Mastering git

Lesson 4 (fixing problems)

Irina Gulina

Tomas Tomecek

Class 2 homework feedback

- Some solutions were fairly similar
- `git push -u` is winning
- Loved the creativity of the content
- Why the `micro` editor?
-
- Slides are not meant to be the source of information, instead use:
  - `$ man git` – https://git-scm.com/docs/git
  - https://git-scm.com/book/en/v2 (a bit outdated)
  - https://www.atlassian.com/git/tutorials

Red Hat

Class 3 followups (push --set-upstream)

- `$ man git-push`

  ```
  When neither the command-line nor the configuration specify what to
  push, the default behavior is used, which corresponds to the simple
  value for push.default: the current branch is pushed to the
  corresponding upstream branch, but as a safety measure, the push is
  aborted if the upstream branch does not have the same name as the
  local one.
  ```

Class 3 followups (push.default #1)

- `$ man git-config`

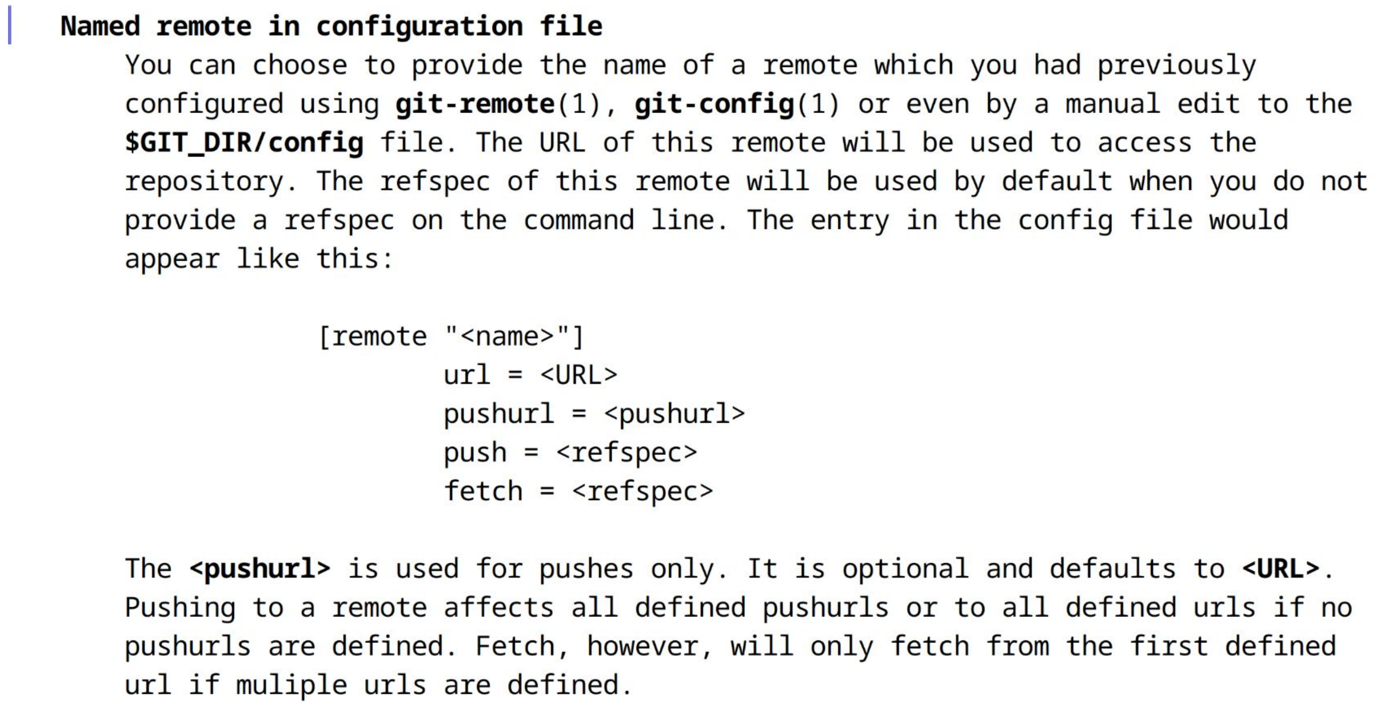
```
push.default
  simple - pushes the current branch with the same name on the
remote.

If you are working on a centralized workflow (pushing to the same
repository you pull from, which is typically origin), then you need
to configure an upstream branch with the same name.

This mode is the default since Git 2.0, and is the safest option
suited for beginners.
```

