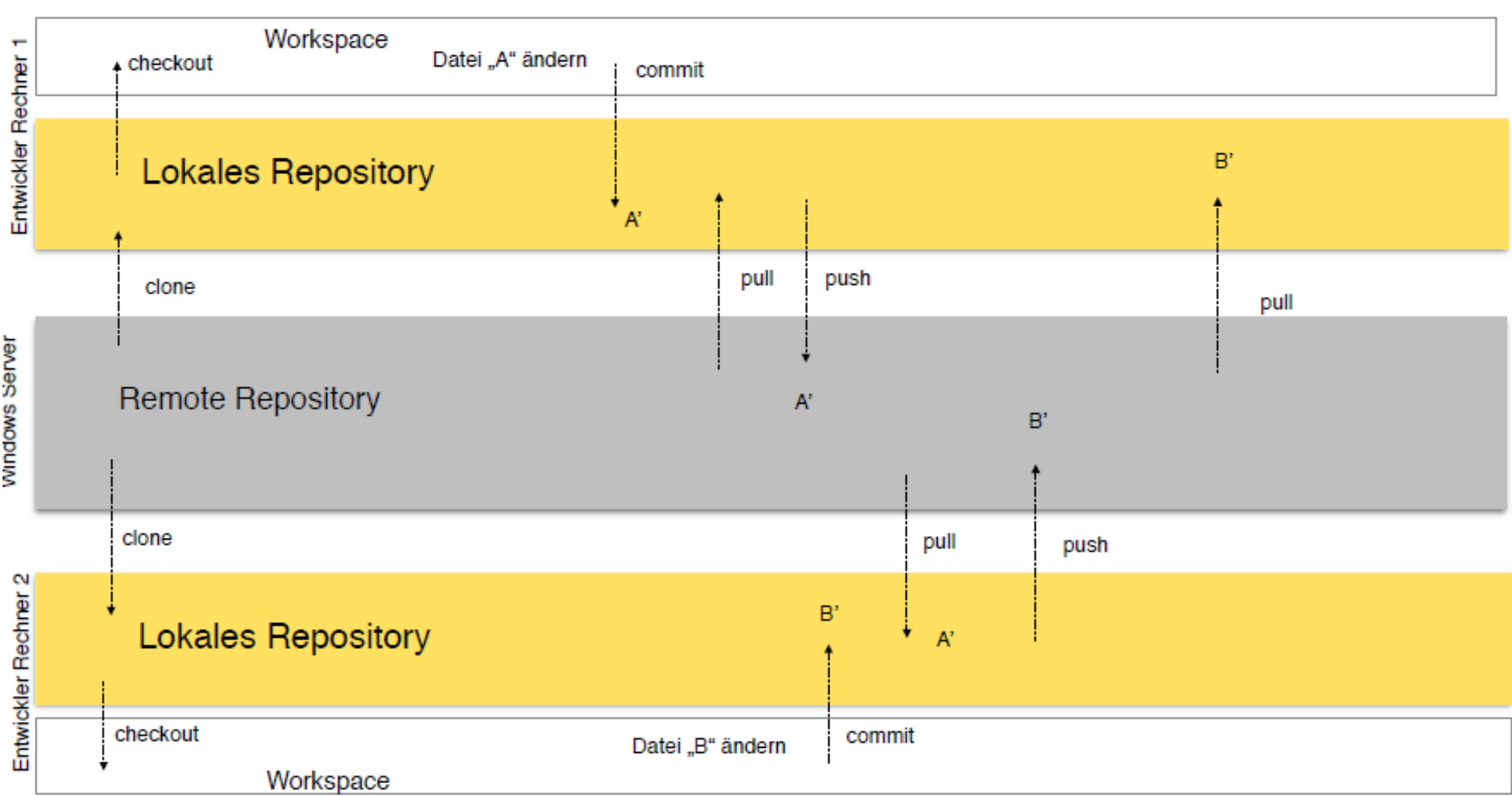


Grundlagen

Git arbeitet mit mehreren Repositories. Ein "commit" überträgt die Änderungen nur vom Workspace in das lokale Repository, ein "push" überträgt die Änderungen vom lokalen Repository in das Remote-Repository. Ein "pull" holt die Änderungen des Remote-Repositories in das lokale Repository.



