Demonstration Details of Git and Github

Zhe "Tim" Wu

0. Preparation

We are using standard Git Bash for this demonstration. Any third-party Git tools (such as GitHub Desktop) may also work with the following contents.

0.1 Remote repo preparation

On Github.com, create a repo named `gitdemo` under your account. Please do NOT select `README.md` or any `.gitignore` file to prevent possible conflicts.

0.2 Local repo preparation

Create a local file folder named `gitdemo` with a empty text file named `tfdemo.py`. Initialize this folder as a git repo with command `git init`.

Type the following content in `tfdemo.py` with any text editor at your preference:

```
import tensorflow as tf

tf.print("Hello World")
```

Use `git add` and `git commit` to commit this modification.

Last use the following command to link the remote and local repo, then push your initial commit to the remote repo:

```
$ git remote add origin git@github.com:<YourAccountName>/gitdemo.git

$ git push -u origin master
```

Note that the `-u` argument for `git push` is for setting the upstream, which is only needed for your first push.

1. Time Machine - How to "regret" with Git

1.1 Check the history of modifications

The command `git log` is for checking the history.

```
$ git log
commit 9cc4a6f23d3cee06a74bb76df7b4365ef0bda04a (HEAD -> master, origin/master)
```

```
Author: Tim Wu <tim.wuzhe@gmail.com>
Date:   Tue May 26 12:58:19 2020 -0400

    Initial commit
```

This is quite verbose. We may also use some parameters to abbreviate it:

```
$ git log --pretty=oneline
9cc4a6f23d3cee06a74bb76df7b4365ef0bda04a (HEAD -> master, origin/master) Initial
commit
```

1.2 Revoke your modifications

Suppose it is 2 am and you just finished your very last line to the file `tfdemo.py`, saved the file in the text editor and went to bed.

```
tf.print("MRI research is not very interesting.")
```

After woke up in the morning you realized that you wrote something stupid and you don't want your boss to see it. Fortunately Git provides you an opportunity to regret.

1.2.1 Your modification is in your work directory (before `git add`)

Before add your modification into stage (cache), you have two options (1) Manually change the file back in text editor, or (2) Use `git checkout -- <filename>`:

You can see that the `tfdemo.py` has been modified but the modification has not been added to the cache:

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   tfdemo.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Use `git checkout -- <filename>` to replace the file with the one from the committed version.

```
$ git checkout -- tfdemo.py
```

Now the unwanted line is disappeared and the file has been changed back to your previously committed version.

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

1.2.2 Your modification is in stage (before `git commit` but after `git add`)

In this case, `git reset HEAD <filename>` should be used. Note that this step is **returning** the modifications back to the working directory. If you want to discard it, use `git checkout -- <filename>` after that.

```
$ git add *

$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   tfdemo.py
```

After we used the `git reset` command:

```
$ git reset HEAD tfdemo.py
Unstaged changes after reset:
M       tfdemo.py

$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   tfdemo.py

no changes added to commit (use "git add" and/or "git commit -a")
```

1.2.3 Your modification is in blob (before `git push` but after `git commit`)

This is very unfortunate but there is still a way to save you! Use `git reset --hard <SHA1>` command to replace the last visible commit with the one with the hash code you designated. Use `git log` to search for the hash code you want.

```
$ git log --pretty=oneline
d59619e231b398c8ff730276beee9f6c62869d64 (HEAD -> master) First modification
9cc4a6f23d3cee06a74bb76df7b4365ef0bda04a (origin/master) Initial commit

$ git reset --hard 9cc4a
HEAD is now at 9cc4a6f Initial commit
```

1.2.4 Your modification has been pushed to the remote repo

This is the most extreme case that someone (could be your boss) has already seen your line. If you still want to save it, you should use `git reset` on your local repo to revoke your modifications. Your local repo status will be:

```
$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

Then use `git push --force origin master`. We have to use `--force` parameters to push our local repo that falls behind the remote one.

