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`)
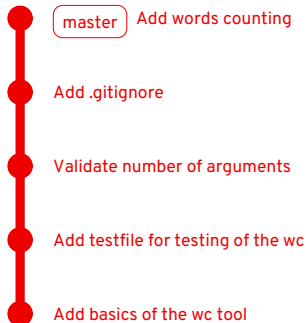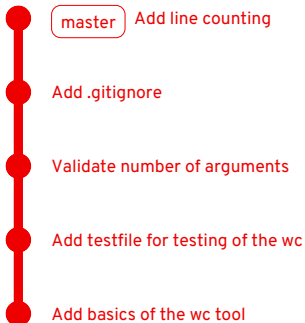
Red Hat

Advanced work with Git

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
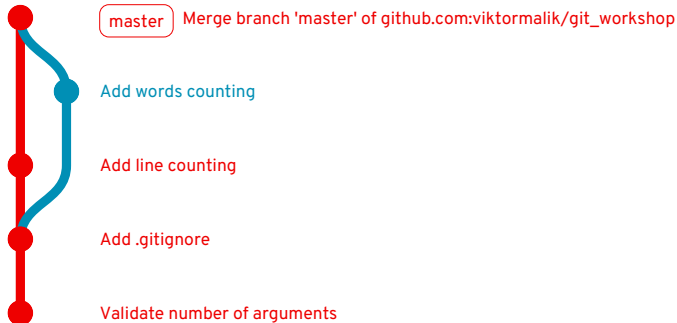
Red Hat

Current status of the repo

master Add .gitignore

Validate number of arguments

Add testfile for testing of the wc

Add basics of the wc tool

Red Hat

Basic team synchronisation

Every member implements a different feature in their *master*

master — Add line counting

Add .gitignore

Validate number of arguments

Add testfile for testing of the wc

Add basics of the wc tool

master — Add words counting

Add .gitignore

Validate number of arguments

Add testfile for testing of the wc

Add basics of the wc tool

Red Hat

Basic team synchronisation

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

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

Add words counting

Add line counting

Add .gitignore

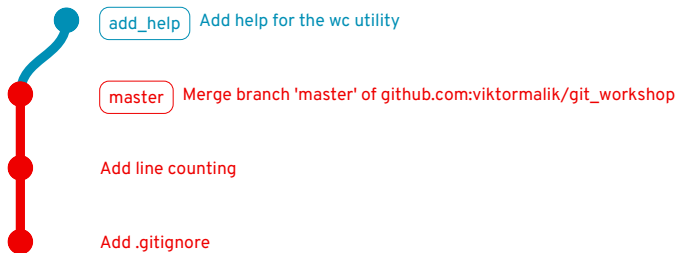Validate number of arguments

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
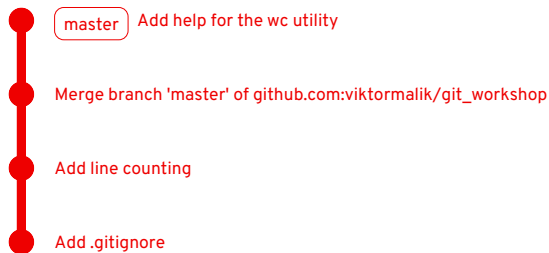 - Master always contains a working and approved version of the project

Red Hat

Using a feature branch

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

add_help — Add help for the wc utility

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

Add line counting

Add .gitignore

Red Hat

Using a feature branch

The state of *master* after **rebase**:

