Advanced Git

IVS demonstration exercise

Viktor Malík, Petr Stodůlka, Pavel Odvody

Red Hat

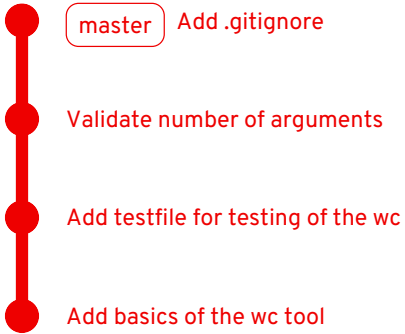
Prerequisites

- Basic knowledge of Git commands for:
  - creating commits (`git add`, `git commit`)
  - inspecting current state (`git status`, `git diff`)
  - inspecting history (`git log`, `git show`)
  - working with remotes (`git pull`, `git push`)
  - working with branches (`git checkout`, `git branch`)

Let's start

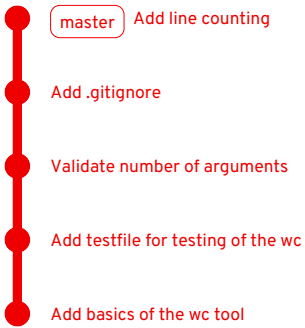
- We'll write a simple tool for counting characters, words, and lines in a file (similar to the `wc` utility)
- We start with a pre-initialized repo containing very basics of the tool: https://github.com/viktormalik/git-workshop
- The repo contains a source file `wc.c`, a testing file, and a `Makefile`
- We start by adding `.gitignore` and commiting it

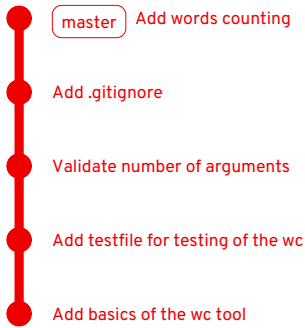
Current status of the repo



Basic team synchronisation

Every member implements a different feature in their master

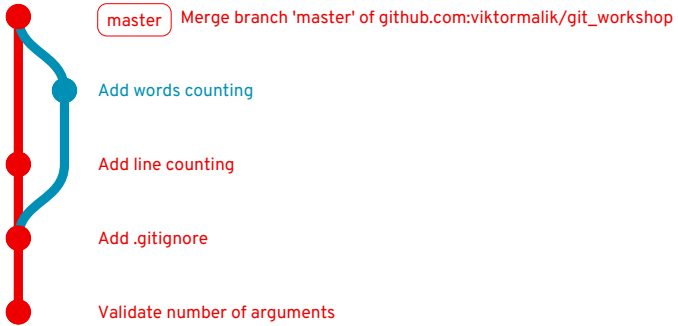




Red Hat

Basic team synchronisation

The second one to push must do a merge (and resolve a merge conflict)



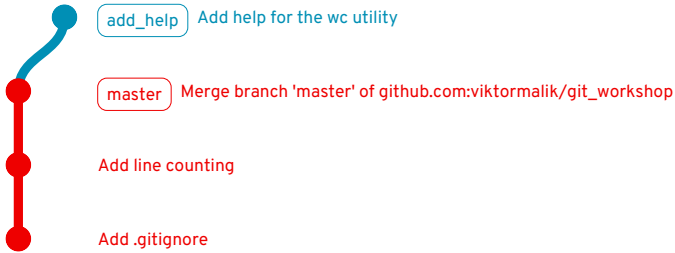
Red Hat

Better team synchronisation

- **This is not a good practice!**
- Always implement new features in **separate branches**.
- Potential merge conflicts should be resolved in the feature branch.
- Ideally, merging into master should be always done using **pull requests**
  - They allow other team members to comment on the changes
  - Changes can be **reviewed** before they get into master
  - Master always contains a working and approved version of the project

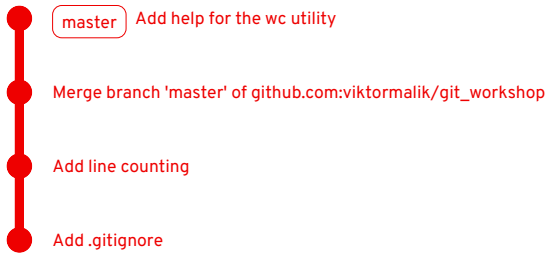
Using a feature branch

Let us add help into the tool using a separate branch add_help



Using a feature branch

The state of master after **rebase**:



Moving branches

We have 2 branches pointing to the same commit and we want to move one backwards.

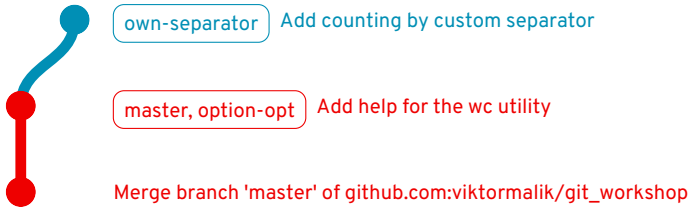
own-separator, option-opt Add counting by custom separator

master Add help for the wc utility

Merge branch 'master' of github.com:viktormalik/git_workshop

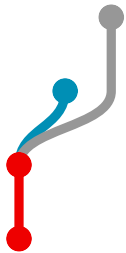
Moving branches

This can be done using `git reset HEAD^`.