```
git push --force origin master
Total 0 (delta 0), reused 0 (delta 0)
To github.com:nbwuzhe/gitdemo
 + d59619e...9cc4a6f master -> master (forced update)
```

And check the history and log

```
$ git log
commit 9cc4a6f23d3cee06a74bb76df7b4365ef0bda04a (HEAD -> master, origin/master)
Author: Tim Wu <tim.wuzhe@gmail.com>
Date:   Tue May 26 21:58:19 2020 -0400

    Initial commit

$ git status
On branch master
Your branch is up to date with 'origin/master'.
```

```
nothing to commit, working tree clean
```

1.3 How to "revoke" your revocations

If you want to restore the newer commit that has been previously revoked:

(1) If you still have the previous hash code output of `git log`, use `git reset` to restore your newer commit.

(2) If you turned off your Git Bash. There is a command called `git reflog` to show your previous command history. You will find the hash code before you execute `git reset --hard <SHA1>` and use this command again to restore the newer commit.

```
$ git reflog
9cc4a6f (HEAD -> master, origin/master) HEAD@{0}: reset: moving to 9cc4a
d59619e HEAD@{1}: reset: moving to d59619
9cc4a6f (HEAD -> master, origin/master) HEAD@{2}: reset: moving to 9cc4a
d59619e HEAD@{3}: commit: First modification
9cc4a6f (HEAD -> master, origin/master) HEAD@{4}: commit (initial): Initial commit
```

2. Operations of Branch

Suppose there are two radiologists diagnosing patient based on his/her MR examination results and they are giving totally opposite diagnosis. They are asking a senior radiologist for the final decision.

2.1 Creating a branch and switch to the new branch

Each radiologist creates their own branch before writing their diagnosis.

(1) The first radiologist, giving a positive diagnosis:

```
$ git checkout -b positive
Switched to a new branch 'positive'

$ git branch
  master
* positive
```

Then add a line to the file `tfdemo.py`:

```
tf.print("The MR result indicates a brain tumor.")
```

Add and commit the modification:

```
$ git add tfdemo.py

$ git commit -m "A tumor detected"
```

(2) The second radiologist, giving a negative diagnosis:

We want both `positive` and `negative` branches to be **based** on `master` branch.

```
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

$ git checkout -b negative
Switched to a new branch 'negative'

$ git branch
  master
* negative
  positive
```

Then we add the diagnosis:

```
tf.print("This patient is normal.")
```

Add and commit the modification:

```
$ git add tfdemo.py

$ git commit -m "Patient is normal"
```

2.2 Merging the branches and resolving conflicts

The senior radiologist determines that the patient is negative.

We need to merge the `positive` branch into `negative`, then merge `negative` into `master`. Make sure you are currently in `negative` branch, then use `git merge <branch_name_to_come_in>`

```
$ git branch
  master
* negative
  positive

$ git merge positive
Auto-merging tfdemo.py
```

```
CONFLICT (content): Merge conflict in tfdemo.py
Automatic merge failed; fix conflicts and then commit the result.
```

Ops! We are encountering a conflict that the merge cannot proceed. We have to resolve it first. Open the file `tfdemo.py`, it shows:

```
import tensorflow as tf

tf.print("Hello World")

<<<<<<< HEAD
tf.print("This patient is normal.")
=======
tf.print("The MR result indicates a brain tumor.")
>>>>>>> positive
```

The <<<, ===, and >>> marks the conflicting part. Note that the part above === is the one in the current branch (i.e. `negative`); the part below === is the one from the branch that tries to merge in (i.e. `positive`). Since the senior radiologist determines the patient is normal, we delete the unwanted one.

```
import tensorflow as tf

tf.print("Hello World")

tf.print("This patient is normal.")
```

Then we add and commit the change.

```
$ git add *

$ git commit -m "Merge from negative"
[negative d063b06] Merge from negative
```

We then switch to `master` branch, and merge the branch `negative` to `master` to make it as the final result.

```
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

$ git merge negative
Updating 9cc4a6f..d063b06
Fast-forward
 tfdemo.py | 6 ++++++
 1 file changed, 6 insertions(+)
```

This time the merge proceeds smoothly.

We can check the history of our commits in graph.

