



**JAVA 达摩班**

# **Git & Github最佳实践**



# 详解Git 与版本控制

1

# 版本控制

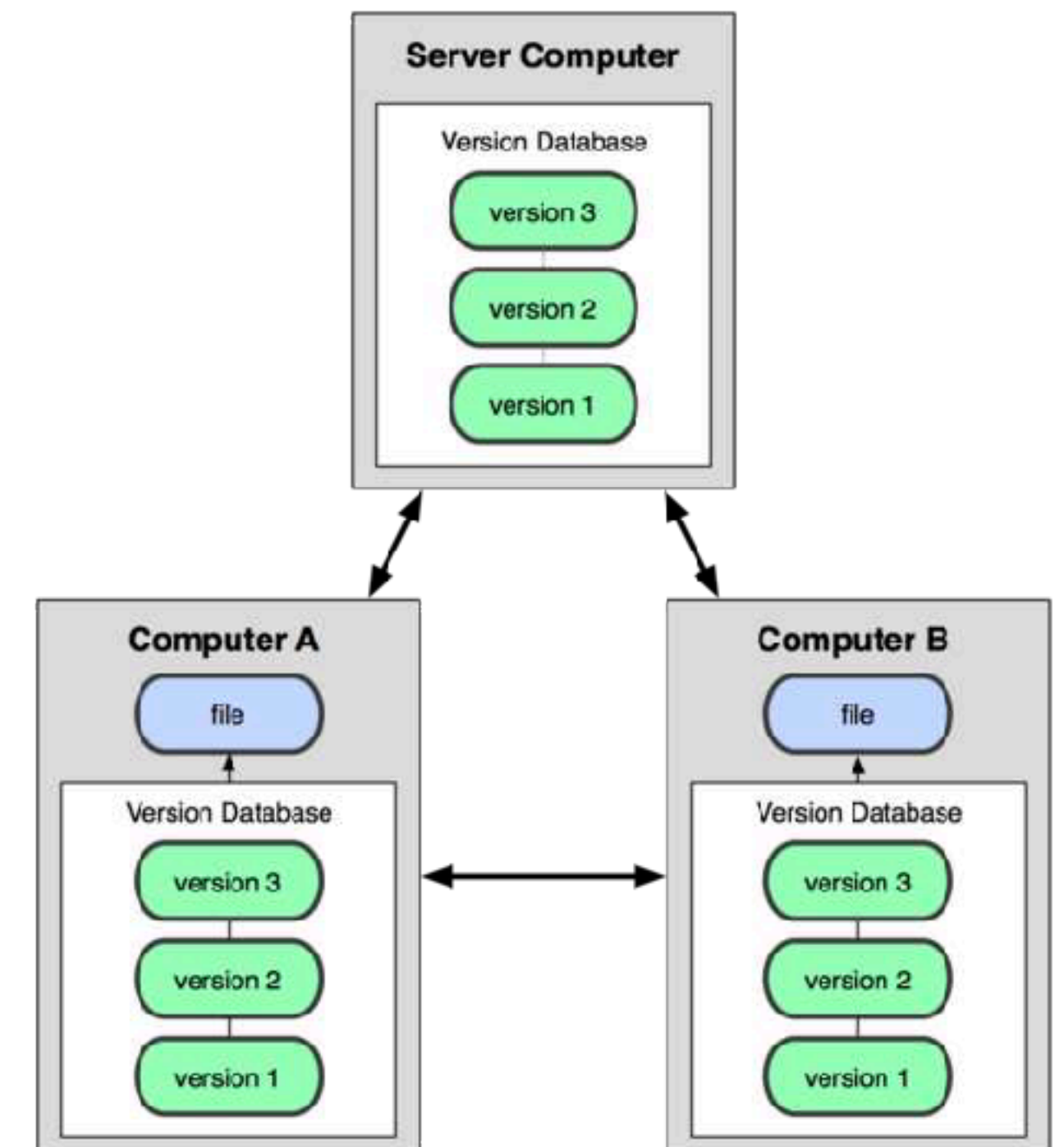
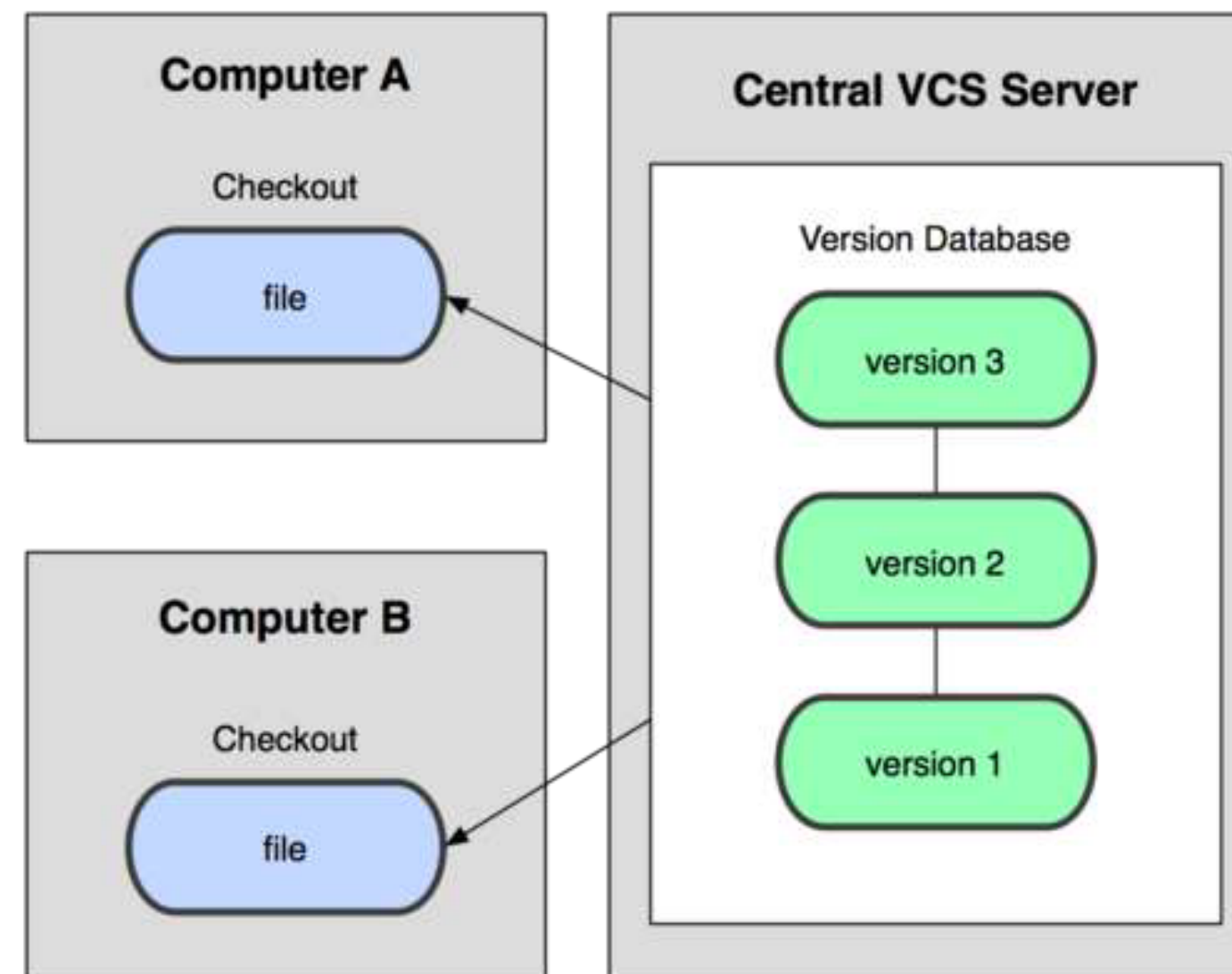