Links

- http://wiki.eclipse.org/EGit/User_Guide

Konfiguration

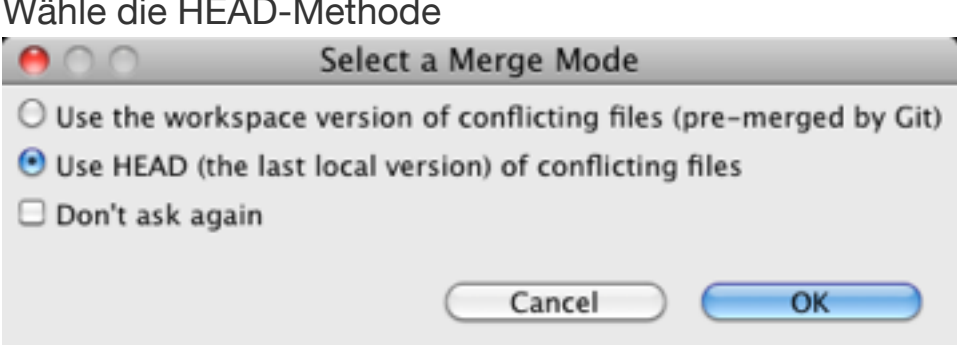
- `git config --global user.name "Friedrich Kiltz"`
- `git config --global user.email "f@kiltz.de"`

Selbsterklärende Nützlichkeiten

- `git status`
- `git clone user@server:project.git`

Konflikte lösen

s. http://wiki.eclipse.org/EGit/User_Guide#Resolving_a_merge_conflict

- Wähle auf dem übergeordneten Element: **Team > Merge Tool**
 - Wähle die HEAD-Methode
- 
3. Editiere die lokale Version
 4. Wähle **Team > Add** um die gemergten Dateien zum Stage hinzuzufügen.
 5. Committe die Änderungen
 - 6.

Entfernen von Dateien, die in Git sind aber laut .gitignore gelöscht werden sollen:

```
git rm --cached -r *
```

Für Platzhalter muss ein Schrägstrich vor den * gesetzt werden, da sonst der RegEx * gemeint ist:

```
git rm log/\*.log
```

Log-Informationen

```
git log
git log --since=2016-04-18 --reverse
```

oder mit Anzeige der geänderten Dateien:

```
git log --since=2.days --name-status --reverse
```

die letzten zwei Einträge mit ihren Änderungen

```
git log -p -2
```

Ein Remote-Repository hinzu fügen

```
git remote add <kurzName> <URL>
```

dann kann man aus dem neuen Repository Fetchen und dahin auch puschen:

```
git fetch <kurzName>
```

```
git push <kurzName> <branch>
```

```
git push origin master
```

Tagging

Mit einem Tag kann man einen speziellen Punkt in der History benennen - z.B. einen Meilenstein oder einen Release-Punkt (V1.0)

Anzeige der Tags mit

```
git tag
oder spezieller:
git tag -l 'v1.8.5*'
...zeige alle Tags, die mit 'v1.8.5' beginnen.
```

Erzeugen eines Tags

```
git tag -a v1.4 -m 'my version 1.4'
```

zum Anzeigen eines bestimmten Tags:

```
git show v1.4
```

Tags können auch nachträglich einem Commit zugeordnet werden. Dafür gibt man die Checksumme oder einen Teil davon beim Taggen an

Tags müssen noch gepusht werden:

```
git push origin --tags
```

und können dann auch ausgecheckt werden:

```
git checkout -b version2 v1.4
```

Branch

1. Beginne ein neuen Aufgabe (Issue 53) in einem neuen Branch:

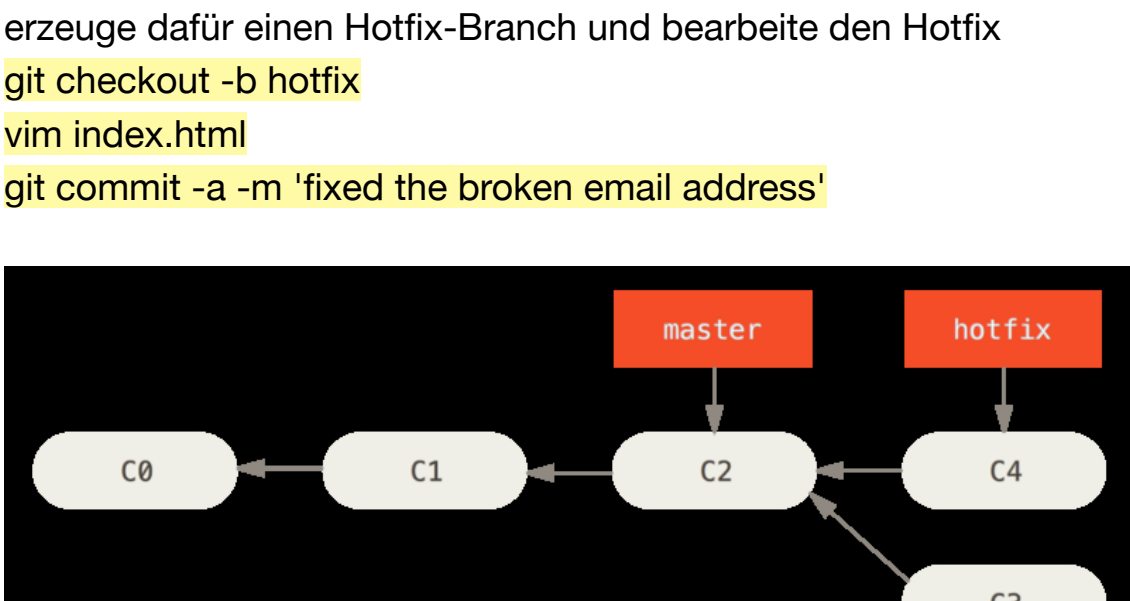
```
git checkout -b iss53
was die Kurzform von
git branch iss53
git checkout iss53
ist.
```