```
$ git log --graph --pretty=oneline
*   d063b06bca5380c88a98e8f3690b7a9412b8eca9 (HEAD -> master, negative) Merge from
negative
|\
| * 2c2bfd4306d1972258370ef9affb8e6bdd392aca (positive) A tumor detected.
* | 4b0bfee6021c4140a141e9c852ac24236bd21495 Patient is normal
|/
* 9cc4a6f23d3cee06a74bb76df7b4365ef0bda04a (origin/master) Initial commit
```

2.4 Deleting a branch

Now we can delete the unwanted `positive` and `negative` branches.

```
$ git branch -d positive
Deleted branch positive (was 2c2bfd4).

$ git branch -d negative
Deleted branch negative (was d063b06).
```

If a branch has not been merged to another one, use `-D` to force delete it: `$ git branch -D positive`.

3. Managing remote repo

3.1 Adding and deleting remote source

```
$ git remote -v
origin  git@github.com:nbwuzhe/gitdemo (fetch)
origin  git@github.com:nbwuzhe/gitdemo (push)

$ git remote add origin2 git@github.com:nbwuzhe/gitdemo2

$ git remote -v
origin  git@github.com:nbwuzhe/gitdemo (fetch)
origin  git@github.com:nbwuzhe/gitdemo (push)
origin2 git@github.com:nbwuzhe/gitdemo2 (fetch)
origin2 git@github.com:nbwuzhe/gitdemo2 (push)

$ git remote remove origin2

$ git remote -v
origin  git@github.com:nbwuzhe/gitdemo (fetch)
origin  git@github.com:nbwuzhe/gitdemo (push)
```

3.2 Pull request on Github

Let's create a branch `finaldiag` from `master`, then add a line to the file `tfdemo.py`.

```
tf.print("This is the final diagnosis.")
```

Then add and commit the modification. Here we don't merge this branch to the `master`.

We may directly push this new local branch, and a new branch with the same name will be created automatically on the remote repo.

On GitHub, create a "pull request" under the branch `finaldiag`. Github will merge this branch into `master` for us and indicate if there is any conflicts. This provides possibility of code reviewing for the collaboration projects.

4. Miscellaneous

4.1 .gitignore

Ignore the file that you don't want to include into the local and remote repo, such as middle files of compilation: `.class` file of Java projects and `.obj` files of C++ projects. You may find a complete list of `.gitignore` file for different programming languages here.

4.2 Alias

As demonstrated briefly, aliasing helps you to abbreviate the most frequently used command or the long customized commands.

```
$ git config --global alias.co checkout

$ git config --global alias.ci commit

$ git config --global alias.br branch

$ git config --global alias.lg "log --color --graph --
pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)
<%an>%Creset' --abbrev-commit"
```

4.3 `git stash`

The command `git stash` is for the occation that we need to create some other works in another branch before we commit the modifications (i.e. modifications is still in working directory or stage).

Suppose we are finalizing the report on `master` branch. Add a line to `tfdemo.py` but don't add or commit:

```
tf.print("Generating report...")
```

We can see that this modification has not been staged (cached).

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   tfdemo.py

no changes added to commit (use "git add" and/or "git commit -a")
```

At this time we are notified that we need to fix the report for another patient immediately, so we need to create another branch `hotfix` based on `master`. But before that, we need to save the unfinished work temporarily using `git stash`.

```
$ git stash
Saved working directory and index state WIP on master: 7fbb607 Merge from positive

$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

The work-in-progress has been hidden! So we continue creating a new branch and finish hot fix:

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
```

Add a new file in `patient2.py`, then add and commit this modification.

```
import tensorflow as tf

tf.print("Report for Patient 2 has been fixed.")
```

Now we can switch back to `master` and merge this `hotfix` branch, then delete `hotfix`.

```
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)

$ git merge --no-ff -m "merged hot fix for patient 2" hotfix
Merge made by the 'recursive' strategy.
 patient2.py | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 patient2.py

 $ git branch -d hotfix
Deleted branch hotfix (was 0526d5c).
```

Now we can see that Git stored our work-in-progress somewhere.

```
$ git stash list
stash@{0}: WIP on master: 7fbb607 Merge from positive
```

Now we can restore our saved work-in-progress and finalize it by adding, commiting and pushing it.

```
$ git stash pop
On branch master
Your branch is ahead of 'origin/master' by 5 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   tfdemo.py

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (de5cd7f7fca822cd5b5b15090ea39da3913ed05d)
```