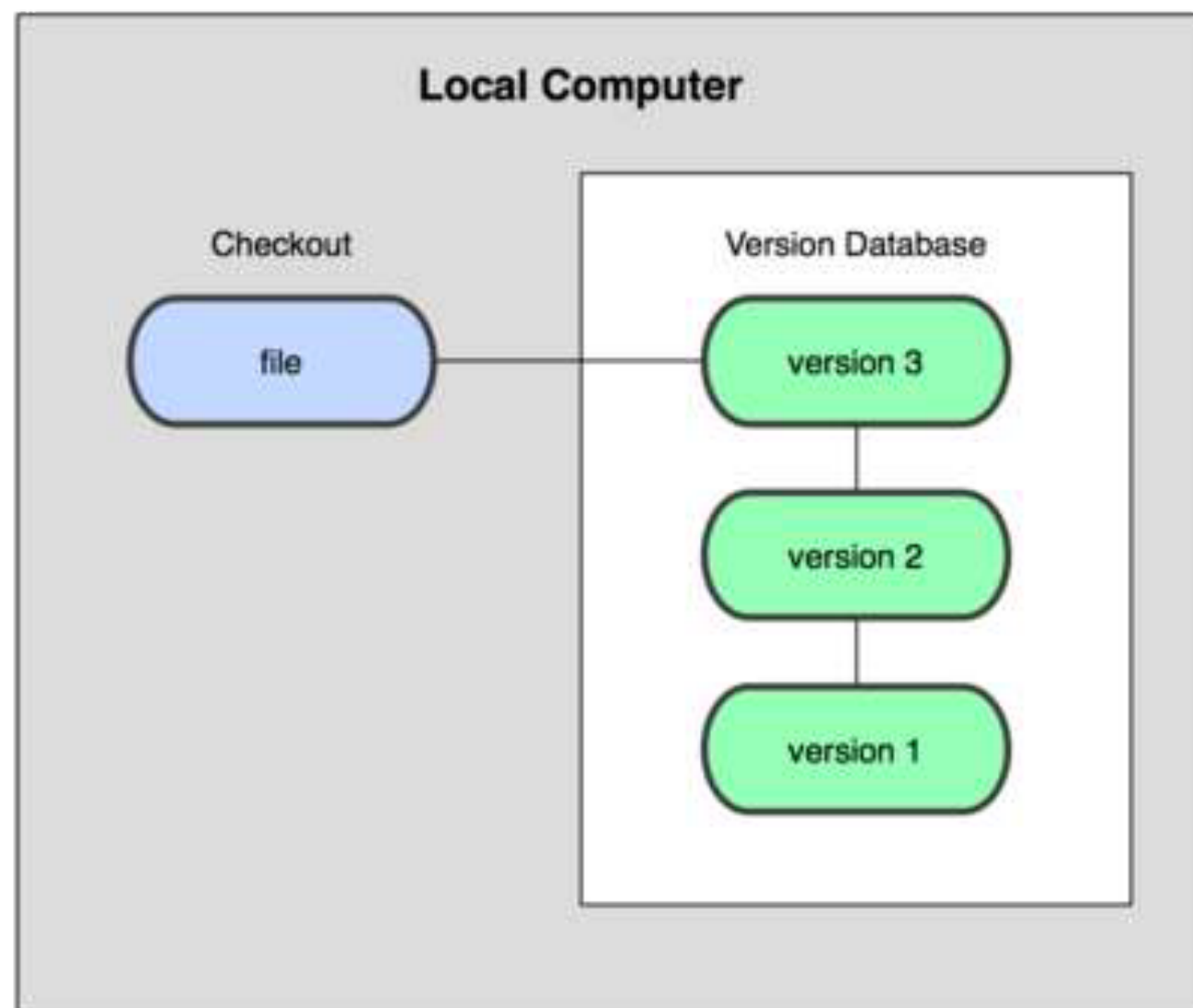
版本控制（version control system, VCS）是一个软件系统，它记录文件变化，或者设置时间点上的软件版本，这样就可以回看历史版本