Class 3 followups (push.default #2)

- `$ man git-config`

```
push.default
  current - push the current branch to update a branch with the
same name on the receiving end. Works in both central and
non-central workflows.
  upstream - push the current branch back to the branch whose
changes are usually integrated into the current branch (which is
called @{upstream}). This mode only makes sense if you are pushing
to the same repository you would normally pull from (i.e. central
workflow).
```

Class 3 followups (fetch/push URLs)



Named remote in configuration file

You can choose to provide the name of a remote which you had previously configured using **git-remote**(1), **git-config**(1) or even by a manual edit to the **$GIT_DIR/config** file. The URL of this remote will be used to access the repository. The refspec of this remote will be used by default when you do not provide a refspec on the command line. The entry in the config file would appear like this:

```
        [remote "<name>"]
                url = <URL>
                pushurl = <pushurl>
                push = <refspec>
                fetch = <refspec>
```

The **<pushurl>** is used for pushes only. It is optional and defaults to **<URL>**. Pushing to a remote affects all defined pushurls or to all defined urls if no pushurls are defined. Fetch, however, will only fetch from the first defined url if muliple urls are defined.

Class 3 HW feedback next time

- notebook updating
- some of your feedback applied already in this class

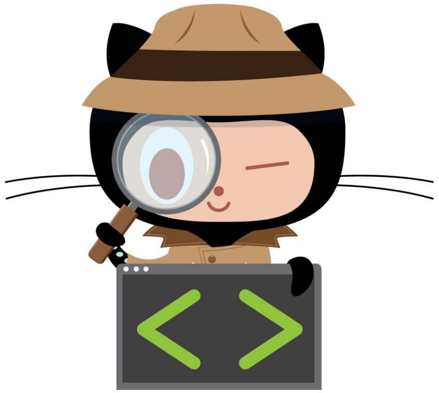
Any questions or suggestions?

We do have questions!

- What’s the difference between origin and upstream remotes?
- How do you get the latest content from upstream
  - To your local clone
  - To your fork
- What constitutes a bad pull request in terms of best practices we discussed last week?
- Did anyone already use some knowledge from this class in their projects?

Today’s class

- **Fixing Mistakes**
- Handy git tools and commands
- Rebase and Merge Conflicts
- Issues of various complexity
  - Local troubles
  - Public troubles
- Labs:
  - Solving merge conflicts
  - Changing history



How to find things?

Revision selectors

1. `1c002dd4b536e7479fe34593e72e6c6c1819e53b`

2. `$ git log --oneline`

```
1c002dd changed the version number
085bb3b removed unnecessary test code
a11bef0 init commit
```

3. `$ git reflog`

```
734713b HEAD@{0}: commit: fix refs handling, added gc
d921970 HEAD@{1}: merge phedders/rdocs: Merge made by the
'recursive' strategy.
1c002dd HEAD@{2}: commit: add some blame stuff
1c36188 HEAD@{3}: rebase -i (squash): updating HEAD
95df984 HEAD@{4}: commit: # This is a combination of two
```

4.
```
$ git show main@{yesterday}
$ git show main@{2.months.ago}
$ man gitrevisions
```

5. Ancestry references (`^` `~`) : git show + <reference/pointer>
   - `ca82a6d^`
   - `ca82a6d^^`
   - `HEAD`
   - `HEAD^`
   - `HEAD^2` (is it the same as `ca82a6d^^` ?)
   - `HEAD~` (is it the same as HEAD^ ?)
   - `HEAD~2` (is it the same as HEAD^2 ?)
   - `HEAD~3^2` (is it valid?)

