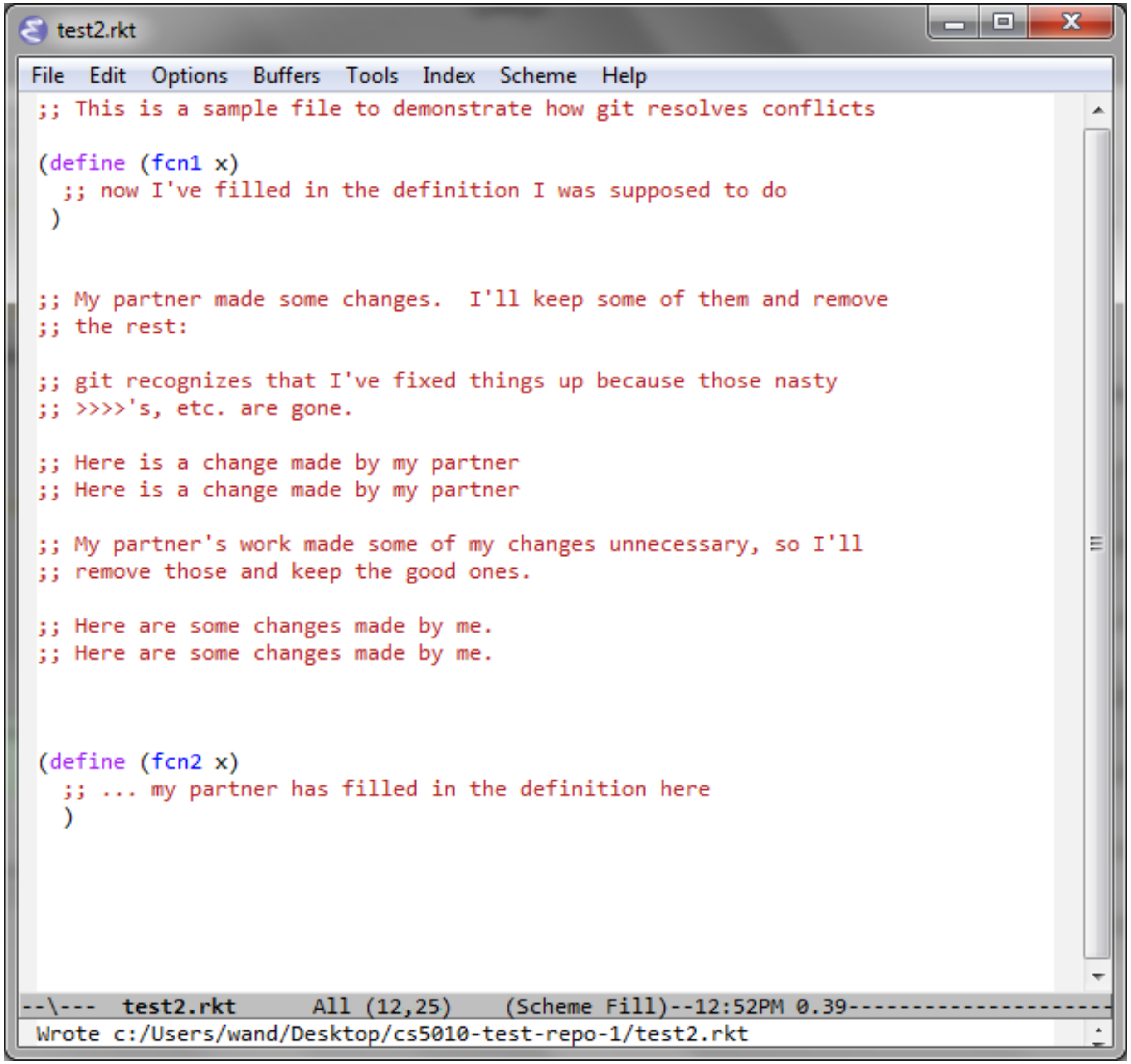
Don't panic!
First, look at the file in an editor


