Conflicts

As soon as people can work in parallel, they'll likely step on each other's toes. This will even happen with a single person: if we are working on a piece of software on both our laptop and a server in the lab, we could make different changes to each copy. Version control helps us manage these conflicts by giving us tools to resolve overlapping changes.

To see how we can resolve conflicts, we must first create one. The file `mars.txt` currently looks like this in both partners' copies of our `planets` repository:

```
$ cat mars.txt
```

Output:

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
```

Let's add a line to the collaborator's copy only:

```
$ nano mars.txt
$ cat mars.txt
```

Output:

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
This line added to Wolfman's copy
```

and then push the change to GitHub:

```
$ git add mars.txt
$ git commit -m "Add a line in our home copy"
```

Output:

```
[master 5ae9631] Add a line in our home copy
 1 file changed, 1 insertion(+)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 331 bytes | 331.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/vlad/planets.git
   29aba7c..dabb4c8  master -> master
```

Now let’s have the owner make a different change to their copy *without* updating from GitHub:

```
$ nano mars.txt
$ cat mars.txt
```

Output:

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
We added a different line in the other copy
```

We can commit the change locally:

```
$ git add mars.txt
$ git commit -m "Add a line in my copy"
```

Output:

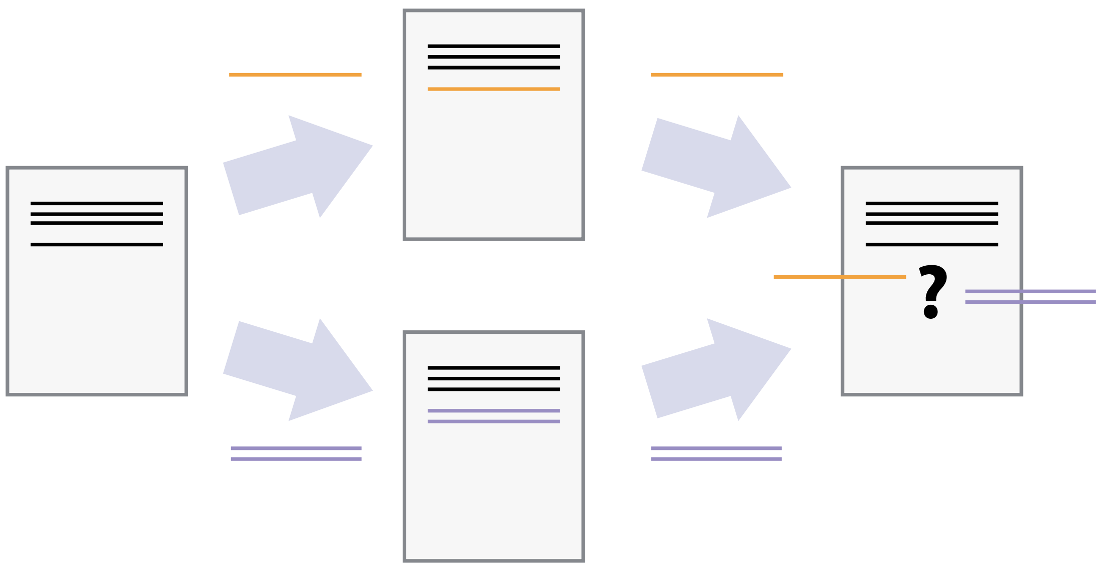
```
[master 07ebc69] Add a line in my copy
 1 file changed, 1 insertion(+)
```

but Git won’t let us push it to GitHub:

```
$ git push origin master
```

Output:

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



Git rejects the push because it detects that the remote repository has new updates that have not been incorporated into the local branch. What we have to do is pull the changes from GitHub, merge them into the copy we're currently working in, and then push that. Let's start by pulling:

```
$ git pull origin master
```

Output:

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/vlad/planets
 * branch            master     -> FETCH_HEAD
    29aba7c..dabb4c8  master     -> origin/master
Auto-merging mars.txt
CONFLICT (content): Merge conflict in mars.txt
Automatic merge failed; fix conflicts and then commit the result.
```