2. Tue irgend etwas in der neuen Branch und Committe es.

```
vim index.html
git commit -a -m 'added a new footer [issue 53]'
```

3. Anruf: im "master" muss etwas geändert werden!

```
git checkout master
erzeuge dafür einen Hotfix-Branch und bearbeite den Hotfix
git checkout -b hotfix
vim index.html
git commit -a -m 'fixed the broken email address'
```



4. Merge den Hotfix zum Master

```
git checkout master
git merge hotfix
```

Da vorher keine Änderungen auf dem Master gemacht wurden reicht ein "fast-forward", ein Weitersetzen des Merge auf den HotFix-Commit

Der Hotfix kann gelöscht werden:

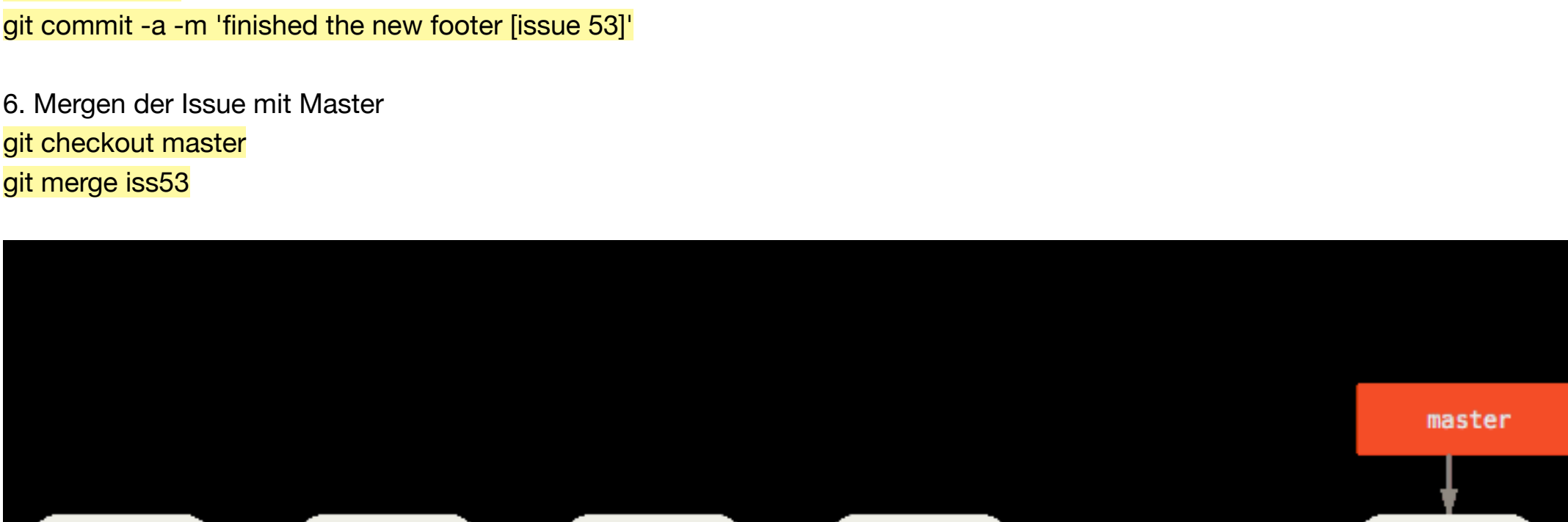
```
git branch -d hotfix
```

5. Weiter mit Issue 53

```
git checkout iss53
vim index.html
git commit -a -m 'finished the new footer [issue 53]'
```

6. Mergen der Issue mit Master

```
git checkout master
git merge iss53
```



Das Ergebnis des Merges ist ein Commit (C6) der aus C2, C4 und C5 entstand.