test2.rkt
File Edit Options Buffers Tools Index Scheme Help
what's on the server
what's on the local machine

```
;; This is a sample file to demonstrate how git resolves conflicts

(define (fcn1 x)
  ;; now I've filled in the definition I was supposed to do
 )


<<<<<<< HEAD
;; Here is a change made by my partner
;; Here is a change made by my partner
;; Here is a change made by my partner
;; Here is a change made by my partner
=======
;; Here are some changes made by me.
;; Here are some changes made by me.
;; Here are some changes made by me.
;; Here are some changes made by me.

>>>>>>> I made some changes

(define (fcn2 x)
  ;; ... my partner has filled in the definition here
  )
```

```
--\---  test2.rkt      All (6,0)      (Scheme Fill)--12:49PM 0.62--------------------
File reverted: c:/Users/wand/Desktop/cs5010-test-repo-1/test2.rkt
```

Here's what the conflicted file looks like (in emacs)

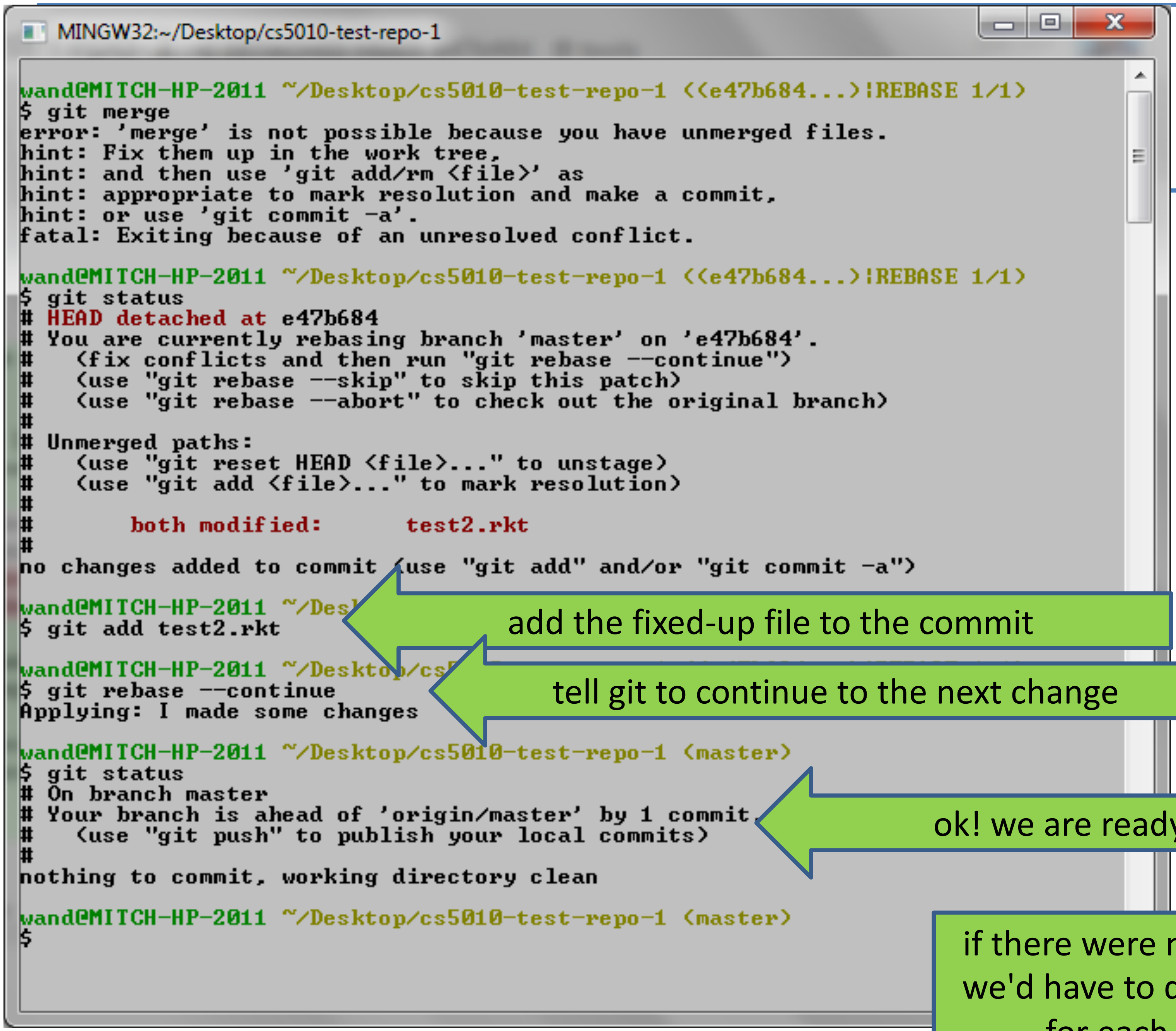


```
;; This is a sample file to demonstrate how git resolves conflicts

(define (fcn1 x)
  ;; now I've filled in the definition I was supposed to do
 )


;; My partner made some changes.  I'll keep some of them and remove
;; the rest:

;; git recognizes that I've fixed things up because those nasty
;; >>>>'s, etc. are gone.

;; Here is a change made by my partner
;; Here is a change made by my partner

;; My partner's work made some of my changes unnecessary, so I'll
;; remove those and keep the good ones.

;; Here are some changes made by me.
;; Here are some changes made by me.



(define (fcn2 x)
  ;; ... my partner has filled in the definition here
  )
```

```
--\---  test2.rkt      All (12,25)    (Scheme Fill)--12:52PM 0.39--------------------
Wrote c:/Users/wand/Desktop/cs5010-test-repo-1/test2.rkt
```