git log

Navigating in history of commits, formatting + filtering

Examples:

git log --oneline

```
0e25143 Merge branch 'feature'
ad8621a Fix a bug in the feature
16b36c6 Add a new feature
23ad9ad Add the initial code base
```

git log --oneline --decorating

```
0e25143 (HEAD, main) Merge branch 'feature'
ad8621a (feature) Fix a bug in the feature
16b36c6 Add a new feature
23ad9ad (tag: v0.2) Add the initial code base
```

git log

Examples:

git log --stat

```
commit f7dfcf6b91c015f01e449dcfa77c3072c3d71cca
Author: Janine Machs <jmachs@example.com>
Date:   Thu Jun 30 06:42:02 2022 +0000

    https://github.com/myProject/projectname/pull/161#pullrequestreview-1023271581
    + renamed sample to display purpose

 ...cluster.yml => cluster.yml}                   |  2 +-
 roles/configure_hsr.yml                          | 13 ++-----------
 roles/main.yml                                   | 27 ++++++++++++++-------------
 3 files changed, 17 insertions(+), 25 deletions(-)
```

git log

Examples:

git log -p

```
commit 16b36c697eb2d24302f89aa22d9170dfe609855b
Author: Anna <anna@example.com>
Date:   Fri Sept 25 17:31:57 2022 -0500
    Fix a bug in the feature
diff --git a/hello.py b/hello.py
index 18ca709..c673b40 100644
--- a/hello.py
+++ b/hello.py
@@ -13,14 +13,14 @@ B
-print("Hello, World!")
+print("Hello, Git!")
```

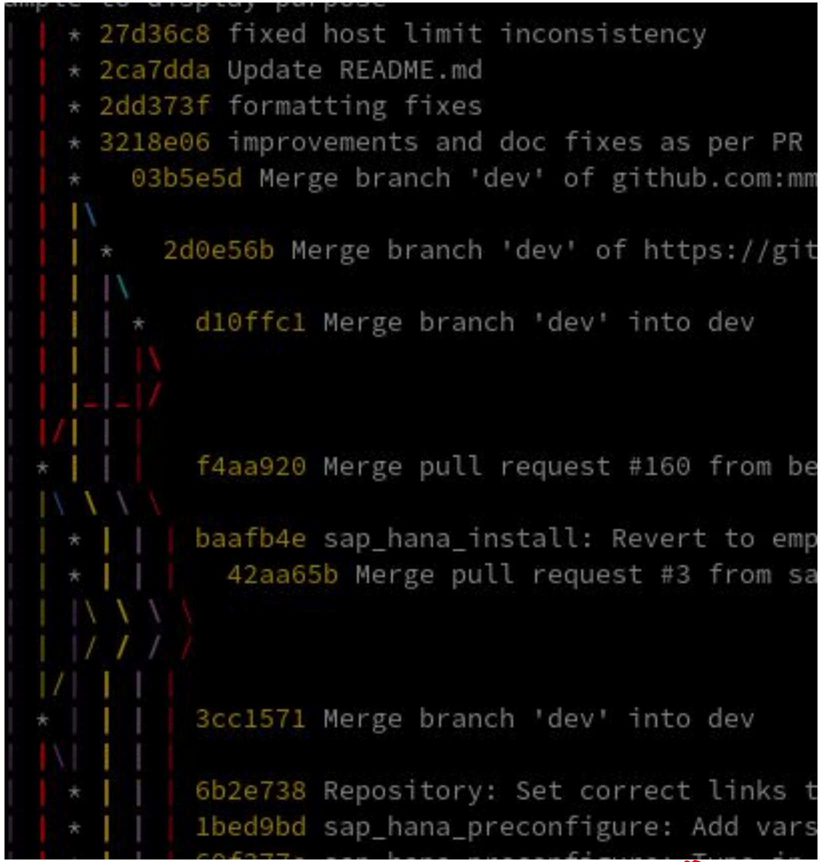
git shortlog

```
Release announcement

Barbora (2):
      Fix a bug in the feature
      Fix a serious security hole in our framework

Jirka (3):
      Add the initial code base
      Add a new feature
      Merge branch 'feature'
```