VCS类型：

1. 本地VCS

2. 中央VCS

3. 分布式VCS



# 为什么Git

英文解释：饭桶，笨蛋，无用的人

命名源自于Linus的自嘲

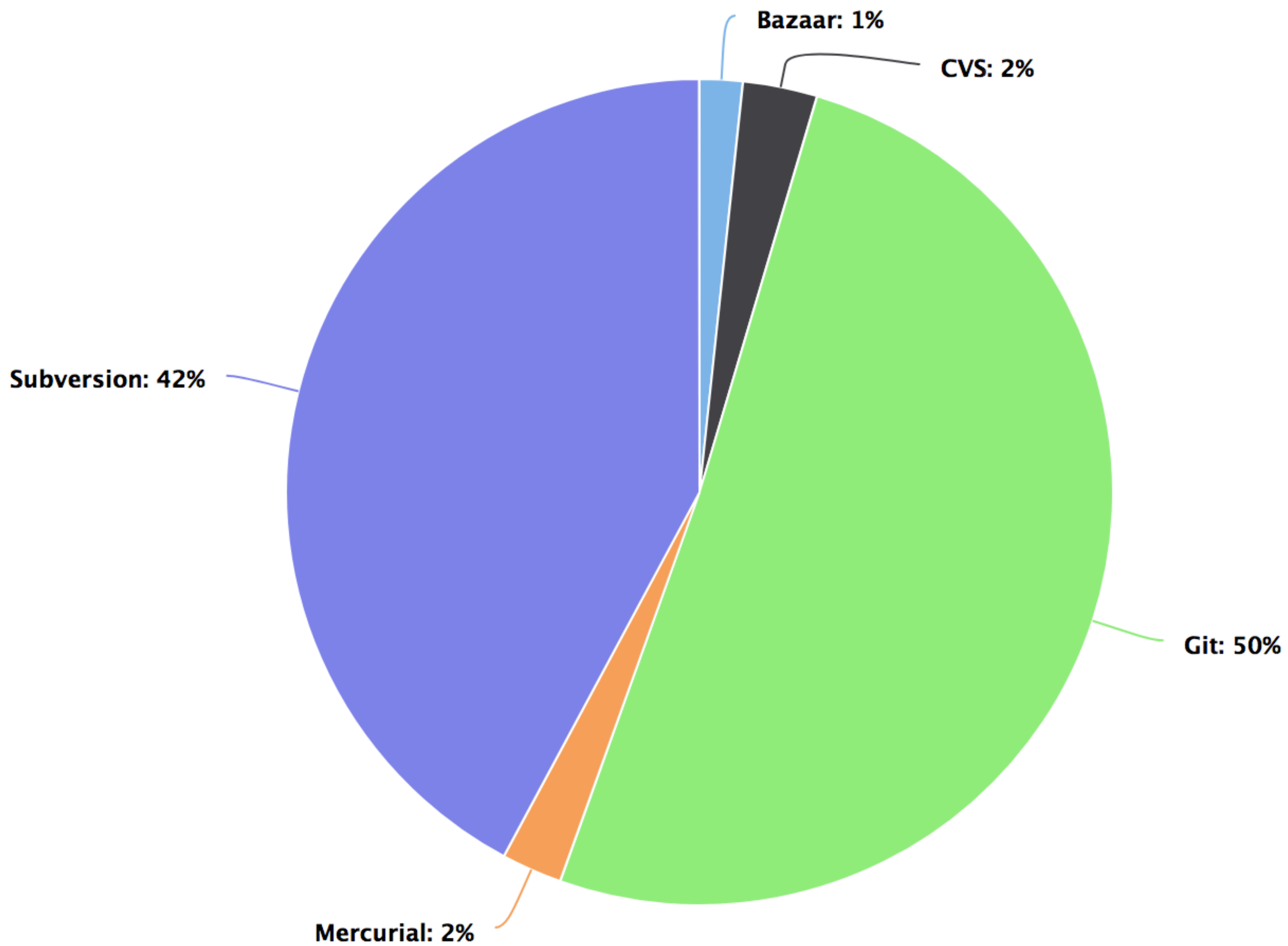
“I'm an egotistical bastard, so I name all my projects after myself. First Linux, now Git”

Linus Torvalds, PC World. 2012-07-14

源于2005年，Linux需要版本管理工具，Kernel 2.6.12第一次使用Git

Git 1.0发布于2005年12月





# Git核心概念

## 快照 (Snapshot)

- 用于追踪文件历史
- 记录文件在某个时间点的状态
- 你来决定什么时间，对哪些文件“拍照”
- 你可以回看之前的快照



# Git核心概念

## 提交 (Commit)

- 提交就是创建快照的动作
- 既是动词也是名词
- 项目可以看作是有提交组成的
- 提交包括三部分：
  1. 文件如何变化的信息
  2. 创建对该提交的引用指向“父提交”
  3. hash值

# Git核心概念

## 代码仓库 (Repository)

- 简称 repo
- 包括所有代码文件及其历史记录
- 包括所有的提交
- 可以在本地，也可以在远程 (Github)
- 将远程repo复制到本地叫做克隆clone
- 团队协作
- 同步本地没有的commit叫做pull
- 同本远程没有的commit叫做push



