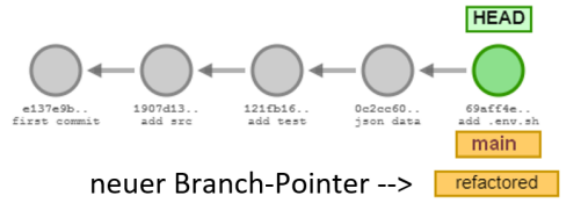
Aufgaben zur Übung Software Engineering SE-1 G5

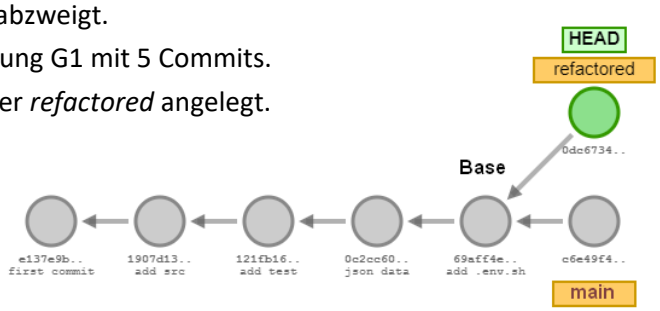
Git - Branches, Punkte: 6

Branches sind eigenständig fortgesetzte Commit-Folgen. Commits erfolgen stets auf einen Branch, der durch Commits wächst. Sollen Entwicklungen als eigenständige Commit-Folgen abgezweigt werden, wird ein neuer ***Branch-Pointer*** („Branch Name“) erstellt, auf den dann die weiteren Commits erfolgen. Damit wächst dieser Branch. Der vorherige Branch bleibt unverändert. Als ***Base*** des neuen Branches bezeichnet man das Commit, an dem der Branch abzweigt.

Die Abbildung links zeigt den *main*-Branch aus Übung G1 mit 5 Commits.

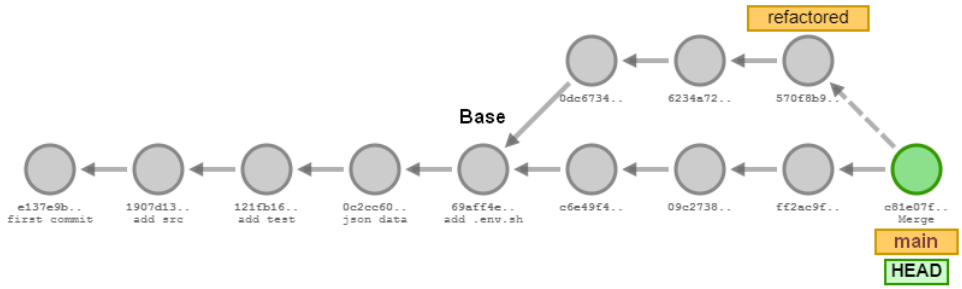
Am Ende von *main* wurde ein neuer Branch-Pointer *refactored* angelegt.