graph

Display branch structure of commit history

git log –graph –oneline



```
ample to display purpose
| | * 27d36c8 fixed host limit inconsistency
| | * 2ca7dda Update README.md
| | * 2dd373f formatting fixes
| | * 3218e06 improvements and doc fixes as per PR
| | *   03b5e5d Merge branch 'dev' of github.com:mm
| | |\
| | | *   2d0e56b Merge branch 'dev' of https://git
| | | |\
| | | | *   d10ffc1 Merge branch 'dev' into dev
| | | | |\
| | |_|_|/
| |/| | |
| * | | |   f4aa920 Merge pull request #160 from be
| |\ \ \ \
| | * | | | baafb4e sap_hana_install: Revert to emp
| | *   | | | 42aa65b Merge pull request #3 from sa
| | |\ \ \ \
| | |/ / / /
| |/| | | |
| * | | | |   3cc1571 Merge branch 'dev' into dev
| |\| | | |
| | * | | | 6b2e738 Repository: Set correct links t
| | * | | | 1bed9bd sap_hana_preconfigure: Add vars
```

git log

git log -3 <- what does it do?

git log --after="2022-6-11"

git log --after='yesterday'

git log --after="2022-6-11" –before='2022-7-11'

git log --author='Mary'

git log --author='Mary\|Tomas'

git log --grep="BZ-123:"

git log --foo.py bar.py

giit log -S"Hello, World"

git log main..feature

Questions

How to show commits on the “experiment” branch, which are not on “main”?

The opposite?

How to show local commits which are not on origin remote?

Solution:

```
$ git log main..experiment
```

7. Multiple points

   How to see what commits are in any of several branches, that aren’t in the branch you’re currently on?

   How to see all commits on A and B, which are not on C?

   More than two references can be specified.

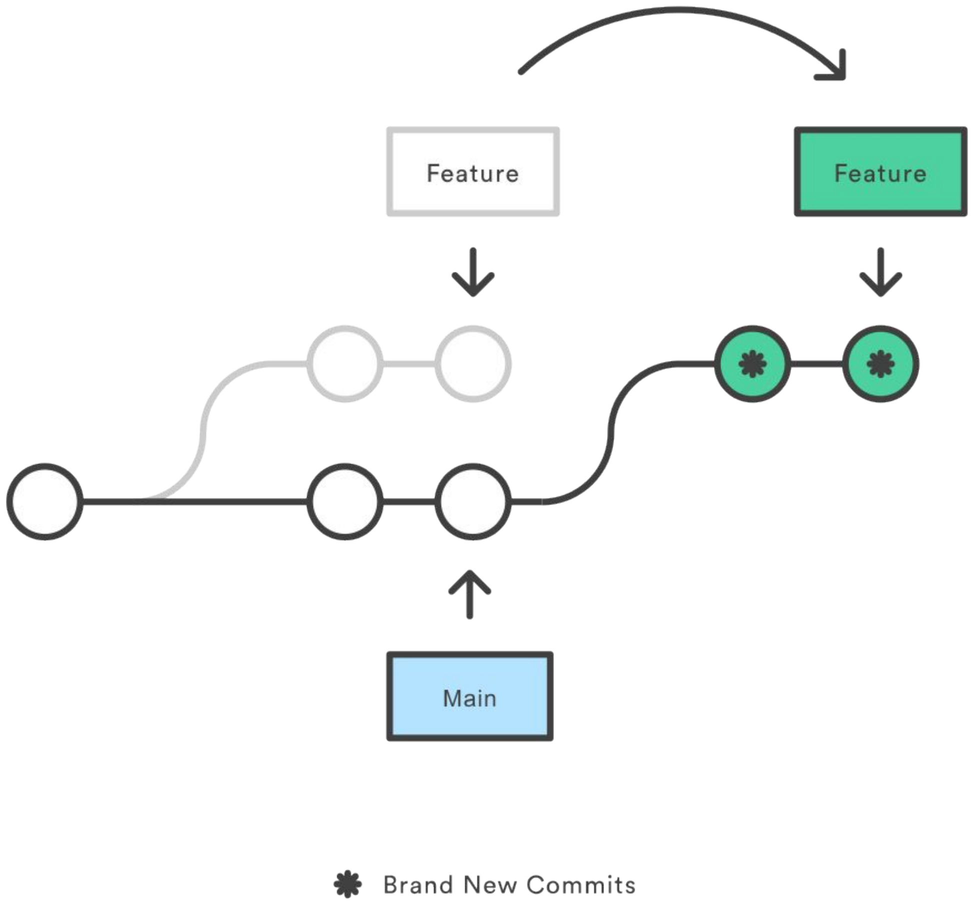
Solution:

```
$ git log A B ^C
$ git log A B --not C
```