Moving branches

After adding a new commit to *options-opt*:



option-opt Optimise getting CLI options

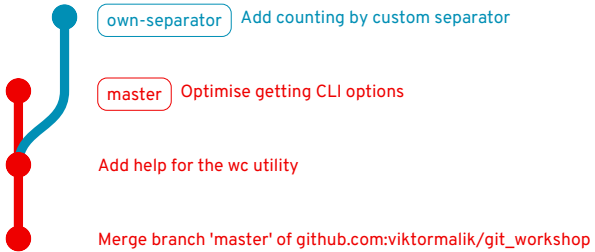
own-separator Add counting by custom separator

master Add help for the wc utility

Merge branch 'master' of github.com:viktormalik/git_workshop

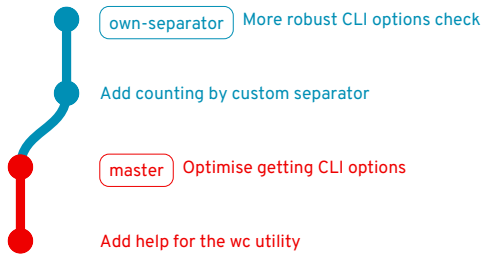
Moving branches

options-opt can be now merged into master while *own-separator* remains a feature branch in development.



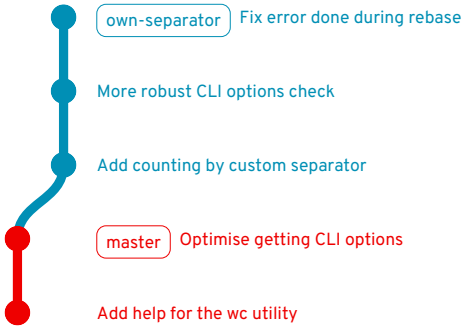
Rebasing feature branches

We add more commits to the feature branch and then **rebase** it onto master (to avoid creation of a merge commit).



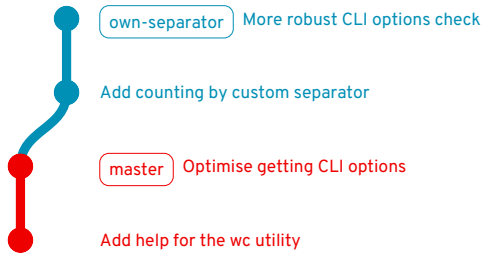
Rebasing feature branches

We made a mistake during rebase, which we had to fix with an additional commit.



Rebasing feature branches

It is possible to merge the “fix commit” into one of the previous commits using **interactive rebase** (`git rebase -i`).

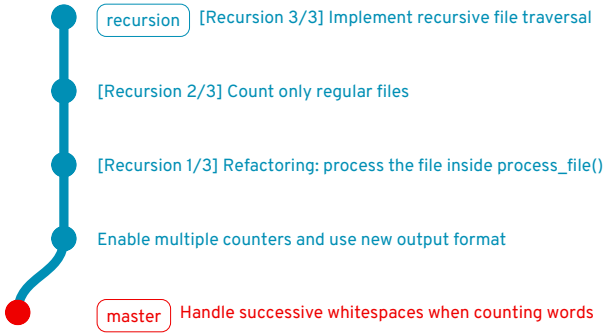


Interactive rebase

- One of the most important Git features in the modern pull request-based workflow.
- Allows to **edit**, **reorder**, **merge**, or **drop** commits.
- **Rewrites history** – should be only used on feature branches.
- **Never rewrite history of master!**
  - Other developers would not be able to do `git pull`.

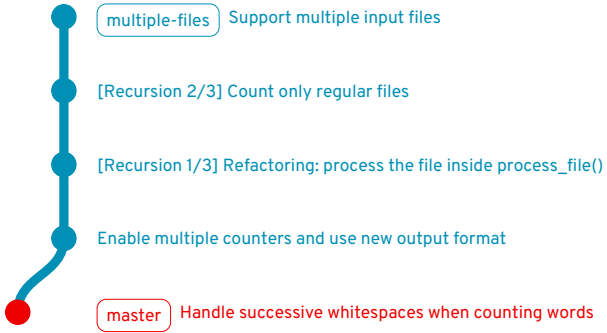
Copying commits from other branches

It is possible to copy commits from other branches (e.g. commits implementing useful features from co-workers feature branches) using `git cherry-pick`.



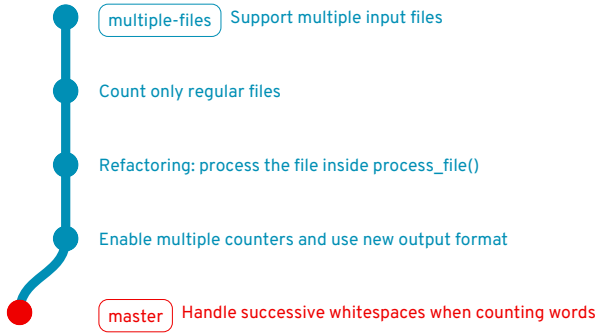
Copying commits from other branches

After moving 3 commits from *recursion multiple-files*:



Copying commits from other branches

If the commits are altered in *multiple-files*, it mey be needed to use `skip` when rebasing *recursion* onto *multiple-files*.



Hunting bugs in Git history

- We often discover a bug that was certainly introduced **somewhere in the Git history**.
  - There is a revision in the past where some test works correctly.
  - However, the test does not work now.
- Git offers `git bisect` that uses **binary search** to localise the commit that caused the bug.
  - `git bisect start` starts bisecting.
  - `git bisect good` marks a commit that does not contain the bug.
  - `git bisect bad` marks a commit contains the bug.
- The process can be **automated** using a script that returns 0 on success and a non-zero result on failure.