(master) Add help for the wc utility

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

Add line counting

Add .gitignore

Red Hat

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

Red Hat
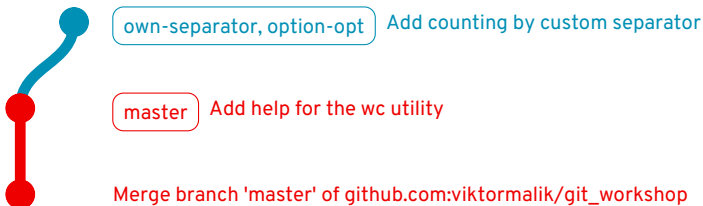
Moving branches

This can be done using `git reset HEAD^`

own-separator — Add counting by custom separator

master, option-opt — Add help for the wc utility

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

Red Hat
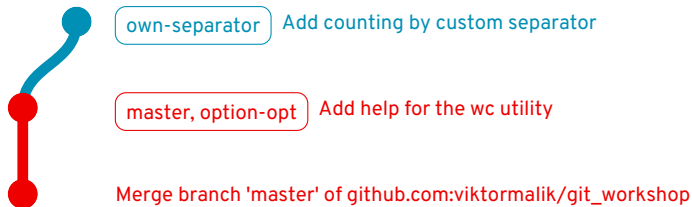
Moving branches

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
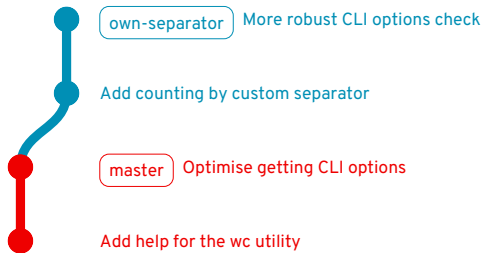
Red Hat
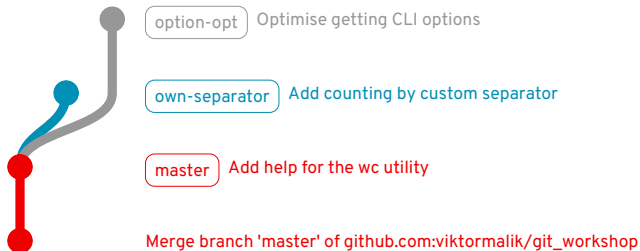
Moving branches

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

own-separator — Add counting by custom separator

master — Optimise getting CLI options

Add help for the wc utility

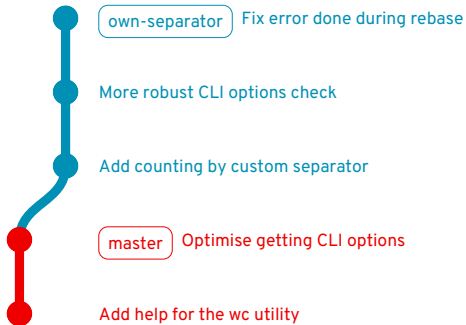Merge branch 'master' of github.com:viktormalik/git_workshop

Red Hat
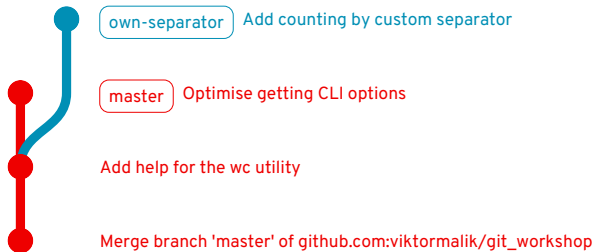
Rebasing feature branches

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

own-separator — More robust CLI options check

Add counting by custom separator

master — Optimise getting CLI options

Add help for the wc utility

Red Hat

Rebasing feature branches

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

own-separator   Fix error done during rebase

More robust CLI options check

Add counting by custom separator

master   Optimise getting CLI options

Add help for the wc utility

Red Hat

Rebasing feature branches

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

own-separator — More robust CLI options check

Add counting by custom separator

master — Optimise getting CLI options
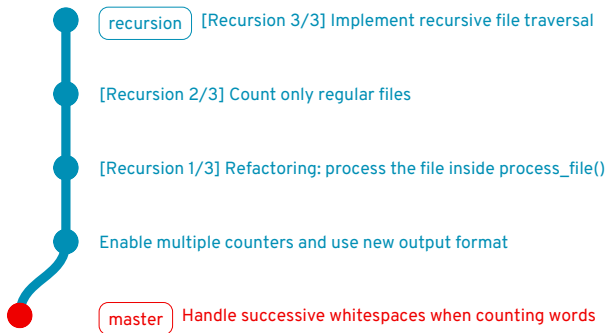
Add help for the wc utility

Red Hat
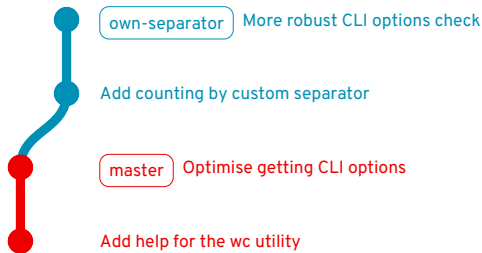
Interactive rebase

- 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!**
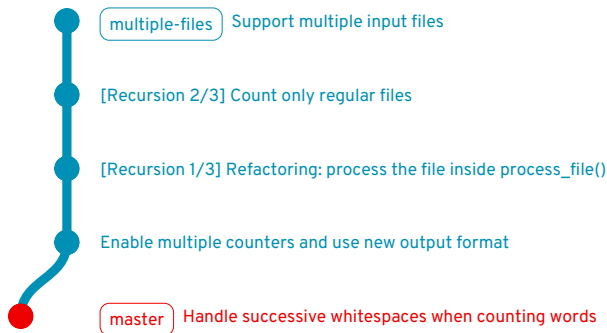 - Other developers would not be able to do `git pull`.