Next: edit the file the way you want it

no more >>>'s


MINGW32:~/Desktop/cs5010-test-repo-1

```
wand@MITCH-HP-2011 ~/Desktop/cs5010-test-repo-1 ((e47b684...)|REBASE 1/1)
$ git merge
error: 'merge' is not possible because you have unmerged files.
hint: Fix them up in the work tree,
hint: and then use 'git add/rm <file>' as
hint: appropriate to mark resolution and make a commit,
hint: or use 'git commit -a'.
fatal: Exiting because of an unresolved conflict.

wand@MITCH-HP-2011 ~/Desktop/cs5010-test-repo-1 ((e47b684...)|REBASE 1/1)
$ git status
# HEAD detached at e47b684
# You are currently rebasing branch 'master' on 'e47b684'.
#   (fix conflicts and then run "git rebase --continue")
#   (use "git rebase --skip" to skip this patch)
#   (use "git rebase --abort" to check out the original branch)
#
# Unmerged paths:
#   (use "git reset HEAD <file>..." to unstage)
#   (use "git add <file>..." to mark resolution)
#
#       both modified:      test2.rkt
#
no changes added to commit (use "git add" and/or "git commit -a")

wand@MITCH-HP-2011 ~/Des
$ git add test2.rkt

wand@MITCH-HP-2011 ~/Desktop/cs
$ git rebase --continue
Applying: I made some changes

wand@MITCH-HP-2011 ~/Desktop/cs5010-test-repo-1 (master)
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean

wand@MITCH-HP-2011 ~/Desktop/cs5010-test-repo-1 (master)
$
```