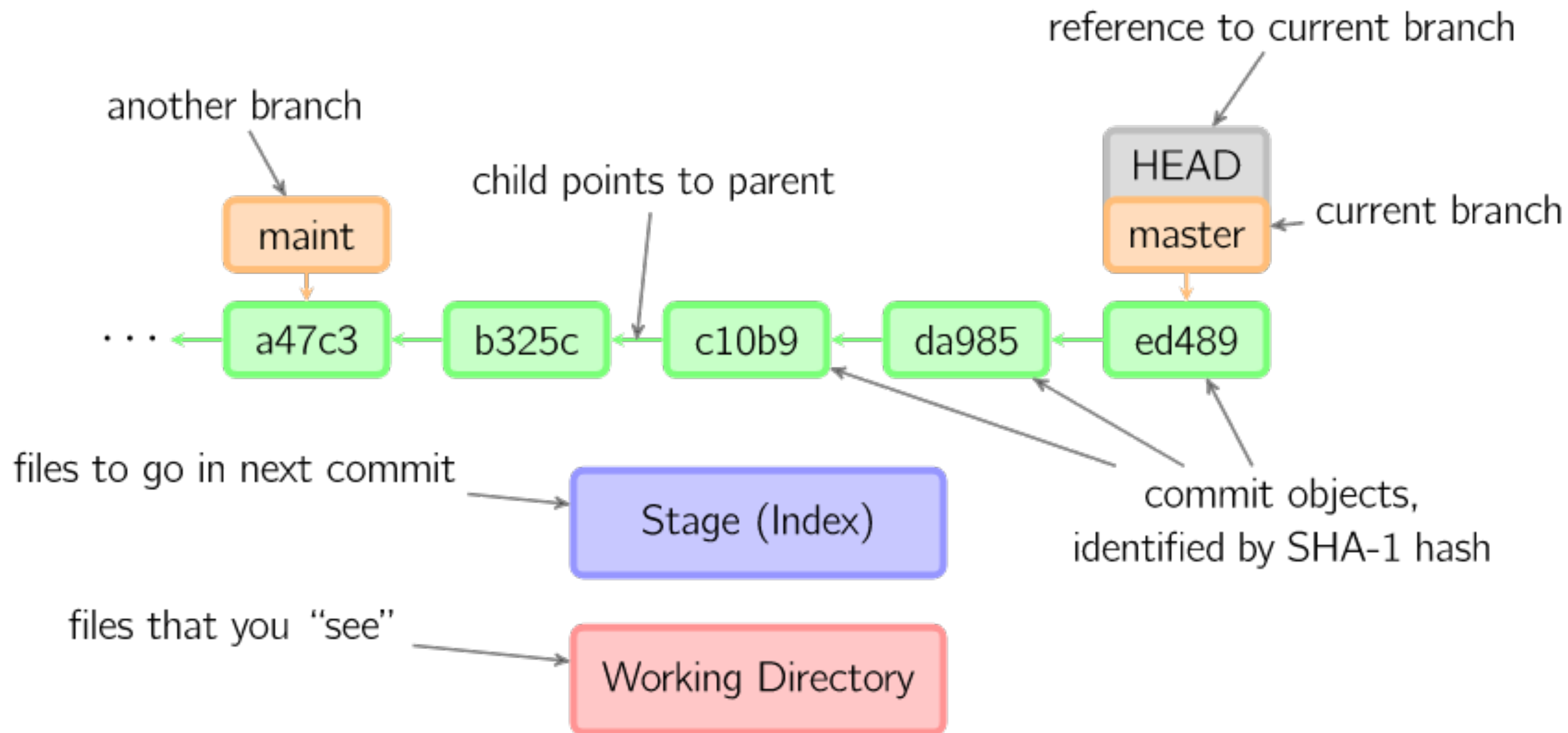
# Git核心概念

## 分支 (Branch)

- 所有提交都发生在分支上
- 可以有很多分支
- 默认创建的主分支叫做master branch



# Git核心概念

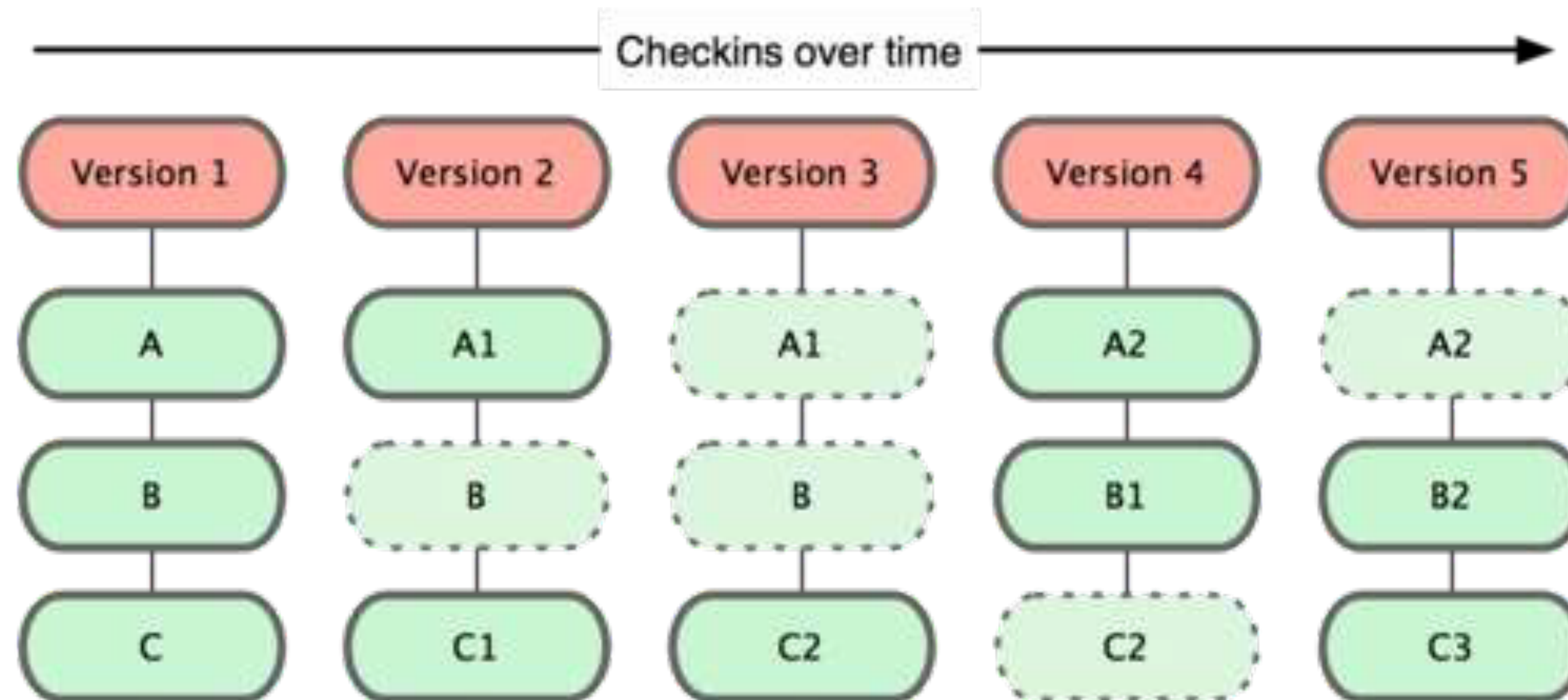


# Git原理

基于快照，而不是记录文件变化

Git是分布式版本控制系统（DVCS）

Git中的数据是快照集合，它是一个小型文件系统。每次提交，它会“拍照”记录当时的文件，并存储索引指向那组快照。



# Git原理

大部分操作都发生在本地文件和资源

Git有三种文件状态：提交(committed)，修改(modified)和暂存(staged)