Die Branch für die Aufgabe 53 kann nun gelöscht werden:

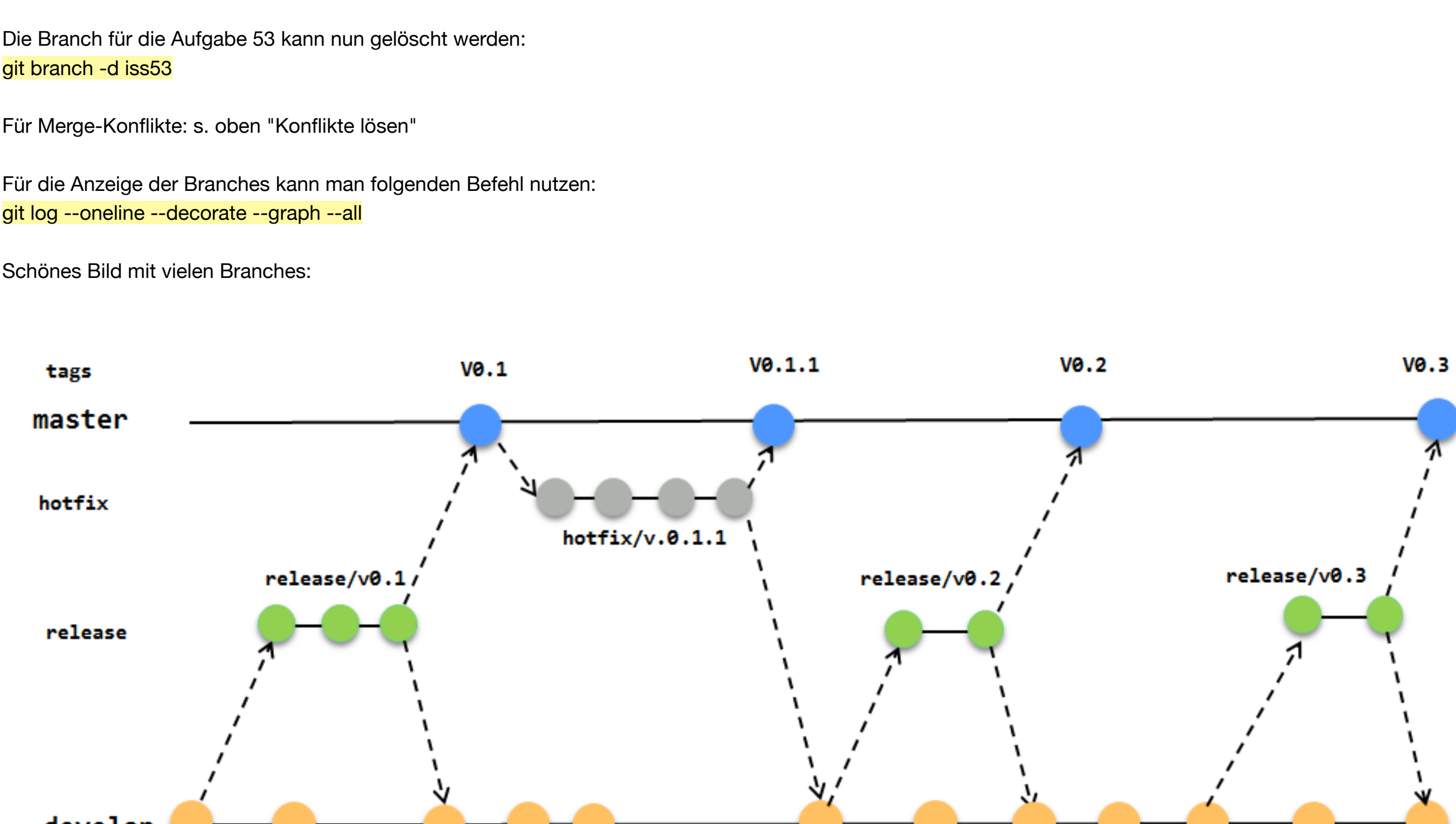
```
git branch -d iss53
```

Für Merge-Konflikte: s. oben "Konflikte lösen"

Für die Anzeige der Branches kann man folgenden Befehl nutzen:

```
git log --oneline --decorate --graph --all
```

Schönes Bild mit vielen Branches:



aus <https://fpy.cz/pub/slides/git-workshop/#/step-21>

Links

- http://danielkummer.github.io/git-flow-cheatsheet/index.de_DE.html
-

Das Buch Pro Git (leider in Plattdeutsch)