Rebasing

More info: https://git-scm.com/book/en/v2/Git-Branching-Rebasing

Rebase



Source: https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase

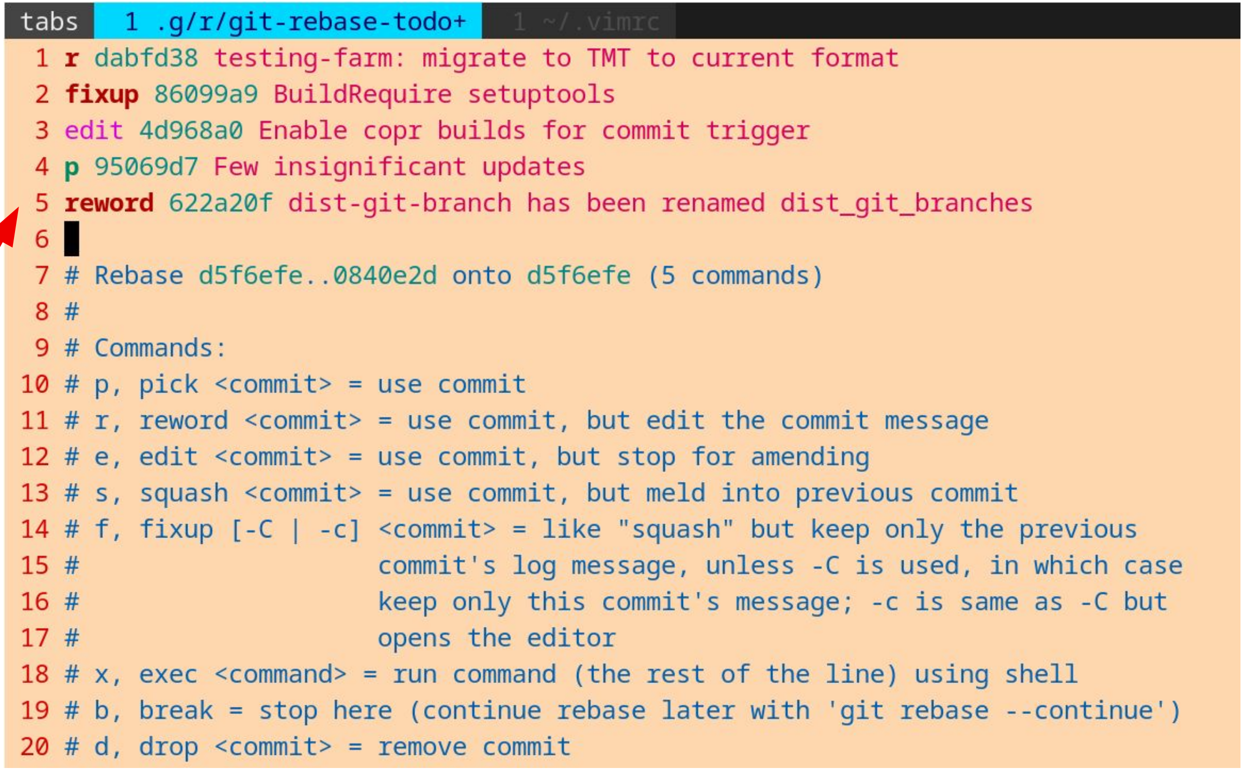
Interactive rebase

Changing of commits in many ways: editing, deleting, squashing, reordering.

```
git rebase -i <revision_selector>
git rebase -i HEAD~4
```

```
$ man git-rebase
    git-rebase - Reapply commits on top of another base tip
```