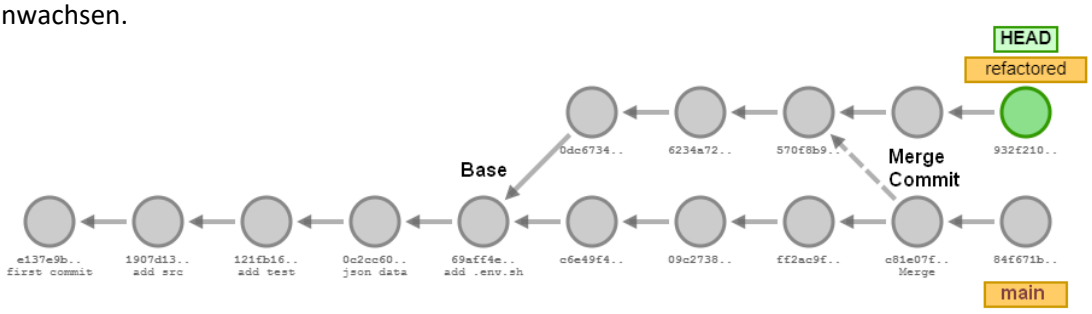


Die rechte Abb. zeigt, dass auf den neuen Branch *refactored* ein neues Commit erfolgte (oben). HEAD steht auf diesem Commit, d.h. das Projektverzeichnis ist auf diesen Stand synchronisiert, *refactored* ist der aktive Branch. Auch auf den *main*-Branch ist ein Commit erfolgt. Base ist das Commit, an dem der Branch *refactored* abgezweigt wurde, d.h. an dem der Branch Pointer angelegt worden ist.

Das Zusammenführen von Branches heißt ***Integration*** oder ***Merge***. Die folgende Abbildung zeigt eine Variante mit Erstellung eines neuen ***Merge Commit*** im **Zielbranch** *main*. Stände aus beiden Branches werden auf *main* in einem neuen Commit vereint. *Merge Commits* haben zwei Vorgänger-Commits.



Nach einem Merge bestehen die Branch-Pointer fort und können durch weitere Commits weiter anwachsen.



1.) Auf: http://git-school.github.io/visualizing-git befindet sich ein anschauliches Lernsystem, in dem Sie einfache *git*-Kommandos eingeben und deren Wirkung visuell nachverfolgen können.

Das System verwendet noch die alte Bezeichnung: „master“ für den Haupt-Branch, anstelle der neueren Bezeichnung: „main“.

Kommandos können links unten eingegeben werden. Das Kommando „help“ zeigt alle Kommandos an. Das Kommando "undo" setzt die letzte Aktion zurück, "clear" löscht das Repository.

Neue Commits legt man vereinfacht mit "git commit" an (ohne vorausgehendes Staging mit *git add*).

Erstellen Sie initial zwei neue Commits und taggen Sie das letzte Commit mit *Base*.

```
git commit                ; 1. Commit erstellen
git commit                ; 2. Commit
git tag Base              ; Commit mit „Base“ taggen
```

2.) Erstellen Sie einen neuen Branch-Pointer: *refactored*:

```
git branch refactored     ; Branch „refactored“ erstellen
```

3.) Setzen Sie die Commit-Folge auf dem Master-Branch fort:

```
git commit                ; 4. Commit erstellen
git commit                ; 5. Commit
```

4.) Schalten Sie auf den Branch *refactored* um:

```
git checkout refactored   ; auf Branch umschalten, um dort mit Commits fortzusetzen
```

5.) Setzen Sie die Commit-Folge auf dem Branch *refactored* fort:

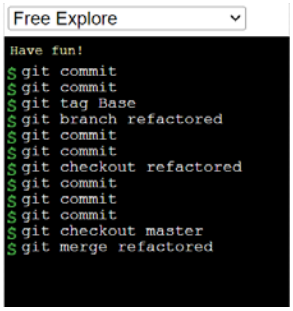
```
git commit                ; 1. Commit auf Branch refactored
git commit                ; 2. Commit auf Branch refactored
git commit                ; 3. Commit auf Branch refactored
```

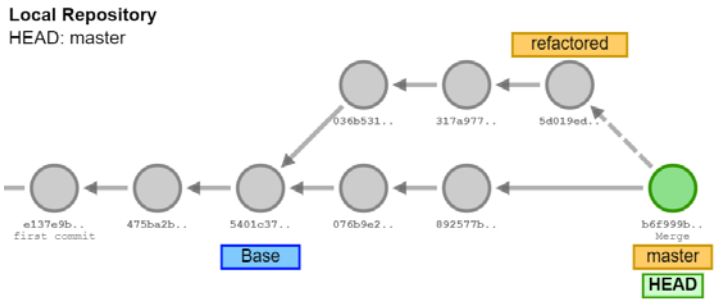
6.) Schalten Sie auf den Branch *master* zurück:

```
git checkout master
```