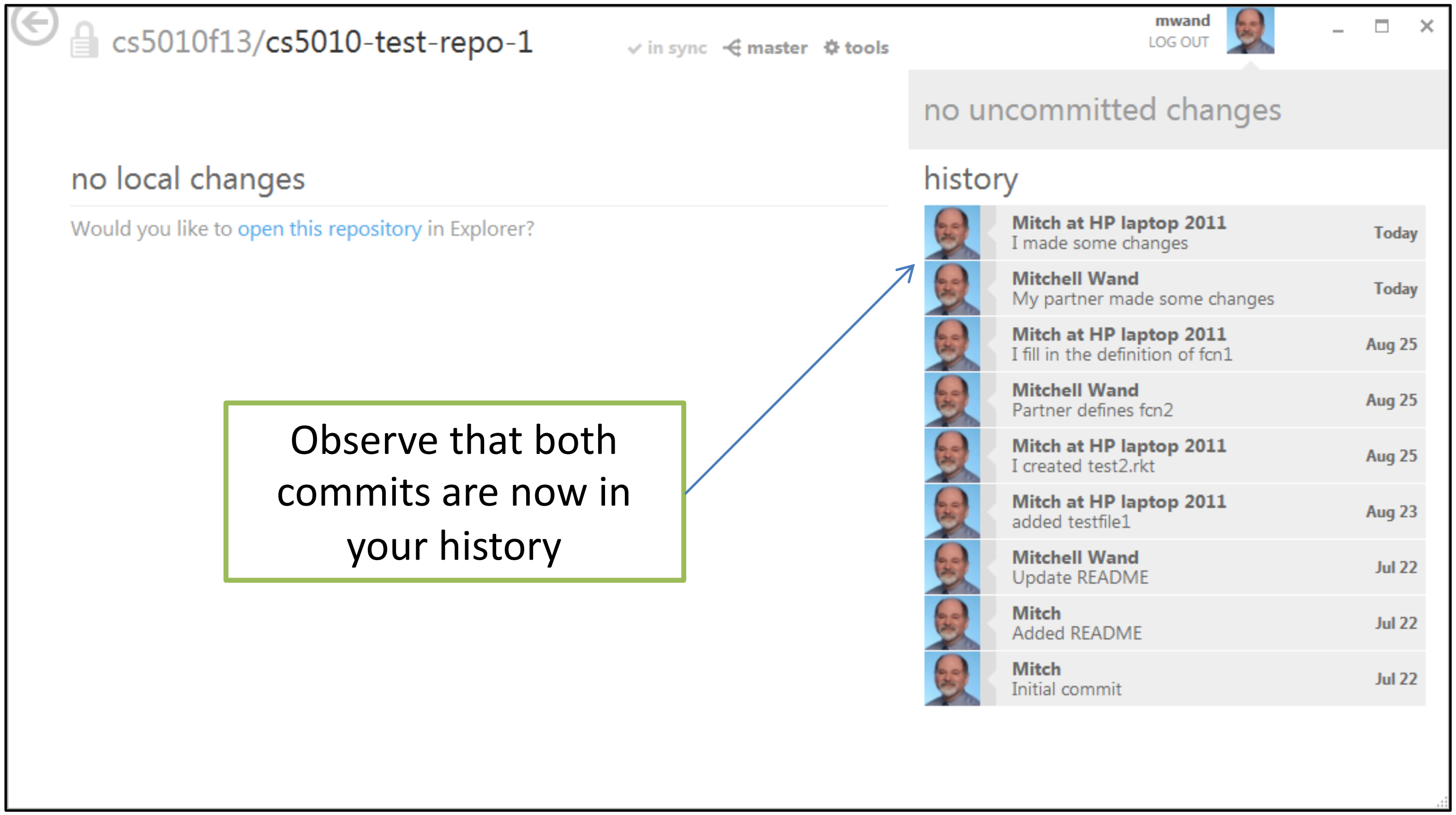
add the fixed-up file to the commit

tell git to continue to the next change

ok! we are ready to sync

if there were more conflicts, we'd have to do this process for each of them.

And we're ready to get back to work



Would you like to open this repository in Explorer?

Mitch at HP laptop 2011 — I made some changes — Today
Mitchell Wand — My partner made some changes — Today
Mitch at HP laptop 2011 — I fill in the definition of fcn1 — Aug 25
Mitchell Wand — Partner defines fcn2 — Aug 25
Mitch at HP laptop 2011 — I created test2.rkt — Aug 25
Mitch at HP laptop 2011 — added testfile1 — Aug 23
Mitchell Wand — Update README — Jul 22
Mitch — Added README — Jul 22
Mitch — Initial commit — Jul 22

Observe that both commits are now in your history

Is this a pain?

- Yes, but it shouldn't happen too often.
- Your interaction with the shell might look somewhat different.
- But the workflow is the same:
  - identify the files that are conflicted
  - identify and resolve the conflicts in each file
    - the conflicted region will be marked with >>>'s.
    - Use your favorite text editor for this.
  - When you get the file the way you want it, add it to your commit.
  - Commit all the fixed-up files.

Summary

- In this lesson you have learned
  - what happens when you pull changes from an upstream repository
  - what a conflict is
  - how to resolve a simple merge conflict in a text file (including a .rkt file)