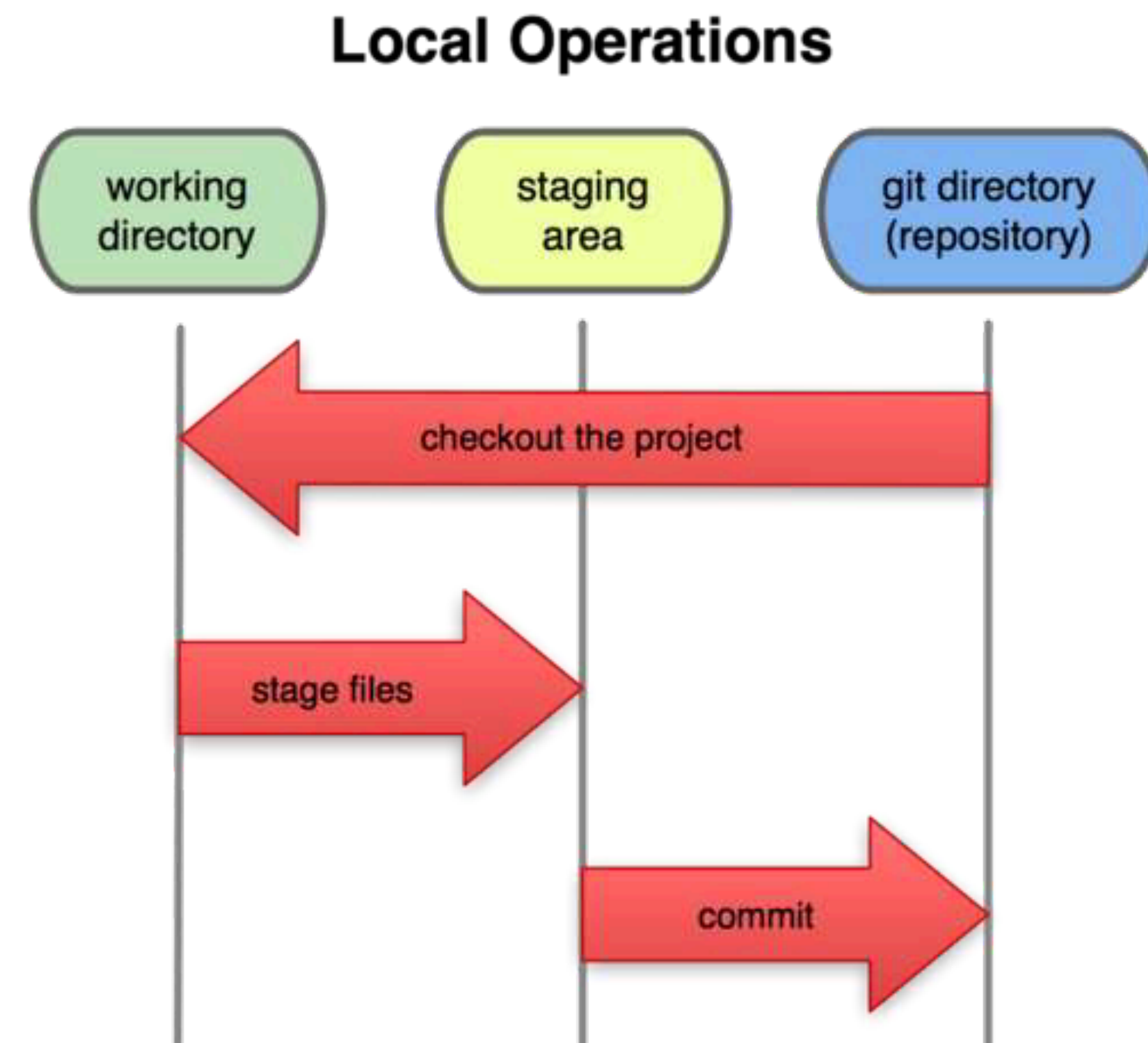
提交是指数据已经安全的存储在本地数据库。

修改是指改变文件，但没有提交到本地数据库。

暂存是指对修改的文件做了标记，暂存准备下次提交

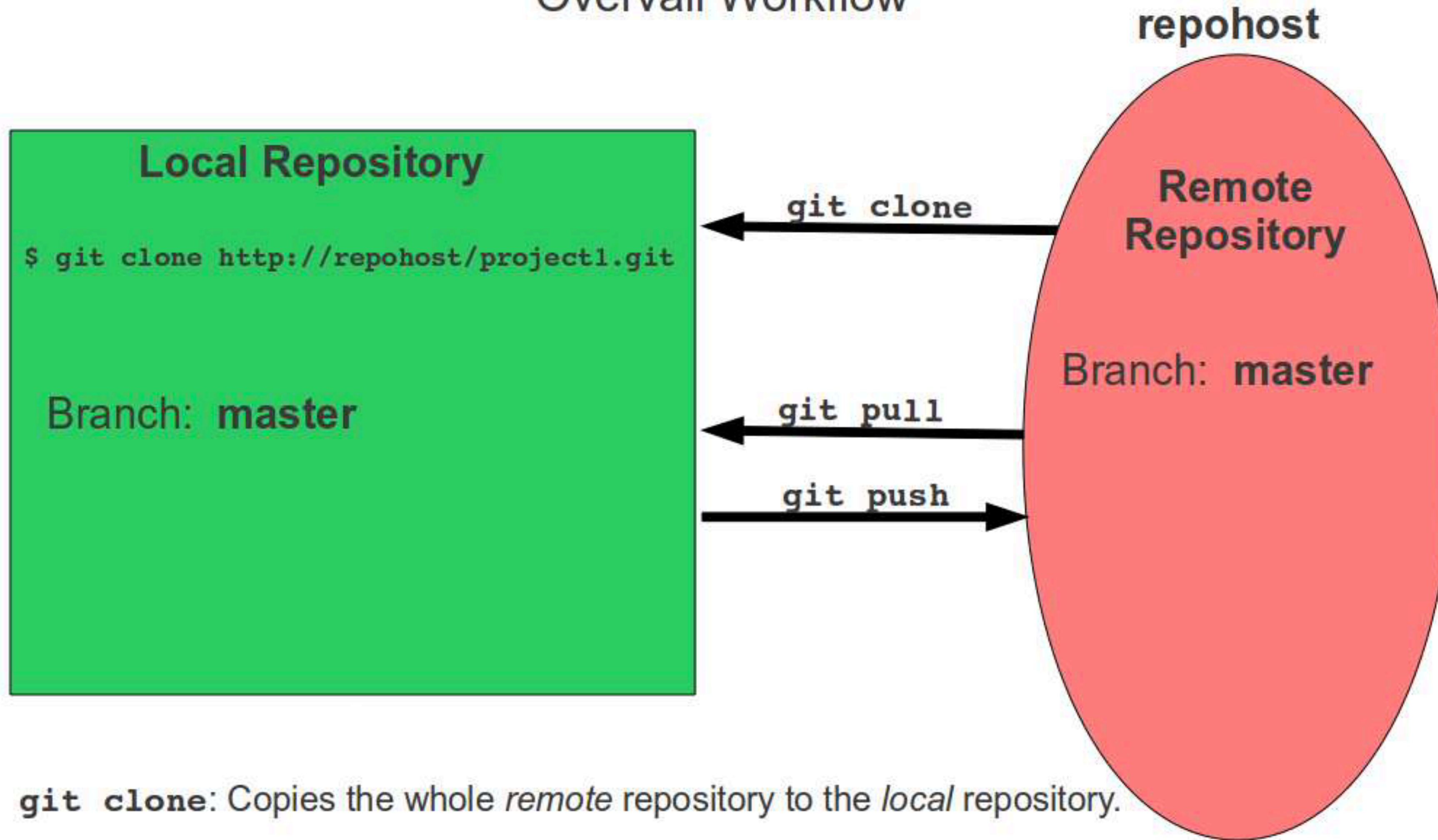
## Git工作流程

1. 在工作目录中修改文件
2. 对修改文件添加快照，放入暂存区
3. 提交修改，暂存区域会将快照同步到远程Git目录





## Git Remote Repositories: The High-level (“10,000 foot”) View: Overall Workflow



**git clone**: Copies the whole *remote* repository to the *local* repository.

**git pull**: Retrieves any updates from the remote repository that aren't yet in the local repository and merges them into the local repository.