Interactive rebase



```
r dabfd38 testing-farm: migrate to TMT to current format
fixup 86099a9 BuildRequire setuptools
edit 4d968a0 Enable copr builds for commit trigger
p 95069d7 Few insignificant updates
reword 622a20f dist-git-branch has been renamed dist_git_branches

# Rebase d5f6efe..0840e2d onto d5f6efe (5 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
```

HEAD

Source: Tomas' laptop

Interactive rebase (lab)

- Let’s try this out
- https://gitlab.com/redhat/research/mastering-git/-/blob/main/labs/lab4.md?ref_type=heads
- Task 3

Interactive rebase

If one modifies a commit, that commit and all following commits will have new ….. ?

How will it affect local/not pushed changes?

How will it affect pushed changes?

Interactive rebase

If one modifies a commit, that commit and all following commits will have new ….. ?

How will it affect local/not pushed changes?

How will it affect pushed changes?

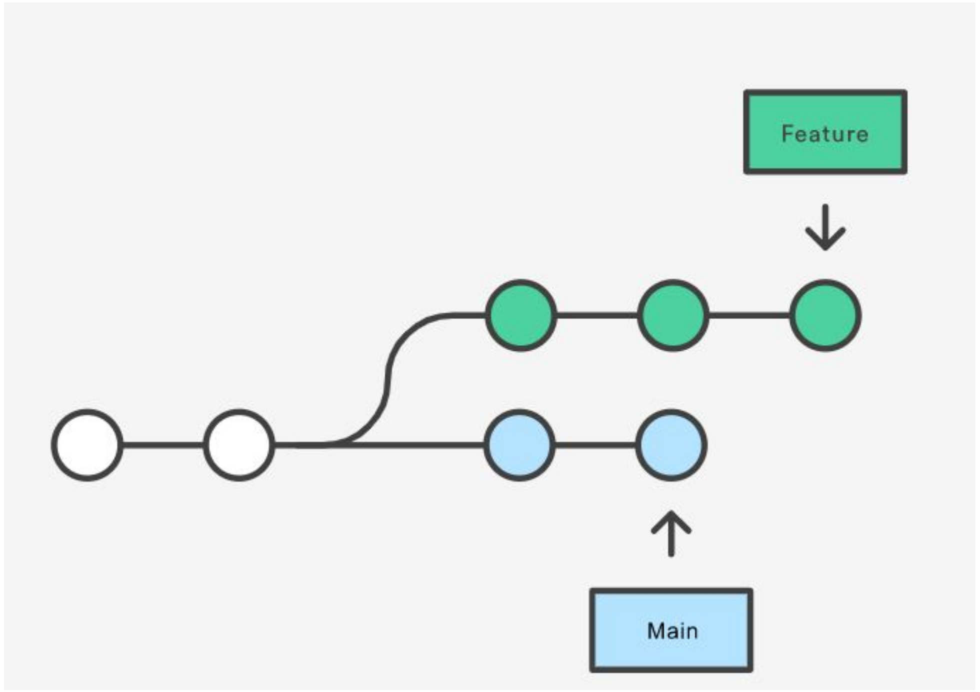
Golden Rule of Rebase – Never rebase on Public branches

Rebase vs Merge

Rebase and Merge solve the same problem – integrate changes between branches, but they do it in a different way.