The `git pull` command updates the local repository to include those changes already included in the remote repository. After the changes from remote branch have been fetched, Git detects that changes made to the local copy overlap with those made to the remote repository, and therefore refuses to merge the two versions to stop us from trampling on our previous work. The conflict is marked in in the affected file:

```
$ cat mars.txt
```

Output:

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
<<<<<<< HEAD
We added a different line in the other copy
=======
This line added to Wolfman's copy
>>>>>>> dabb4c8c450e8475aee9b14b4383acc99f42af1d
```

Our change is preceded by `<<<<<<< HEAD`. Git has then inserted `=======` as a separator between the conflicting changes and marked the end of the content downloaded from GitHub with `>>>>>>>`. (The string of letters and digits after that marker identifies the commit we've just downloaded.)

It is now up to us to edit this file to remove these markers and reconcile the changes. We can do anything we want: keep the change made in the local repository, keep the change made in the remote repository, write something new to replace both, or get rid of the change entirely. Let's replace both so that the file looks like this:

```
$ cat mars.txt
```

Output:

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
We removed the conflict on this line
```

To finish merging, we add `mars.txt` to the changes being made by the merge and then commit:

```
$ git add mars.txt
$ git status
```

Output:

```
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

    modified:   mars.txt
$ git commit -m "Merge changes from GitHub"
[master 2abf2b1] Merge changes from GitHub
```

Now we can push our changes to GitHub:

```
$ git push origin master
```

Output:

```
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 645 bytes | 645.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 2 local objects.
To https://github.com/vlad/planets.git
   dabb4c8..2abf2b1  master -> master
```

Git keeps track of what we've merged with what, so we don't have to fix things by hand again when the collaborator who made the first change pulls again:

```
$ git pull origin master
```

Output:

```
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 4), reused 6 (delta 4), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/vlad/planets
 * branch            master     -> FETCH_HEAD
    dabb4c8..2abf2b1  master     -> origin/master
Updating dabb4c8..2abf2b1
Fast-forward
 mars.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

We get the merged file:

```
$ cat mars.txt
```

Output:

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
We removed the conflict on this line
```

We don't need to merge again because Git knows someone has already done that.

Git's ability to resolve conflicts is very useful, but conflict resolution costs time and effort, and can introduce errors if conflicts are not resolved correctly. If you find yourself resolving a lot of conflicts in a project, consider these technical approaches to reducing them:

- Pull from upstream more frequently, especially before starting new work
- Use topic branches to segregate work, merging to master when complete
- Make smaller more atomic commits
- Where logically appropriate, break large files into smaller ones so that it is less likely that two authors will alter the same file simultaneously

Conflicts can also be minimized with project management strategies:

- Clarify who is responsible for what areas with your collaborators
- Discuss what order tasks should be carried out in with your collaborators so that tasks expected to change the same lines won't be worked on simultaneously
- If the conflicts are stylistic churn (e.g. tabs vs. spaces), establish a project convention that is governing and use code style tools (e.g. `htmltidy`, `perltidy`, `rubocop`, etc.) to enforce, if necessary

Solving Conflicts that You Create

Clone the repository created by your instructor. Add a new file to it, and modify an existing file (your instructor will tell you which one). When asked by your instructor, pull her changes from the repository to create a conflict, then resolve it.

Conflicts on Non-textual files

What does Git do when there is a conflict in an image or some other non-textual file that is stored in version control?

Solution

Let's try it. Suppose Dracula takes a picture of Martian surface and calls it `mars.jpg`.

If you do not have an image file of Mars available, you can create a dummy binary file like this:

```
$ head -c 1024 /dev/urandom > mars.jpg
$ ls -lh mars.jpg
```

Output:

```
-rw-r--r-- 1 vlad 57095 1.0K Mar  8 20:24 mars.jpg
```

`ls` shows us that this created a 1-kilobyte file. It is full of random bytes read from the special file, `/dev/urandom`.

