Lab 4 - Branching and Merging

Last updated by | kees.vanloenen | Jun 29, 2025 at 11:36 AM GMT+2

Lab 4 - Branching and Merging

Reveal.js is an open source HTML framework to build presentations. On Github the *Kennis Centrum of Info Support* is hosting a **fork** (remote copy) of this project.
In this lab you'll clone this repository and make changes to it.

Exercise 1

1. Clone the *reveal.js* source code.
   - `cd c:/git`
   - `git clone https://github.com/kc/reveal.js.git`
2. Watch the 4 folders:
   - `ls`
3. Go into the directory *reveal.js*
   - `cd reveal.js`
4. Watch the folders.
   - `ls -a`
5. Watch the commit history (`log`), one line per commit. There are many commits. You can scroll with the 'ARROW-UP' and 'ARROW-DOWN' keys or quit with 'Q'.
6. Log again, this time showing only the 10 most recent commits.
   - `git log -10 --oneline`
7. Look at the demo presentation in the browser (*demo.html*) and especially the slide 'PRETTY CODE' (`demo.html#/13`).
   We will make some minor changes here.

Exercise 2

1. To start our development, first create a branch based on 'master' with the name 'theme' (`branch`) and switch to this branch (`switch`).
2. Check the result of both commands:
   - `git branch`
3. Open *demo.html* in *VSCode* (`code demo.html`) and change the theme in line 18:

```
<link rel="stylesheet" href="css/theme/white.css" id="theme" />
```

4. Look at the results in the browser.
5. Commit the changes.
6. Inspect the last 2 commits.

Exercise 3

1. Let's work on another change. Create another branch based on master named 'code-highlighting' and switch to this new branch. One way of doing that:

```
git switch master
git branch code-highlighting
git switch code-highlighting
```

   Pro-tip. All can be done in one command:

```
git switch -c code-highlighting master
```

2. Change the stylesheet for code syntax highlighting in line 21 in *demo.html* :

```
<link rel="stylesheet" href="lib/css/github.css" />
```

3. Look at the results in the browser (slide 'PRETTY CODE' via demo.html#/13).

4. Commit the changes.

5. Inspect the last 2 commits.
   Which local branches do you see?

6. Extend the log command:
   `git log -3 --oneline --graph --all`

   - **`--graph`** = use a graph
   - **`--all`** = include all branches
   - **`-3`** = as we include all branches, let's pick a higher number

Exercise 4

1. Let's try merging these changes into a new branch...
   Create a third new branch 'combined' based on master and switch to it.

2. As an experiment, try to remove the branch 'theme':

   - `git branch -d theme`

     Why can't you?

3. Merge branch 'theme' with the current branch 'combined':

   - `git merge theme`

   Look at the output you get when merging. Why can Git use a fast-forward merge?

4. Watch the last 2 commits.

   - Make a drawing for yourself what has happened.
   - Rerun the log command with graph (last step in previous exercise)

5. Now remove branch 'theme' with the same command you tried earlier.

- `git branch -d theme`

6. Also rerun the logging statement:

   - `git log -3 --oneline --graph --all`

   Can you spot the difference?

   Another observation: There is no direct line between the commit in branch `code-highlighting` and the commit in branch `combined` ...

7. Also merge the branch 'code-highlighting' into `combined`. This time a new commit will be generated! Keep the suggested commit message and close the text editor.

   (The previous merge didn't result in a new commit. This one did. Can you explain why?)

8. Look at the results in the browser.

9. Again try to understand of what has happened and let a 'powered-up' log statement help you.

   - `git log -4 --oneline --graph --all`

   Branch 'combined' is three commits ahead of 'master'.

10. Remove the branch 'code-highlighting'. If you did it correctly, there are two local branches left now: 'combined' and 'master'.

    a. Rerun the log command from the previous step.
    b. List the two branches (`branch`).
    c. Branch 'combined' has two changed lines in *demo.html* compared to 'master'.

    - Show them with: `git diff master combined`
    - What would be shown if you had run: `git diff combined master`

Exercise 5

Let's pretend that a colleague has made a change in the meantime and adapted the theme to 'sky'.

1. Switch back to the 'master' branch (`switch`)

2. Again change the theme in line 18 of *demo.html*:

```
<link rel="stylesheet" href="css/theme/sky.css" id="theme" />
```

3. Inspect this change immediately. Compare the current working directory with the index.

   - `git diff`

4. Commit the change in branch 'master'

5. Compare again. Can you guess the outcome?

   - `git diff`

6. Try to merge branch 'combined' into 'master'.

   a. Look at the output of the command and see that the automatic merge didn't succeed. One commit changed the theme to 'white' and the other to 'sky'. It is up to you to point out which change we want to keep.

b. Mark the *CONFLICT* message. Git Bash also shows that a merge action is going on: *(master|MERGING)* is shown.

7. Open *demo.html* in the editor and see how Git has presented the conflict in the file:

```
<<<<<<< HEAD
<link rel="stylesheet" href="css/theme/sky.css" id="theme" />
=======
<link rel="stylesheet" href="css/theme/white.css" id="theme" />
>>>>>>> combined
```

HEAD refers to the changes in the current branch 'master', from where we initiated the merge command.

8. Let's decide to keep the 'sky' theme. Remove the line with the 'white' theme and remove the `<<<`, `===` and `>>>` lines that git added. Save the file as it should be.

9. Check the status (`git status`)

10. Commit the changes. As a commit message you can use something like 'merge from combined'.

11. Now the merge action is completed.

    a. Mark that there is no *(master|MERGING)* anymore in the console.
    b. Look at the result in the browser (demo.html#/13). If all went fine, we'll see the 'sky' theme combined with the 'github' theme for code highlighting.
    c. Also look at the commit history:

    - `git log -6 --graph --oneline`

Exercise 6

(if time permits)

Let's practice *squashing* multiple commits into one, using a so-called interactive rebase.

1. Make in a new branch 'feat/x' 3 commits: in each commit you add a word to a different paragraph element (`<p></p>`) in 'demo.html'. Inspect the last 4 commits.

Note that you haven't pushed any change to a remote repository yet. Only then it's safe to rewrite history. Let's squash the 3 commits into one. It'll keep your history clean.

1. First start an interactive rebase (`git rebase -i HEAD~3`)
   In the editor that opens, leave the first commit as `pick` and change `pick` to `squash` (or `s`) for the second and third commit.

2. Save and close the editor.

3. Another editor window will open to combine the commit messages into one clear message (simply ignore lines starting with `#`).

4. Save and close the editor.

5. Check your commit history again to confirm the squash.

6. Under the hood commits are immutable. So the old commits are not really gone. For about 28 days they will be kept in a kind of archive which you can inspect with `git reflog` .

Exercise 7

(if time permits)

Sometimes you make a typo in a commit message. It's possible to update the commit message with the `--amend` flag.

1. In 'demo.html' update the `<h3></h3>` element on line 45 with 'The HTML/JavaScript Presentation Framework'
2. Commit the change with a typo in the message "Corected Framework title"
3. View the most recent commit

Note that you haven't pushed any change to a remote repository yet. Only then it's safe to rewrite history. Let's amend the message of the last commit. It'll keep your history clean.

1. Instruct git to amend the last commit (`git commit --amend -m "Corrected Framework Title"`)
2. View the last two commits
3. How to view the old 'removed' commit?