https://www.atlassian.com/git/tutorials/merging-vs-rebasing

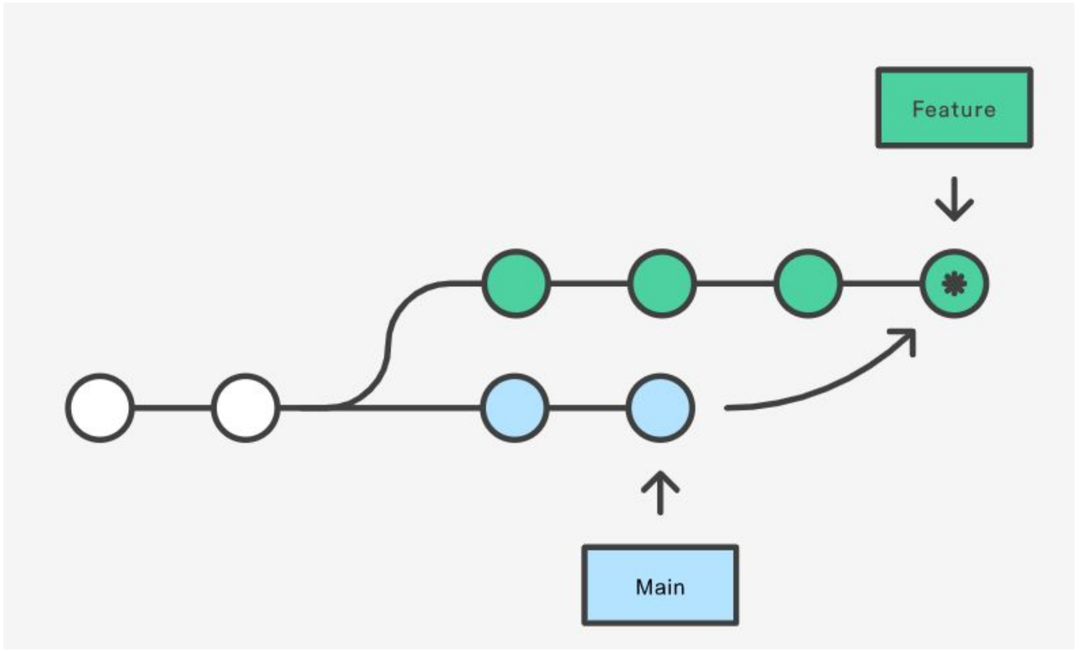
Rebase vs Merge



```
git checkout feature
git merge main
```

```
git merge feature main
```

