Exercises

Creating Snapshots

1- Initialize a new repository. Add two files in your working directory.

2- View the status of the working directory and the staging area.

3- Stage both files.

4- View the changes in the staging area.

5- Create a commit.

6- View the list of commits.

7- View the content of the last commit.

8- Update one of the files. View the changes in the working directory.

9- Stage the changes.

10- Unstage the file.

Solutions

1- Initialize a new repository. Add two text files in your working directory.

```
git init
echo hello > file1.txt
echo hello > file2.txt
```

2- View the status of the working directory and the staging area.

```
git status
git status -s
```

3- Stage both files.

```
git add .
```

4- View the changes in the staging area.

```
git diff --staged
```

5- Create a commit.

```
git commit -m “Initial commit.”
```

6- View the list of commits.

```
git log
```

7- View the content of the last commit.

```
git show HEAD
```

8- Update one of the files. View the changes in the working directory.

```
echo world >> file1.txt
git diff
```

9- Stage the changes.

```
git add file1.txt
```

10- Unstage the file.

```
git restore --staged file1.txt
```

Exercises

Browsing History

1- Make another commit. Now you should have two commits in your repo.

2- Show the changes in the last 2 commits.

3- Show all commits made by yourself. Use the one-liner option.

4- Show all commits with GUI in their message.

5- Show all commits with changes to file1.txt. Include the number of lines added/removed.

6- Compare the last two commits.

7- Check out the commit before the last commit. Note the detached HEAD in the terminal. Check out the master branch.

8- Show the author of every line in file1.txt.

9- Create a tag (v1.0) for the last commit. Show the history using the one-liner option and note the tag you just created.

10- Delete the tag.

Solutions

1- Make another commit. Now you should have two commits in your repo.

```
git add .
git commit -m "Update file1"
```

2- Show the changes in the last 2 commits.

```
git log --patch -2
```

3- Show all commits made by yourself. Use the one-liner option.

```
git log --author="Your name" --oneline
```

4- Show all commits with GUI in their message.

```
git log --grep="GUI"
```

5- Show all commits with changes to file1.txt. Include the number of lines added/removed.

```
git log --stat file1.txt
```

6- Compare the last two commits.

```
git diff HEAD~1 HEAD
```

7- Check out the commit before the last commit. Note the detached HEAD in the terminal. Check out the master branch.

```
git checkout HEAD~1
git checkout master
```

8- Show the author of every line in file1.txt.

```
git blame file1.txt
```

9- Create a tag (v1.0) for the last commit. Show the history using the one-liner option and note the tag you just created.

```
git tag v1.0
git log --oneline
```

10- Delete the tag.

```
git tag -d v1.0
```

Exercises

Branching & Merging

1- Create a new branch called feature/login. Switch to the new branch.

2- Show all the branches.

3- Update file1.txt in the current branch (feature/login) and make a new commit.

4- Show the commits across all branches.

5- Switch back to the master branch. Show the commits in the feature branch that don’t exist on master.

6- View the differences between master and feature/login.

7- Merge feature/login into master.

8- View the merged and unmerged branches.

9- Delete the feature branch.

10- Create a new branch called feature/logout. On this branch, write blue to file1.txt and make a commit.

Switch back to master, write green to file1.txt and make another commit.

Try to merge your feature branch into master. You’ll see a conflict. Resolve the conflict by accepting both changes. When you’re done merging, delete the new branch.

11- Create a new branch called bugfix/login. On this branch, write orange to file1.txt and make a commit.

Switch back to master, write green to file2.txt and make another commit.

View a graph of your branches. You'll see divergence.

Rebase the new branch on top of master.

View the graph of branches again. Note that the divergence is gone.

Do a fast-forward merge to bring the changes in the bugfix branch into master.

Solutions

1- Create a new branch called feature/login. Switch to the new branch.

```
git switch -C feature/login
```

2- Show all the branches.

```
git branch
```

3- Update file1.txt in the current branch (feature/login) and make a new commit.

```
echo sky >> file1.txt
git add .
git commit -m “Write sky to file1”
```

4- Show the commits across all branches.

```
git log --oneline --all
```

5- Switch back to the master branch. Show the commits in the feature branch that don’t exist on master.

```
git switch master
git log master..feature/login
```

6- View the differences between master and feature/login.

```
git diff master..feature/login
```

7- Merge feature/login into master.

```
git merge feature/login
```

8- View the merged and unmerged branches.

```
git branch --merged
git branch --no-merged
```

9- Delete the feature branch.

```
git branch -d feature/login
```

10- Create a new branch called feature/logout. On this branch, write blue to file1.txt and make a commit.

Switch back to master, write green to file1.txt and make another commit.

Try to merge your feature branch into master. You'll see a conflict. Resolve the conflict by accepting both changes. When you're done merging, delete the new branch.

```
git switch -C feature/logout
echo blue >> file1.txt
git commit -am “Write blue to file1”

git switch master
echo green >> file1.txt
git commit -am “Write green to file1”

git merge feature/logout
git mergetool
git add file1.txt
git commit

git branch -d feature/logout
```

11- Create a new branch called bugfix/login. On this branch, write orange to file1.txt and make a commit.

Switch back to master, write green to file2.txt and make another commit.