**git push**: Publishes updates from the local repository to the remote repository

# Git产品比较

## Github

- 最流行的Git产品
- 网页 + CLI + 桌面App
- 免费 + 企业（付费）
- 平台集成（e.g. JIRA, Jenkins, Trello等）
- Gist
- 静态页面（个人博客，产品展示）

## Gitlab

- 提供和Github几乎一模一样的功能
- 社区版 + 企业版，社区版提供更多免费功能
- 价格比Github便宜
- Gitlab可以装在任何一个地方（云平台，虚拟机，各种系统）

## Bitbucket

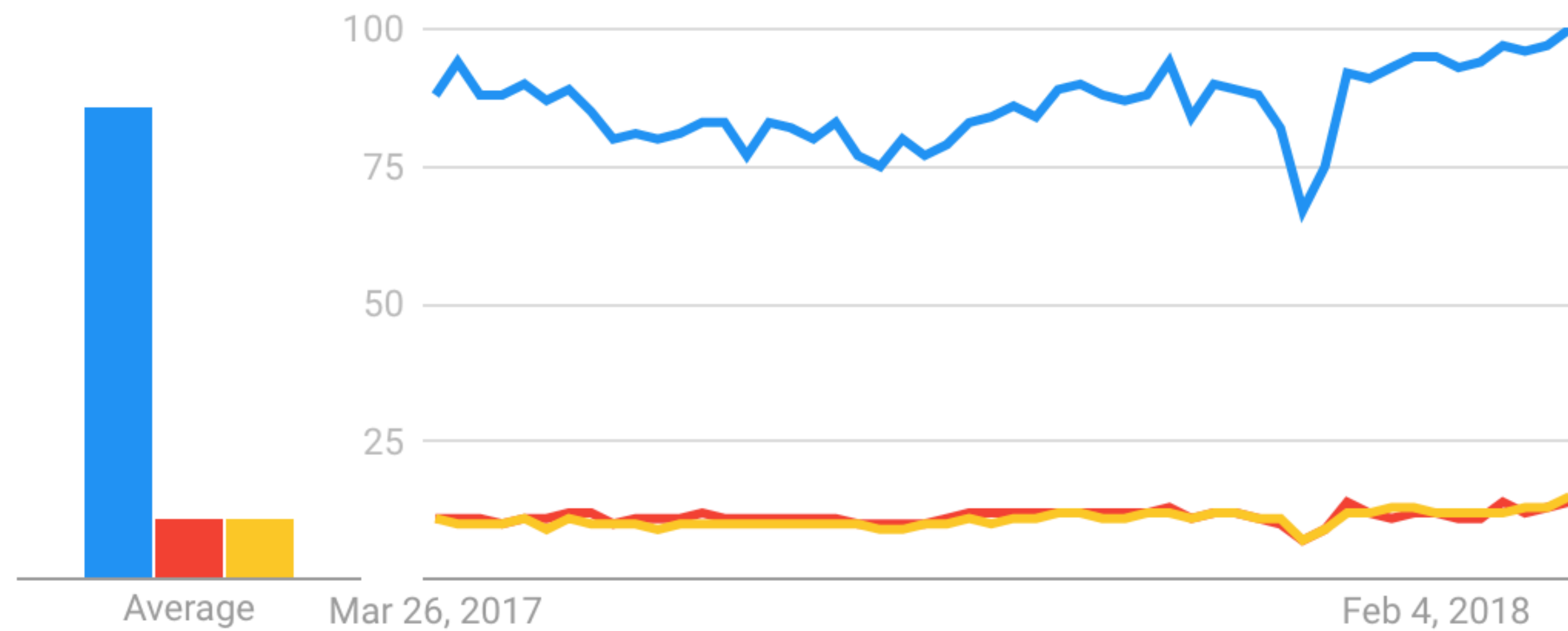
- Atlassian产品之一
- 支持Git和Mercurial
- 免费 + 商业版（免费可以拥有private repo）
- Gitlab可以装在任何一个地方（云平台，虚拟机，各种系统）

# Git产品比较

Interest over time

Google Trends

● GitHub ● Bitbucket ● Gitlab



Worldwide. Past 12 months. Web Search.