Now, suppose Dracula adds `mars.jpg` to his repository:

```
$ git add mars.jpg
$ git commit -m "Add picture of Martian surface"
```

Output:

```
[master 8e4115c] Add picture of Martian surface
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 mars.jpg
```

Suppose that Wolfman has added a similar picture in the meantime. His is a picture of the Martian sky, but it is *also* called `mars.jpg`. When Dracula tries to push, he gets a familiar message:

```
$ git push origin master
```

Output:

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

We've learned that we must pull first and resolve any conflicts:

```
$ git pull origin master
```

When there is a conflict on an image or other binary file, git prints a message like this:

```
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://github.com/vlad/planets.git
 * branch            master     -> FETCH_HEAD
   6a67967..439dc8c  master     -> origin/master
warning: Cannot merge binary files: mars.jpg (HEAD vs.
439dc8c08869c342438f6dc4a2b615b05b93c76e)
Auto-merging mars.jpg
CONFLICT (add/add): Merge conflict in mars.jpg
Automatic merge failed; fix conflicts and then commit the result.
```

The conflict message here is mostly the same as it was for `mars.txt`, but there is one key additional line:

```
warning: Cannot merge binary files: mars.jpg (HEAD vs.
439dc8c08869c342438f6dc4a2b615b05b93c76e)
```

Git cannot automatically insert conflict markers into an image as it does for text files. So, instead of editing the image file, we must check out the version we want to keep. Then we can add and commit this version.

On the key line above, Git has conveniently given us commit identifiers for the two versions of `mars.jpg`. Our version is `HEAD`, and Wolfman's version is `439dc8c0...`. If we want to use our version, we can use `git checkout`:

```
$ git checkout HEAD mars.jpg
$ git add mars.jpg
$ git commit -m "Use image of surface instead of sky"
```

Output:

```
[master 21032c3] Use image of surface instead of sky
```

If instead we want to use Wolfman's version, we can use `git checkout` with Wolfman's commit identifier, `439dc8c0`:

```
$ git checkout 439dc8c0 mars.jpg
$ git add mars.jpg
$ git commit -m "Use image of sky instead of surface"
```

Output:

```
[master da21b34] Use image of sky instead of surface
```

We can also keep *both* images. The catch is that we cannot keep them under the same name. But, we can check out each version in succession and *rename* it, then add the renamed versions. First, check out each image and rename it:

```
$ git checkout HEAD mars.jpg
$ git mv mars.jpg mars-surface.jpg
$ git checkout 439dc8c0 mars.jpg
$ mv mars.jpg mars-sky.jpg
```

Then, remove the old `mars.jpg` and add the two new files:

```
$ git rm mars.jpg
$ git add mars-surface.jpg
$ git add mars-sky.jpg
$ git commit -m "Use two images: surface and sky"
```

Output:

```
[master 94ae08c] Use two images: surface and sky
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 mars-sky.jpg
rename mars.jpg => mars-surface.jpg (100%)
```

Now both images of Mars are checked into the repository, and `mars.jpg` no longer exists.

A Typical Work Session

You sit down at your computer to work on a shared project that is tracked in a remote Git repository. During your work session, you take the following actions, but not in this order:

- *Make changes* by appending the number `100` to a text file `numbers.txt`
- *Update remote* repository to match the local repository
- *Celebrate* your success with some fancy beverage(s)
- *Update local* repository to match the remote repository
- *Stage changes* to be committed
- *Commit changes* to the local repository

In what order should you perform these actions to minimize the chances of conflicts? Put the commands above in order in the *action* column of the table below. When you have the order right, see if you can write the corresponding commands in the *command* column. A few steps are populated to get you started.

order	action	command . . . .
1		
2		`echo 100 >> numbers.txt`
3		
4		
5		
6	Celebrate!	`AFK`

Solution

order	action	command . . . .
1	Update local	`git pull origin master`
2	Make changes	`echo 100 >> numbers.txt`
3	Stage changes	`git add numbers.txt`
4	Commit changes	`git commit -m "Add 100 to numbers.txt"`
5	Update remote	`git push origin master`
6	Celebrate!	`AFK`

Key Points

- Conflicts occur when two or more people change the same lines of the same file.
- The version control system does not allow people to overwrite each other’s changes blindly, but highlights conflicts so that they can be resolved.