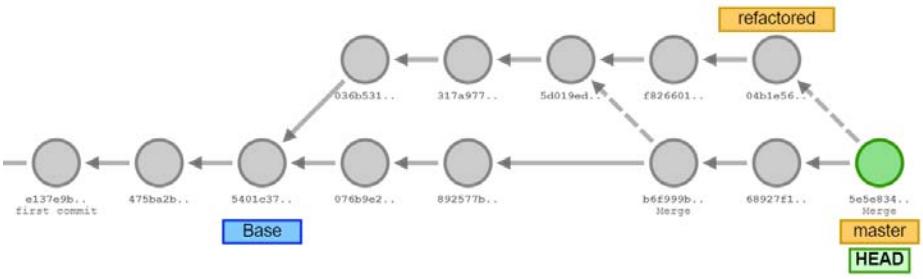
Mergen Sie den Branch *refactored* nach *master*. Man muss sich stets im Ziel-Branch befinden (hier ist das *master*), in den man einen anderen Branch mergen möchte. Komplettieren Sie das Merge und fertigen Sie einen Screenshot der Anzeige an. [1 Pkt]

```
git merge refactored
```



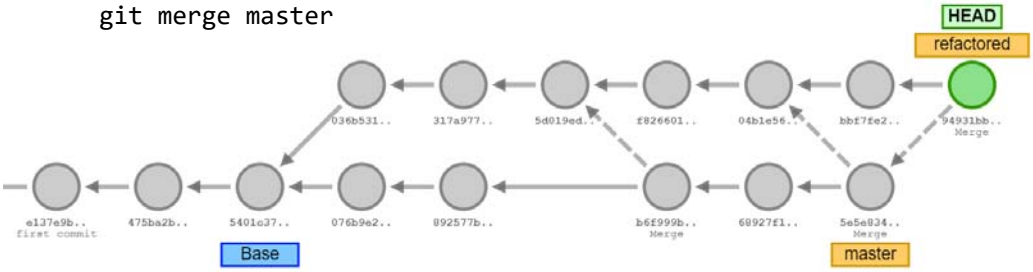


7.) Setzen Sie die Branches wie im folgenden Bild gezeigt fort (Screenshot der Anzeige). [1 Pkt]



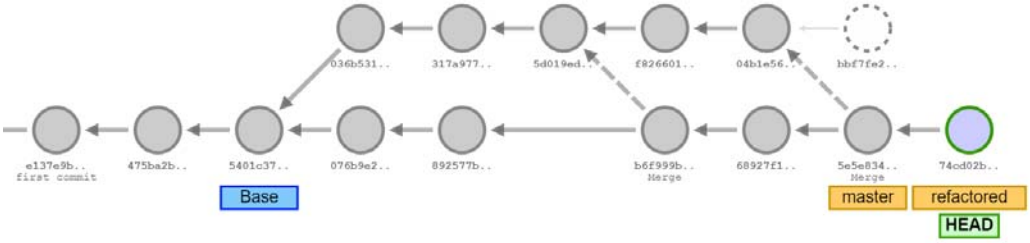
8.) Committen Sie nach dem Merge auf *refactored* undMergen Sie den Branch *master* nach *refactored*. Was ist der Sinn dieses Merges? (Screenshot und Antwort).

```
git merge master
```



9.) Setzen Sie das Merge mit undo zurück und verwenden Sie statt Merge ein Rebase. Beobachten Sie die Animation (Screenshot und Antwort unten). [1 Pkt]