# Git/Github

## 安装与配置

2



# Mac Git安装和配置

## 1. 安装

手动 <https://git-scm.com/downloads>

自动 `brew install git`

## GUI工具

<https://git-scm.com/download/gui/mac>

<https://desktop.github.com/>

## 2. 检测安装

`git version`



# Mac Git安装和配置

## 3. git设置

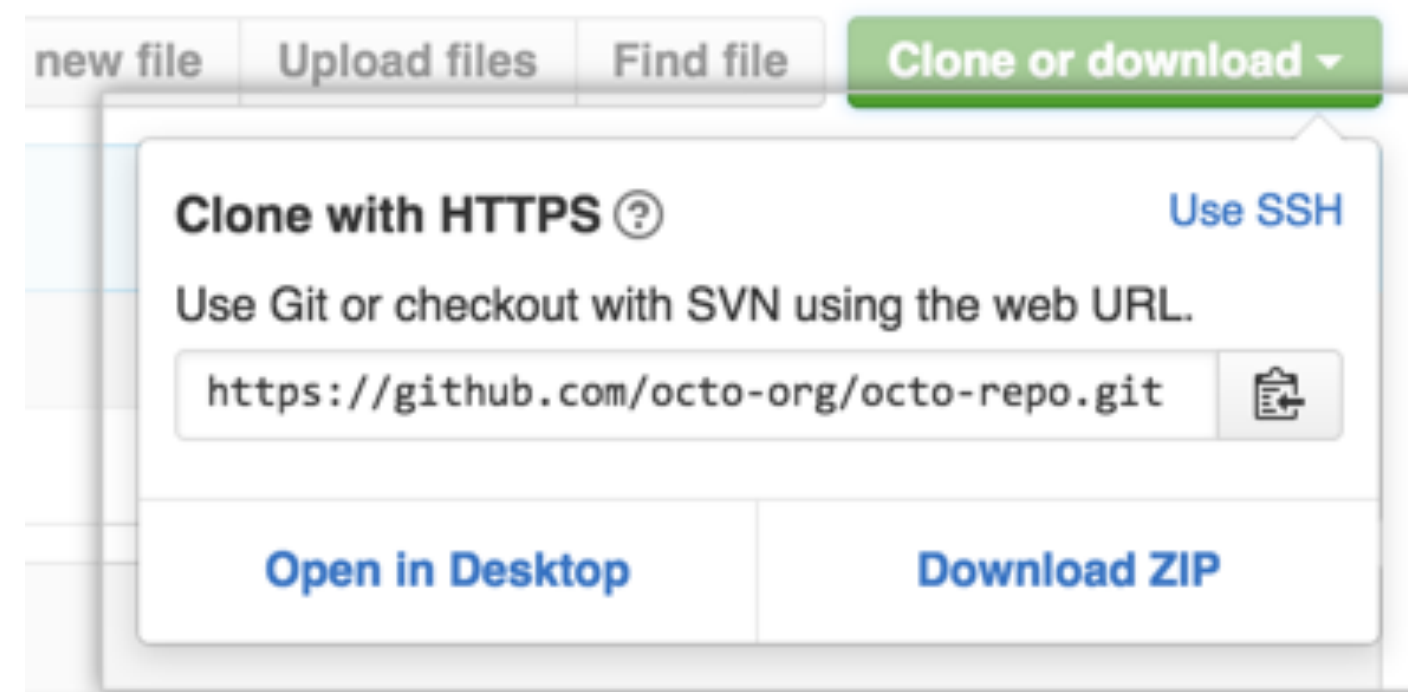
`git config --global user.name "commit-name"`

`git config --global user.email "commit-email@example.com"`

将邮件添加到自己github账户下 <https://github.com/settings/emails>

## 4. Clone选项

官方推荐HTTPS



# 第一个Github项目

1. 注册 <https://github.com/>
2. 创建项目（代码仓库） <https://github.com/new>

## 3. 本地创建项目

```
touch README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin <repo_url>  
git push -u origin master
```