Red Hat

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`.

recursion — [Recursion 3/3] Implement recursive file traversal

[Recursion 2/3] Count only regular files

[Recursion 1/3] Refactoring: process the file inside process_file()

Enable multiple counters and use new output format

master — Handle successive whitespaces when counting words

Red Hat

Copying commits from other branches

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

multiple-files Support multiple input files

[Recursion 2/3] Count only regular files

[Recursion 1/3] Refactoring: process the file inside process_file()

Enable multiple counters and use new output format

master Handle successive whitespaces when counting words

Red Hat

Copying commits from other branches

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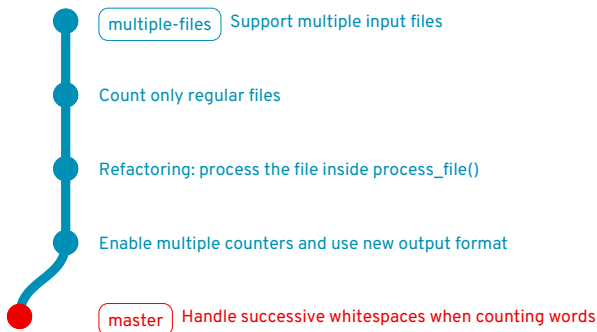
multiple-files — Support multiple input files

Count only regular files

Refactoring: process the file inside process_file()

Enable multiple counters and use new output format

master — Handle successive whitespaces when counting words

Red Hat

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.

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.

Red Hat

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.

Red Hat

Git tips and tricks

Cloning repositories with a long history

- If a repo has a long history, it may take long time to clone it.
- If the entire history is no needed, it is possible to use a **shallow copy**:
  ```
  git clone --max-depth N
  ```
- Try it with the Linux kernel:
  ```
  git clone --max-depth 1 https://github.com/torvalds/linux
  ```

Red Hat

Default push and pull into different remotes

- When using pull requests, it may be useful to pull from the **upstream** repo but push into own **fork**.

- A different remote for push can be configured using:
  `git config remote.pushdefault <remote>`

- Alternatively, this can be configured per-branch:
  `git config branch.<branch>.pushremote <remote>`

Red Hat

Signing commits

- By default, it is not possible to verify that a certain commit was truly created by the person who is stated as the author.
- Theoretically, anyone can set your name and email as theirs and commit on your behalf.

Red Hat

Signing commits

- By default, it is not possible to verify that a certain commit was truly created by the person who is stated as the author.

- Theoretically, anyone can set your name and email as theirs and commit on your behalf.

- To resolve this problem, Git offers **signing commits** using GPG keys.

- GitHub offers a nice tutorial on how to setup commit signing:
  `https://help.github.com/en/github/`
  `authenticating-to-github/signing-commits`

Red Hat

Setup your environment

There are various possibilities on how to ease your life with Git:

- **Git prompt**
  - It is possible to setup Bash prompt such that it shows the current branch, state of the directory, etc.
  - There are many tutorials on how to set the prompt
  - Some alternative shells (e.g. Fish, zsh) include Git prompt by default

Red Hat

Setup your environment

There are various possibilities on how to ease your life with Git:

- **Git prompt**
  - It is possible to setup Bash prompt such that it shows the current branch, state of the directory, etc.
  - There are many tutorials on how to set the prompt
  - Some alternative shells (e.g. Fish, zsh) include Git prompt by default

- **IDE/Editor support**
  - It is useful to see which lines were added/removed/changed from HEAD.
  - Most IDEs and editors offer a way to setup this.

Red Hat

Setup your environment

There are various possibilities on how to ease your life with Git:

- **Git prompt**
  - It is possible to setup Bash prompt such that it shows the current branch, state of the directory, etc.
  - There are many tutorials on how to set the prompt
  - Some alternative shells (e.g. Fish, zsh) include Git prompt by default

- **IDE/Editor support**
  - It is useful to see which lines were added/removed/changed from HEAD.
  - Most IDEs and editors offer a way to setup this.

- **Use tools for history inspection**
  - There is a number of tools for an easier history traversal
  - E.g. **tig**, gitk, …

Red Hat

Setup your environment

- **Command aliases**
    - Many Git commands are quite long (or have many options).
    - It is possible to setup short aliases for most commonly used commands.
    - Git offers a way to set aliases:
      ```
      git config --global alias.co checkout
      ...
      ```
      or edit `$HOME/.gitconfig`:
      ```
      [alias]
       co = checkout
       ...
      ```
    - An alternative is to setup aliases via shell

Red Hat