Merge Practice

Practice resolving conflicts in git.

Exercise

We will deliberately create a **conflict** by editing the README.md in two copies of same repo:

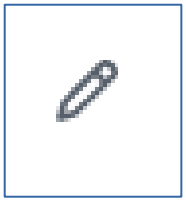
1. **Github** using Github's online editor
2. **Your local repo** using any editor

Then **discover** the differences and **resolve them**.

1. Make a change on Github

Use your **git-commands** repo on **Github**

1. On **Github**, edit README.md using Github's online editor.



Add these three lines at the top:

```
# The Ultimate Guide to Using Git


***For The Impatient.***
```

2. Write a ***descriptive*** *commit message* and **Commit** the change (green **`Commit`** button).

2. Make a different change in your local repo

1. In your **local** **git-commands** **repo**, edit **`README.md`** (in your *working copy*)

Add this title at the top of file:

```
# Quick Guide to Git


By Your Name
```

2. Make some other change in README.md (*anything*).

3. Commit the changes to your local repo.

```
git add -u
git commit -m "Add a title to README"
```

3. Check Status

`git status`

It should return:

What is **origin/master** ???
Where is it ???

```
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 tree clean
```

'**origin/master**' is a short form for '**remotes/origin/master**'

4. What did I change?

You can compare all the changes between your local "`master`" branch and the tracking branch "`origin/master`".

Syntax: **`git diff`** `commit1` `commit2`

where `commit1` & `commit2` refer to any commits.

```
> git diff master origin/master
```

- It should show what you changed locally.
- Does `diff` show the title line you added on Github?

5. push your work

Since your local copy is *ahead* of origin/master, **push it**.

```
> git push
```

What happens?

```
Remote repository contains work that you do not
have! [rejected]   master -> master (fetch
first)

error: failed to push some refs to
'https://github.com/...'

hint: Updates were rejected because the remote
contains work that you do not have locally.
```

This is Good!

This is actually a **good thing**.

Git won't let you accidentally **overwrite** work that someone else has added to the repository.

Git forces you to **merge** the differences first.

6. What has changed on "origin"?

Let's see what has changed on "origin"...

Update your "tracking branch" for **`origin/master`**:

```
git fetch
```

*This **fetches** the latest revisions from the remote* origin *into your tracking branch (**`origin/master`**) but does **not** merge them into your local branch (**`master`**).*

It does not change your working copy either.

7. compare branches (again)

Show a graph of all branches, including tracking branches:

```
gitk --all
```

View the differences between all files in a terminal window:

```
git diff master origin/master
```

Does "git diff" show the differences in the title line now?

Where did "fetch" put the updates?

"**`git fetch`**" downloads changes from **`origin`** (Github).

Where did git save these updates?

[ ] In your working copy

[ ] In the `master` branch

[ ] In the `origin/master` tracking branch

"diff" format

diff is a standard command that shows differences in a standard format. It can be used to create patches, too!

```
diff --git a/README.md b/README.md
index 21b69e8..09b0702 100644
--- a/README.md  ('a' = local version)
+++ b/README.md  ('b' = remote version)
@@ -1,6 +1,6 @@
-# Quick Guide to Git
+# The Ultimate Guide to Using Git

-By Harry Hacker
+***For the impatient.***
```

8. merge the two versions

Use "git merge" to automatically merge branches.

cmd> **`git merge --no-commit`**

```
Auto-merging README.md
CONFLICT (content): Merge conflict in
README.md
Automatic merge failed; fix conflicts and
then commit the result.
```

`--no-commit` gives you a chance to review the results, even if automatic merge succeeds.

9. Edit and fix conflicts

Open an editor and examine the result of all files that contained merged lines.

Part(s) containing a conflict will look like this:

```
<<<<<<<<< HEAD
The text from your local repository
=========
Conflicting text from the remote version
>>>>>>>>> refs/remotes/origin/master
```

Note: Auto-merge may create bugs by successfully merging parts of code that are incompatible! Always test code after merge.

Fix conflicts yourself

You must decide which conflicting lines to keep and which to discard.

- Keep lines you want, delete the others.
- Delete the merge markers ("<<<<", ">>>>", "====").

```
<<<<<<<<< HEAD
The text from your local version
=========
Conflicting text from the remote version
>>>>>>>>> refs/remotes/origin/master
```

10. Mark Conflict as Resolved

Use "git status" to see that there is a conflict

```
cmd> git status
Unmerged paths:
  (use "git add <file>..." to mark resolution)
both modified:   README.md
```

When you are satisfied that file is fixed, then...

```
cmd> git add README.md
```

```
cmd> git commit
```

```
(You should write a good commit message
explaining the merge.)
```

```
cmd> git push
```

I Give Up!

If the merge creates too many conflicts to fix, you can "undo" the merge and try something else.

```
cmd> git merge --abort
```

Graphical Merge Tools

Graphical tools show the differences side-by-side.

1. IDE visual merge feature

2. Graphical mergetool (enter: "git help mergetool").

3. Merge on Github

   - ok if there are no conflicts (e.g. fast-forward merge)

   - for conflicts, merge on your own computer so you can run tests before committing.

End Notes

Optional material you can ignore.

Understanding diffs

"**`diff`**" is a Unix command to show differences between text files. It shows:

- lines **changed** (differences)
- lines **added** in one file
- lines **deleted** in one file

diff may show surrounding identical lines for ***context,*** to make it easier to identify the "diff" in code.

Example: make 2 copies of a text file. Change one copy (add lines, change lines, delete lines). Run diff:

```
cmd>  diff  a.txt  b.txt
```

Diffs on Github

(Demo in class.)

