# Git Review – $GitRich

Let's form a currency exchange app startup called GitRich and practice some git!

Learning Outcomes:

- Review common Git commands
- Practice Branching and Merging
- Identify and solve merge conflicts

Part I – Git Overview

Open bash.

CD into whichever directory you want to create a practice directory.

Type: **mkdir GitRich**

CD into GitRich

Initialize GitRich as a local git repo. Type: **git init**

Following company standards, rename your branch. Type: **git branch -m main**

Let's make a file to track the currencies our app will be able to exchange.

Create a new text file inside GitRich by either:

a. Using file explorer, navigate to your GitRich directory. Inside the directory, right click, select, new > text document. Rename it CurrencyList.txt
b. If you're using Bash, you can do it from the command line. In the GitRich directory type: **touch CurrencyList.txt**

In file explorer, if needed navigate to CurrencyList.txt and open it in a text editor.

Add several lines of currencies of your choice, e.g.:

Russian Ruble
Brazilian Real
Guatemalan Quetzal

Save and close.

Git status is a very useful command to see which files are not yet tracked, which are staged and ready to be committed, and which branch you are on.

Type: **git status**

Notice CurrencyList.txt is red. This indicates it, and any changes are not being tracked.

We're happy with it! Let's stage it. Type: **git add CurrencyList.txt**

Type: **git status**

CurrencyList.txt is now green, indicating there are changes ready to be committed. Also notice in the terminal git often provides relevant handy hints, like how to unstage the file in this example.

Next, let's commit our changes with a meaningful message.

Type: **git commit -m "First commit of CurrencyList.txt"**

Review the terminal output.

To review your commit history type: **git log**

Review the terminal output. It provides the commit id, the author, the date stamp, and our message.

Ruble trades are not very lucrative. Open CurrencyList.txt in a text editor and delete your worst performing currency and replace it with something else. Save and close.

**Do not stage your changes yet!**

Type: **git status**

You can see that there are now untracked changes, indicated by the red text.

To see the difference between the two versions type: **git diff CurrencyList.txt**

Review the output. The differences between the version in your working directory and what's been committed to the repo are nicely displayed.

Stage the file. Type: **git add CurrencyList.txt**

Commit the file. Type: **git commit -m "Replaced underperforming currency"**

Now we have two commits. To see the differences type: **git log**

Git displays all of the commits, with their ids, time stamps, and messages, and the head pointer shows which is the last commit.

To see the differences between two commits use the first seven characters of the commit ids to reference them. Using your own ids type: **git diff 1bba605 00378dd**

1. Which git command allows you to see the state of the working directory, including which files are untracked and tracked, are ready to be committed and what branch you are on?
2. Why is it important to have descriptive commit messages?
3. Which git command is used to see the difference between files in your current working directory and those committed to the git repo?

Part II - Branching and Merging

As a front-end developer at GitRich you've been tasked with working on a style guide and choosing the company's colors for branding and logos.

Let's create a feature branch to work on our new assignment:

a. If you are using git version 2.23 or later you can use the keyword 'switch'. Type: **git switch -c style-guide**
b. If you're using an earlier version of git type: **git checkout -b style-guide**

To see all of the branches type: **git branch**

All the branches are listed, and style-guide has an asterisk and is green indicating it's the branch you're on.

Let's get to work! Type: **touch CompanyColors.txt**

Open the new file in a text editor and add several lines specifying the new company colors, e.g.:

ForestGreen
SlateGray
Coral

Save and close.

Stage your file. Type: **git add CompanyColors.txt**

View the status. Type: **git status** and review the output.

Commit the file. Type: **git commit -m "First commit of CompanyColors.txt"**

Switch back to the main branch. Type: **git switch main** or type: **git checkout main**

You are now on the main branch. Use file explorer to navigate to your GitRich directory. What do you see there? Or rather, what don't you see there?

View the differences. Type: **git diff main..style-guide**

Now merge the code. From the main branch type: **git merge style-guide** and review the output.

Type: **git diff main..style-guide**

Now what do you see?

1. What is the git command you use to create and go to a new branch?
2. Which git command allows you to see all the branches in a repo?
3. Which git command is used to see the difference between files on your branch and those on the main branch?

Part III – Merge Conflicts

Conflicts can happen when two people make changes to the same file. Git doesn't know which is the correct version to accept when merging.

You have a nemesis at GitRich who thinks her company color choices are far superior and will stop at nothing to have her way. She goes into the main branch to edit the code.

While on the main branch, open CompanyColors.txt in a text editor and make some sneaky changes to the colors. Save and close.

Stage the changes. Type: **git add CompanyColors.txt**

Commit the changes. Type: **git commit -m "changed the colors to the best colors"**

Meanwhile you are in a focus group, and decide nearly all the colors are pretty great, but one needs to be changed.

Go to your feature branch. Type: **git switch style-guide** or type: **git checkout style-guide**

Open CompanyColors.txt and edit the same color(s) your nemesis changed. Save and close.

Stage the changes. Type: **git add CompanyColors.txt**

Commit the changes. Type: **git commit -m "changed Coral to Crimson"**

Okay, you are happy with the changes and ready to merge them into the main branch.

Switch to main. Type: **git switch main** or type: **git checkout main**

On the main branch type: **git merge style-guide**

Uh oh! See the message:
Auto-merging CompanyColors.txt
CONFLICT (content): Merge conflict in CompanyColors.txt
Automatic merge failed; fix conflicts and then commit the result.

Type: **git status** and review the output

Type: **git diff main..style-guide** and review the output

While on the main branch, open CompanyColors.txt in a text editor. Notice git has added markers to highlight the differences between the versions.

```
1 ForestGreen
2 SlateGray
3 <<<<<<< HEAD
4 Puce
5 =======
6 Crimson
7 >>>>>>> styleGuide
8
```

Fix the version as desired. Remove your nemesis' undesired code and the markers - like the screenshot below. Save and close.

```
1  ForestGreen
2  SlateGray
3  Crimson
4
```

Stage the changes. Type: **git add CompanyColors.txt**

Commit the changes: **git commit -m "removed conflicting color"**

If you fixed the merge conflict correctly, the merge should finish at this point.

Type: **git merge style-guide**

You should see a message letting you know the branch is already up to date.

If you still have merge conflicts, try again. Repeat the steps above and review the terminal output carefully until the conflict is resolved.

1. What are some ways you can identify the differences between the files that are causing a merge conflict?

Part IV – Additional Resources

If you have more time a good place to get some more practice is at GitHub Learning Lab (approximately 30 mins):
https://lab.github.com/githubtraining/managing-merge-conflicts

GitHub Learning Lab in general has many other related free courses:
https://lab.github.com/

The Tech U Introduction to Git and GitHub:
https://rockfin.sharepoint.com/:b:/s/QLTechUTeam/ETn6iYvwmcFLvuXptnusEM0ByWIczyCuT-x-BPz3iJ6vgA?e=QymbCf