View a graph of your branches. You'll see divergence.

Rebase the new branch on top of master.

View the graph of branches again. Note that the divergence is gone.

Do a fast-forward merge to bring the changes in the bugfix branch into master.

```
git switch -C bugfix/login
echo orange >> file1.txt
git commit -am "Write orange to file1"

git switch master
echo green >> file2.txt
git commit -am "Write green to file2"

git log --oneline --all --graph

git switch bugfix/login
git rebase master

git log --oneline --all —graph

git switch master
git merge bugfix/login
```

Exercises

Collaboration

1- Create a GitHub repository called Tokyo.

2- Clone this repository in two different folders: John and Amy. We're trying to simulate two users collaborating on this repository.

3- Go to John's folder and make a commit. To save yourself from the hassle of re-entering your credentials every time you do a push, store your credentials in memory. Now, do a push.

4- Go to Amy's folder, and fetch the new commit. View the history and see how the remote master branch has moved forward. Merge origin/master into master.

5- From Amy's folder, create a new branch called feature/login. Make a commit on this branch. Once again, you need to configure Git to store your credentials in memory because you're in a different repository. The configuration we made earlier only applies to John's repository. Now, do a push.

6- View the local and remote branches.

7- Go to John's folder and do a fetch. View the history. Note the new branch. Create a local branch and map it to origin/feature/login. View the local and remote branches again to make sure your branches are set up properly.

8- Go back to Amy's folder, make another commit on the feature branch and do a push.

9- Back to John's folder, pull Amy's changes into the feature branch. View the history. The feature branch should be two commits ahead of master.

10- Merge the feature branch into master. View the history. Note that master is ahead of origin/master by two commits. So now you need to do a push to bring origin/master forward. View the history again to make sure master and origin/master are at the same place.

11- You're done with the feature branch. So it's time to remove it. Now, view the local and remote branches. The remote tracking branch is gone, but the local branch is still there and should be removed.

12- Go to Amy's folder. Do a fetch. View the history. Note that origin/master has moved forward and is ahead of master by two commits. Switch to master and merge origin/master into it. View the history to ensure master and origin/master are at the same location.

13- It's time to remove the local and remote-tracking branches.

Well done! You rocked! The stuff you've done requires a deep understanding of Git. Now, go grab a coffee or your favourite drink. :)

Solutions

1- Create a GitHub repository called Tokyo.

2- Clone this repository in two different folders: John and Amy. We're trying to simulate two users collaborating on this repository.

```
git clone <url> John
git clone <url> Amy
```

3- Go to John's folder and make a commit. To save yourself from the hassle of re-entering your credentials every time you do a push, store your credentials in memory. Now, do a push.

```
cd John
echo hello > file1.txt
git add .
git commit -m “Add file1”
git config credential.helper cache
git push
```

4- Go to Amy's folder, and fetch the new commit. View the history and see how the remote master branch has moved forward. Merge origin/master into master.

```
cd ..
cd Amy
git fetch
git log --oneline --all --graph
git merge origin/master
```

5- From Amy's folder, create a new branch called feature/login. Make a commit on this branch. Once again, you need to configure Git to store your credentials in memory because you're in a different repository. The configuration we made earlier only applies to John's repository. Now, do a push.

```
git switch -C feature/login
echo hello > file2.txt
git add .
git commit -m "Add file2"
git config credential.helper cache
git push -u origin feature/login
```

6- View the local and remote branches.

```
git branch -vv
git branch -r
```

7- Go to John's folder and do a fetch. View the history. Note the new branch. Create a local branch and map it to origin/feature/login. View the local and remote branches again to make sure your branches are set up properly.

```
cd ..
cd John
git fetch
git log --oneline --all --graph
git switch -C feature/login origin/feature/login
git branch -vv
```

8- Go back to Amy's folder, make another commit on the feature branch and do a push.

```
cd ..
cd Amy
echo world >> file2.txt
git commit -am "Update file2"
git push
```

9- Back to John’s folder, pull Amy’s changes into the feature branch. View the history. The feature branch should be two commits ahead of master.

```
cd ..
cd John
git switch feature/login
git pull
git log --oneline --all --graph
```

10- Merge the feature branch into master. View the history. Note that master is ahead of origin/master by two commits. So now you need to do a push to bring origin/master forward. View the history again to make sure master and origin/master are at the same place.

```
git switch master
git merge feature/login
git log --oneline --all --graph
git push
git log --oneline --all --graph
```

11- You’re done with the feature branch. So it’s time to remove it. Now, view the local and remote branches. The remote tracking branch is gone, but the local branch is still there and should be removed.

```
git push -d origin feature/login
git branch -vv
git branch -d feature/login
```

12- Go to Amy's folder. Do a fetch. View the history. Note that origin/master has moved forward and is ahead of master by two commits. Switch to master and merge origin/master into it. View the history to ensure master and origin/master are at the same location.

```
cd ..
cd Amy
git fetch
git log --oneline --all --graph
git switch master
git merge origin/master
git log --oneline --all —graph
```

13- It's time to remove the local and remote-tracking branches.

```
git remote prune origin
git branch -d feature/login
```

Well done! You rocked! The stuff you've done requires a deep understanding of Git. Now, go grab a coffee or your favourite drink. :)