Click on "commits" link on a repository.

Find an interesting commit and click the hash (6b57...)

Github shows changes from previous commit.

"git pull" = "git fetch" + "git merge"

"git pull" performs two commands:

`git fetch` - fetch updates from a remote repository.
It saves the remote in a separate branch named:

`origin/master` or `origin/branchname`

`git merge` - merge two development histories.

If you don't specify which branches to merge,

the default is HEAD and origin/tracking_branch_name

git fetch and diff

It is safer to use "**`git fetch`**" instead of "**`git pull`**"

1. fetch the remote branch: **`git fetch`**
2. in your local repo, the branch you just fetched is named **`origin/master`** or **`origin/branch-name`**
3. view differences between working copy and remote:

   `git diff origin/master`

   == or ==

4. view differences between local HEAD and remote:

   `git diff HEAD origin/master`

Visual Merge Tools

You can use a graphical diff viewer to both view and resolve differences. It is easier to comprehend.

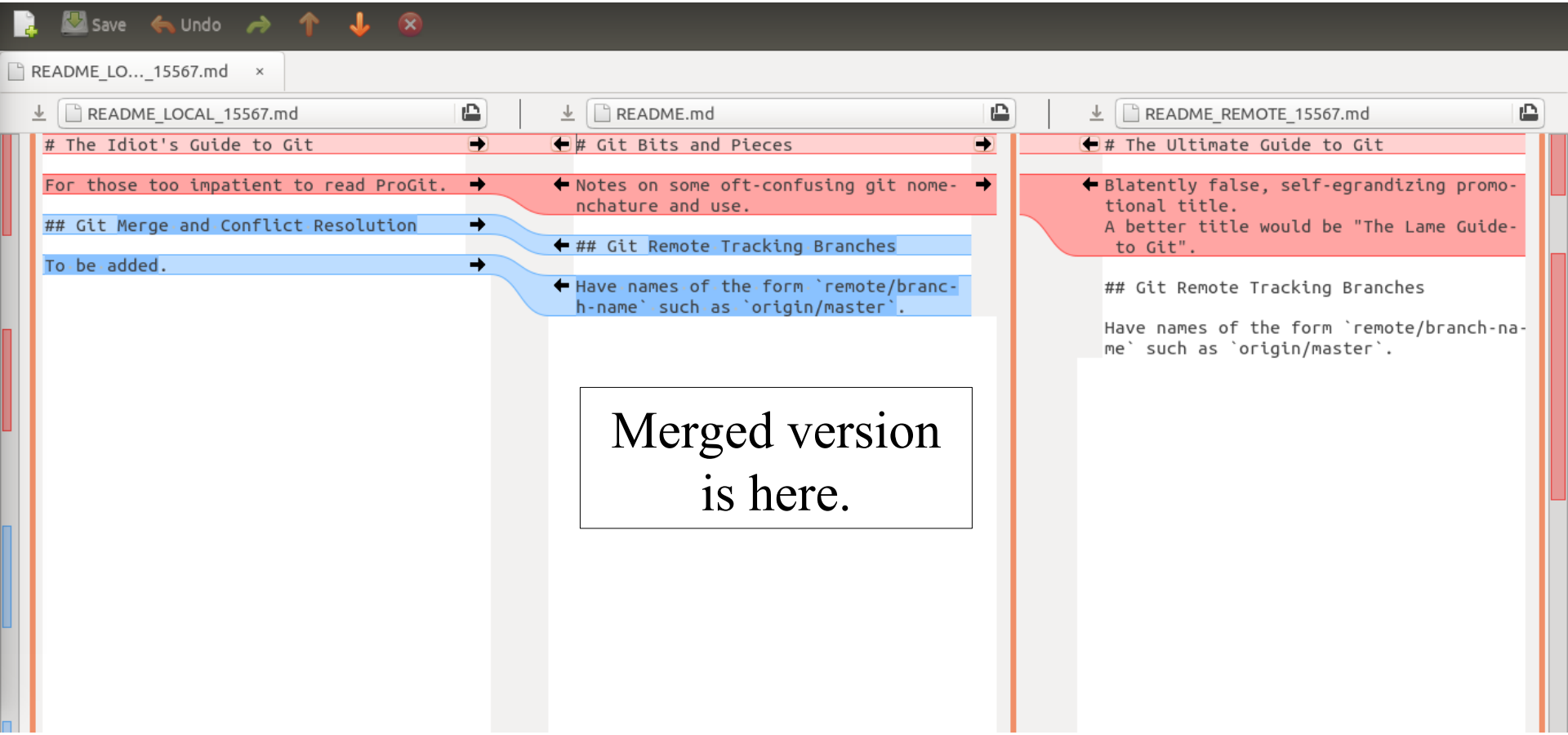
meld and diffuse are good tools known by git.

```
cmd> git help mergetool
```

I use meld or diffuse. Example:

```
cmd> git mergetool --tool=meld
```

cmd> **git mergetool --tool=meld**



Common Cause of Conflicts

1. Developer A **clones** a repo from Github, or "**pulls**" latest rev from Github. Now he is up to date!
2. Developer A starts work on his local copy.
3. Developer B **pushes** a change to some files in the **same repo** to Github.
4. Developer A commits his work and does "**git push**".

What Happens?

What Happens?

```
dev-A> git commit -m "add tests for ..."

dev-A> git push
```

```
! [rejected]   master -> master (fetch first)
error: failed to push some refs to https://github.com/...
hint: Updates were rejected because the remote contains
work that you do not have locally. This is usually caused
by another repository pushing to the same ref.
You may want to first integrate the remote changes
(e.g., 'git pull ...') before pushing again.
```