

Merge Practice

Practice resolving a conflict between revisions.

A Common Problem

1. **Developer A** clones a repo from Github, or "pulls" latest rev from github. Now his local copy 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" to Github.

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.

Exercise

We will deliberately create a conflict by editing the same file (README.md) on:

Github using browser-based editor
in your local repo

Then try to discover the differences and resolve them.

Exercise: create a conflict

Use your **git-commands** repository (any repo with remote on Github is OK):

1. On **Github**, edit README.md using Github's in-browser editor.

Add some lines near the top. Example:

```
*The Ultimate Guide to Using Git*  
For the impatient.
```

Commit the change (green **Commit** button)

2. In your **local repo**, edit README.md

Make some different changes near the top

Commit

Edit again, change some more, and commit

Exercise: try to push

3. Enter this command: `git push`

What happens?

What is the message from git?

Exercise: fetch remote changes

4. Update your "tracking branch" for origin/master:

```
git fetch
```

Exercise: compare branches

5. See a graph of the differences: `gitk --all`
6. View the differences line-by-line (this is also shown in gitk when you click on a file in the commit):

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


"diff" format

diff is a standard Linux/Unix command that shows diffs in a standard format. Can be used to create patches, too!

```
diff --git a/README.md b/README.md  
index 21b69e8..09b0702 100644  
--- a/README.md  
+++ b/README.md  
@@ -1,10 +1,8 @@  
-Text from version a of file (local version)  
-  
+Text from version b of file (remote version)  
---
```

Exercise: merge differences

And hope you are lucky! It may not work.

```
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.

Exercise: edit and fix conflicts

7. Open an editor and examine the result of all files that contained differences, even if auto-merge succeeds.

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

```
<<<<<<<<< HEAD
```

```
The text from your local version
```

```
=====
```

```
Conflicting text from the remote version
```

```
>>>>>>>>> refs/remotes/origin/master
```

Note: Sometimes auto-merge creates bugs by successfully merging parts that are incompatible!

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
```

If you are satisfied that it is fixed, then...

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

```
cmd> git commit
```

```
(git opens an editor. You should write a good  
commit message explaining the merge.)
```

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
```

End Notes

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
```

"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" first.

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's easier to comprehend.

IDE: Eclipse, IntelliJ, VS Code

Tools: Meld, Diffuse

meld and **diffuse** are good tools known by git.

```
cmd> git help mergetool
```