Mastering git

Lesson 5

Irina Gulina

Tomas Tomecek

Red Hat

Course schedule

September						
S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

October						
S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	✖	26	27	28
29	30	31				

November						
S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Class N HW	2	3	4	5	6	Bonus Task
Deadline	11.10	18.10	25.10	1.11	8.11	13.11

Course grade: November 20

Class 2 HW feedback

- I expected you'd fork into your own account namespace
  - Why creating new namespaces?
  - GitLab doesn't have a way to connect a user with a fork in a namespace.

Class 3 HW feedback

- Visualization (`git log --graph` is not great)
- Different levels of git experience in the class
- `git pull` vs. `git pull --rebase`

- Class 6 agenda
  - ~~content of `.git/` directory~~, read Git Internals: https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain
  - how does rebase work internally
  - ~~difference in merge strategies~~, please read git-merge: https://git-scm.com/docs/git-merge#_merge_strategies
  - `git submodule`
  - tig - terminal user interface for git
  - git aliases, shell aliases and shortcuts

Questions

- How to show local commits which are not on origin remote?
- What is the difference between rebase and merge?
- What is the golden rule of rebase?
- What changes to commits can be done during an interactive rebase?
- What is a merge conflict? When/how does it happen?
- `git switch main`
  `git pull upstream main`
  `git pull –rebase upstream main`
  What's the difference between the two pull commands?

Any questions or suggestions?

Today’s class

- Change working tree and HEAD (overflow from last class)
- git-reset
- Labs:
  - reset, restore, show
  - it’s never a bad idea to resolve some merge conflicts & rebase

Local troubles

You have a great freedom

to rewrite your history ***locally***

Change working tree and HEAD

⚠️⚡DON'T LOSE YOUR WORK⚡⚠️

✍ What exactly I changed? Or staged?

Steps to reproduce:

Do changes in working tree. “What did I do exactly? I don’t remember and want to know.”

Solution:

```
$ git diff [<commit>] [--] [<path>...]

$ git diff --cached [<commit>] [--] [<path>...]
```

Typically you would want comparison with the latest commit, so if you do not give <commit>, it defaults to HEAD.

$ man git-diff

😸 Undoing local changes, not committed

Steps to reproduce:

The cat walked across your keyboard, while you were making coffee. You have not noticed and saved the changes in your editor, then saw them with `git diff`.

Solution:

```
$ git restore -- <pathspec>
```

```
$ git checkout -- <pathspec>
```

git-restore - Restore working tree files

pathspec - Pattern used to limit paths in Git commands.

$ man git-restore man git-glossary

Changing the last local commit

1. How to modify the last commit message or add more stuff to the last commit

Solution:

```
$ git commit --amend
```

Don’t amend your last commit if you have already pushed it upstream!

Undo the last local commit(s)

Solution:

```
$ git reset <last good commit>
```

```
$ man git-reset
    git - reset - Reset current HEAD to the specified state

--mixed         Resets the index but not the working tree (i.e., the changed
files are preserved but not marked for commit) and reports what has not been
updated. This is the default action.
```

🤯 Change HEAD into a known state

Solution:

```
$ git reset --hard <last good commit>
```

DISCARDS UNSTAGED CHANGES

```
$ man git-reset

(--hard) set the current branch head (HEAD) to <commit>,

optionally modifying index and working tree to match.
```

👷 How do I set my local main to upstream main?

Solution:

```
$ git reset --hard upstream/main
```

👷 Fix an earlier local commit

Scenario:

A file was not included in an earlier commit.

Solution:

```
$ git add <file>
$ git commit --fixup <earlier-commit>
$ git rebase -i --autosquash <even-more-earlier-commit>
```

Moving local commits between branches

Scenario:

Commits were made on a `main` branch, but they should be on another branch instead

Solution:

```
$ git branch feature
```

What is the difference with `git switch -c feature`?

```
$ git reset --hard origin/main
$ git switch feature
```

How to avoid it?

Public troubles

Undo a commit, pushed

Steps to reproduce:

```
$ touch file.txt
$ git add file.txt
$ git commit -m “Something terribly wrong”
$ git push upstream main
```

Undo a commit, pushed

Solution:

Find SHA hash of that commit.

```
$ git revert <commit>
$ git push
```

It’s the safest scenario, it doesn’t alter history!

Labs

Lab #1 Conflicts and Rebase.

- Pair with someone in a class to a group of two.
- Student #1 shares any git repo they have with a Student #2. It's ok to share a Mastering git fork.
- Student #2 needs to create a PR to the repo of Student #1. Change an existing line, e.g. adding their UCO to the beginning of any line in any file.
- After a PR is opened, Student #1 changes the same file and line too, e.g. by adding their UCO to the end of a line.
- Now Student #2 needs to fix their PR, so it can be approved and merged. After merge, both UCO should exist in that file.
- Mind meaningful commit messages, PR body and title, keep some short conversation in a PR.

Do the same changing the roles.

Class 5 homework

Class 5 homework

https://gitlab.com/redhat/research/mastering-git/-/blob/main/README.md#class-5-homework

- **Deadline: 8. November 2023, 23:59**
- Pair with any student from your class. If you did a similar lab in a class, please switch roles and be Student #1 or #2 accordingly.
- Student #1 shares any git repo (any “Example”, but not Mastering git) with Student #2 and Irina/Tomas.
- Student #2 makes a contribution to an existing file inside Student's #1 git repo by opening a MR.
- Student #1 will push a change to the repo ("Example") that will cause a merge conflict in Student's #2 MR.
- Student #2 resolves the merge conflict.
- After conflict is resolved, Student #1 asks to change something else in the MR. Based on a requested change, Student #2 needs to do a rebase again or add an additional commit.
- MR should be merged in the end and or rejected/closed if no positive value.
- Afterwards, Student #1 makes any contribution to their own (“Example”) repo via MR, and Student #2 needs to process it accordingly.
- Both students shall mind/practise/demonstrate best Git practices on commits and MR.
- All MRs during this HW should also contain a file (in a separate commit) with a shell history. In a case of rebase / new MR change, a new history file should be attached accordingly. So, in the end, each MR may contain several commits demonstrating git history.
- In a case of any troubles tag us on a MR to help, or just share what difficulties you had in MR conversations, so we can address the most common troubles on the last git class.

Source: https://octodex.github.com/

Questions?

THANK YOU!