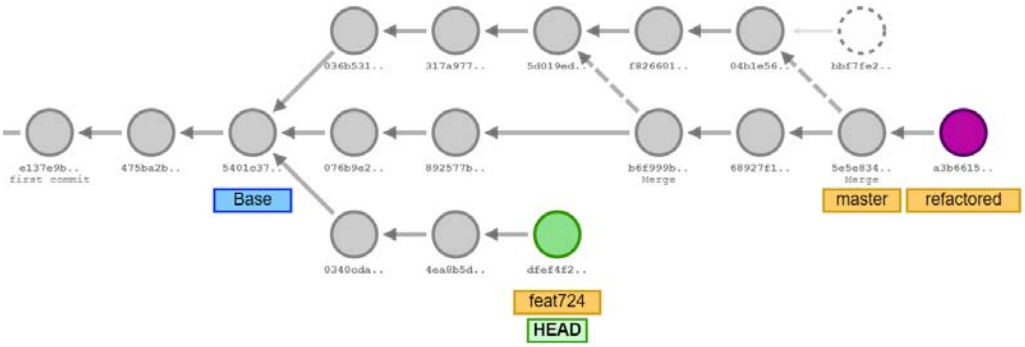
```
git rebase master
```



Was passiert, wenn man mit einem Commit auf *master* fortsetzt und anschließend auf *refactored* ?

10.) Zweigen Sie an *Base* einen weiteren Branch *feat724* ab mit zwei Commits. Sie müssen dazu an dieses Commit mit *checkout Base* navigieren („detached HEAD“ state).

a) Versuchen Sie, an *Base* die zwei Commits direkt anzulegen. Warum funktioniert das nicht?
b) Was ist ein „detached HEAD“ state?
c) Was müssen Sie tun, um auf Branch *feat724* die zwei Commits anlegen zu können (Screenshot und Antworten)? [1 Pkt]



11.) Notieren Sie die Aktionen, welche den Stand von *feat724* in den *master*-Branch mergen.

12.) Was machen die Kommandos (mit HEAD wie im letzten Bild gezeigt):

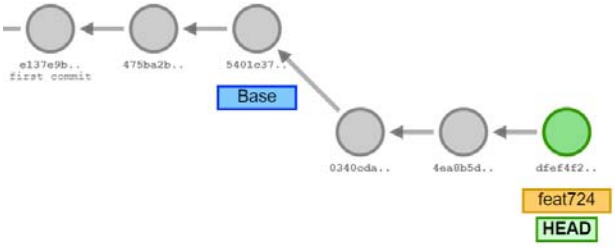
```
git reset HEAD~2
git reset HEAD~4
```

Welche Auswirkungen ergeben sich für Branch *feat724* ?

13.) Was ist der Unterschied zwischen (Antwort bitte notieren) [1] : [1 Pkt]

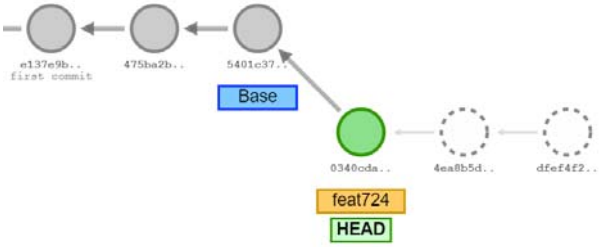
```
git reset --soft HEAD~2
git reset --mixed HEAD~2
git reset --hard HEAD~2
```

14.) Gegeben ist der Ausschnitt von Branch *feat724*:

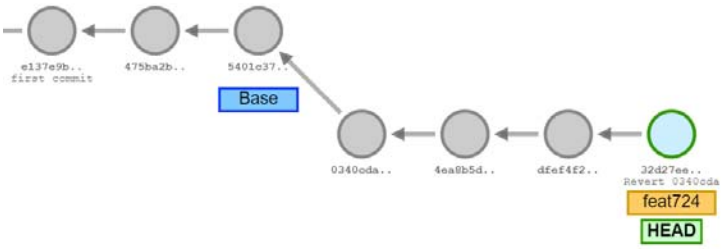


Was ist der Unterschied zwischen (Antwort bitte notieren) [2] : [1 Pkt]

```
git reset HEAD~2
```



und `git revert HEAD~2`



[1] https://stackoverflow.com/questions/24568936/what-is-difference-between-git-reset-hard-head1-and-git-reset-soft-head.
[2] https://stackoverflow.com/questions/8358035/whats-the-difference-between-git-revert-checkout-and-reset.