Rebase vs Merge

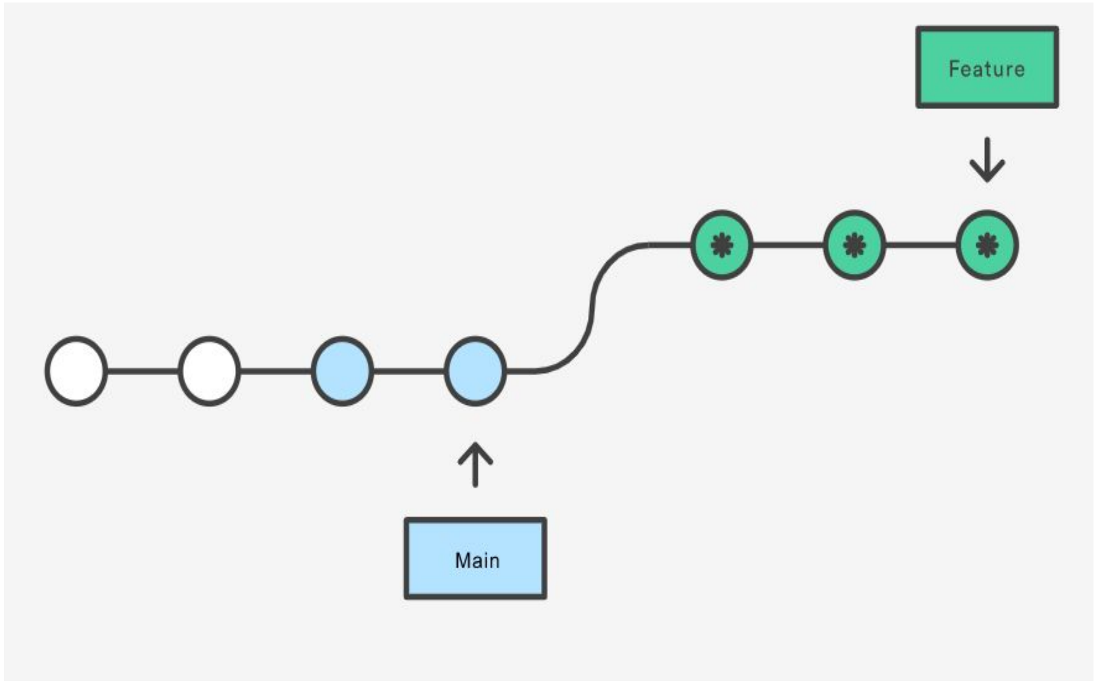


git checkout feature

git merge main

git merge feature main

Rebase vs Merge



```
git checkout feature
git rebase  main
```

Merge conflicts

Merge conflicts

An event when git can’t resolve code differences between commits.

- Git fails to start the merge

```
error: Entry '' not uptodate. Cannot merge. (Changes in working directory)
```

- Git fails during the merge

```
error: Entry '' would be overwritten by merge. Cannot merge. (Changes in
staging area)
```

Create a merge conflict

- Create a Git repo or take an existing one
- Add some text into a file
- Commit the change

- Create a new branch
- Overwrite text in that file and commit it

- Updata the same file again on the main branch, commi it

- Try to merge those two branches

Resolve a merge conflict

- Identify the conflict
- Inspect it
- Make changes
- Stage those changes

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-using-the-command-line

Lab Scenario: rebase & conflicts

- `git fetch upstream`
- `git switch -c lab3-solution-good upstream/lab3-solution-good`
- `git rebase upstream/lab3-solution-bad`

Class 4 lab

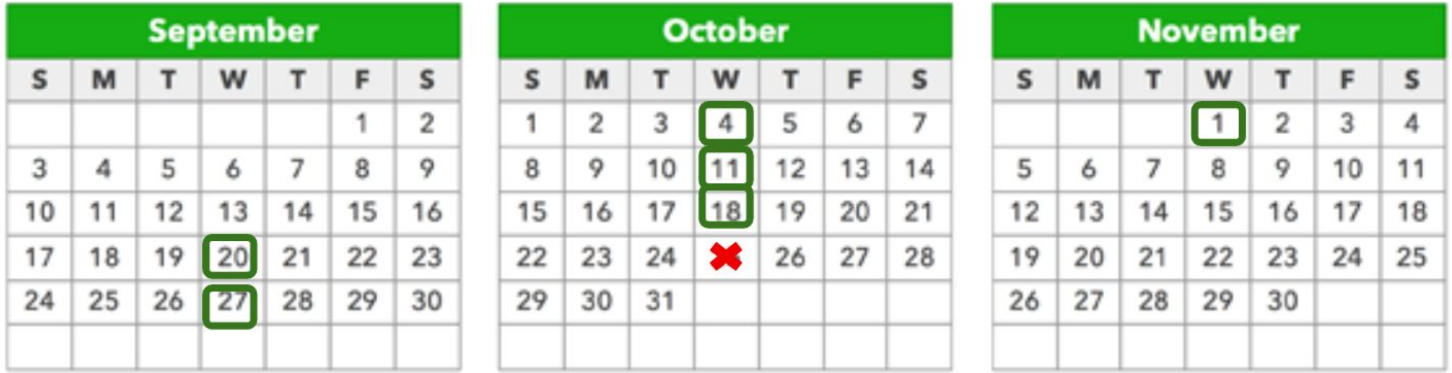
- https://gitlab.com/redhat/research/mastering-git/-/blob/main/labs/lab4.md?ref_type=heads

Class 4 homework

Deadline October 25 23:59

- https://gitlab.com/redhat/research/mastering-git#class-4-homework

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	$\boxed{20}$	21	22	23
24	25	26	$\boxed{27}$	28	29	30

October

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

November

S	M	T	W	T	F	S
			$\boxed{1}$	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		

HW#	1	2	3	4	5	Bonus Task
Deadline	11.10	18.10	25.10	1.11	8.11	13.11

Course grade: November 20

Questions?

THANK YOU!

Red Hat