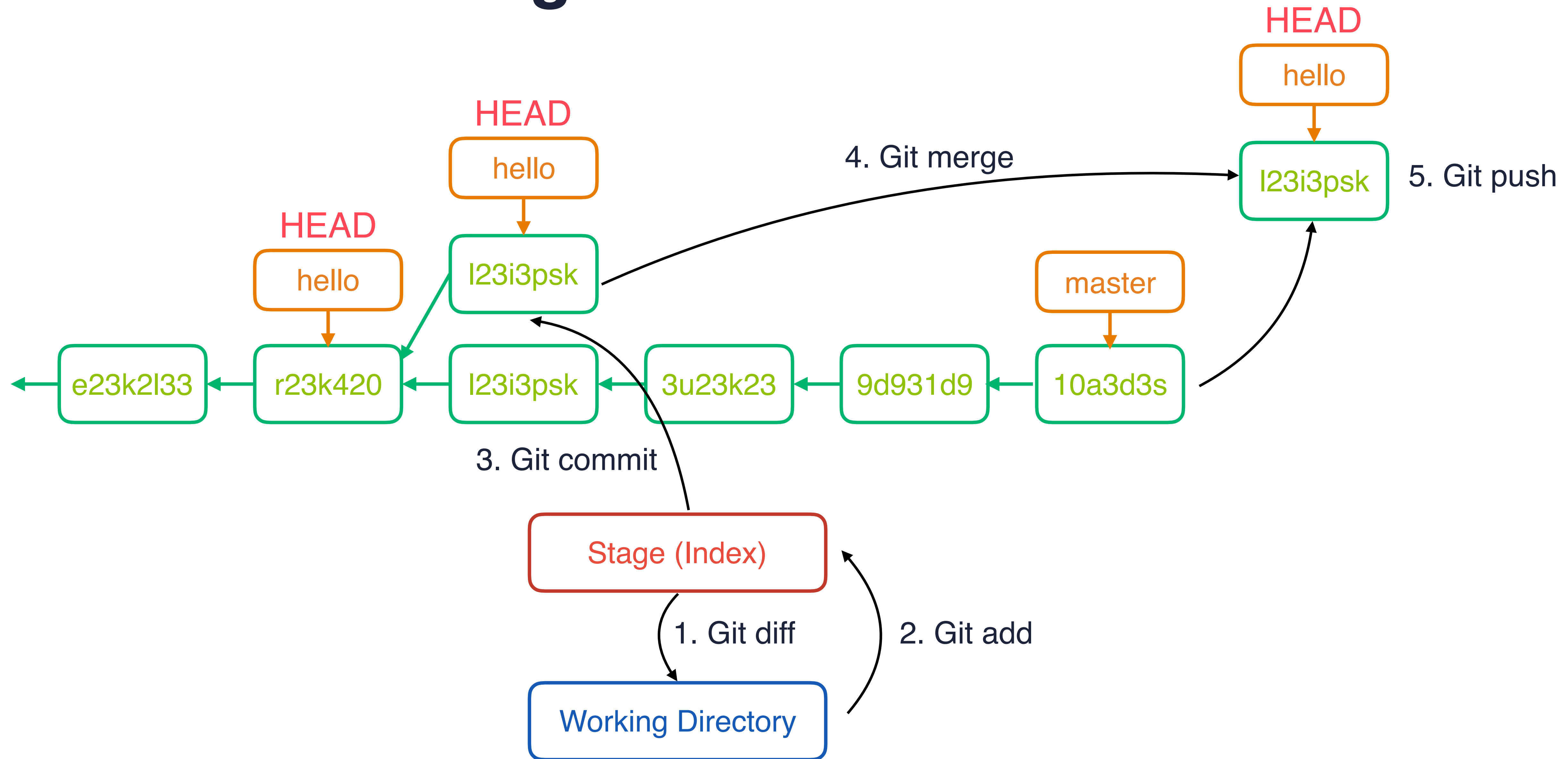
我们的账号 <https://github.com/impredator>



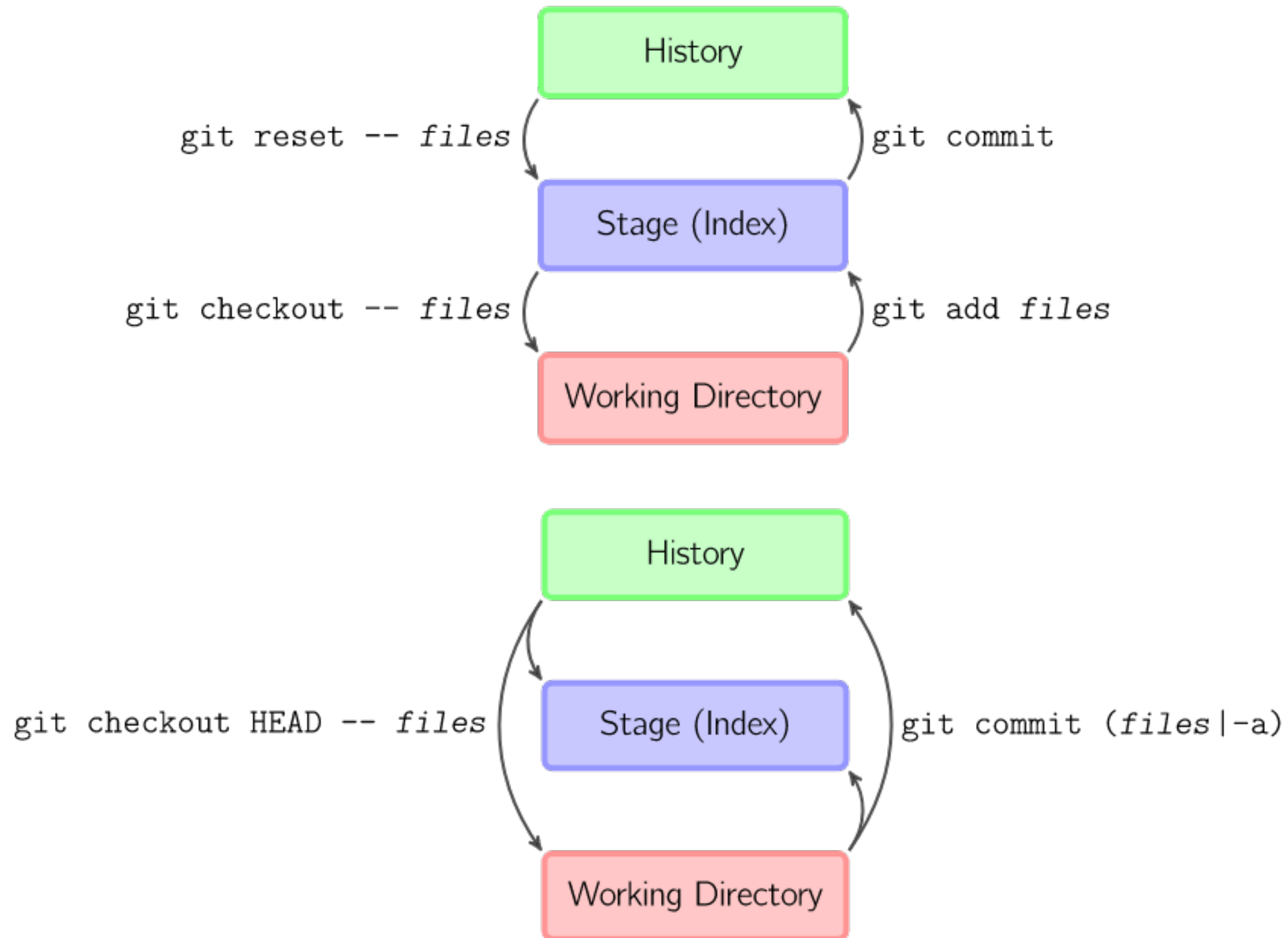
# Git 命令



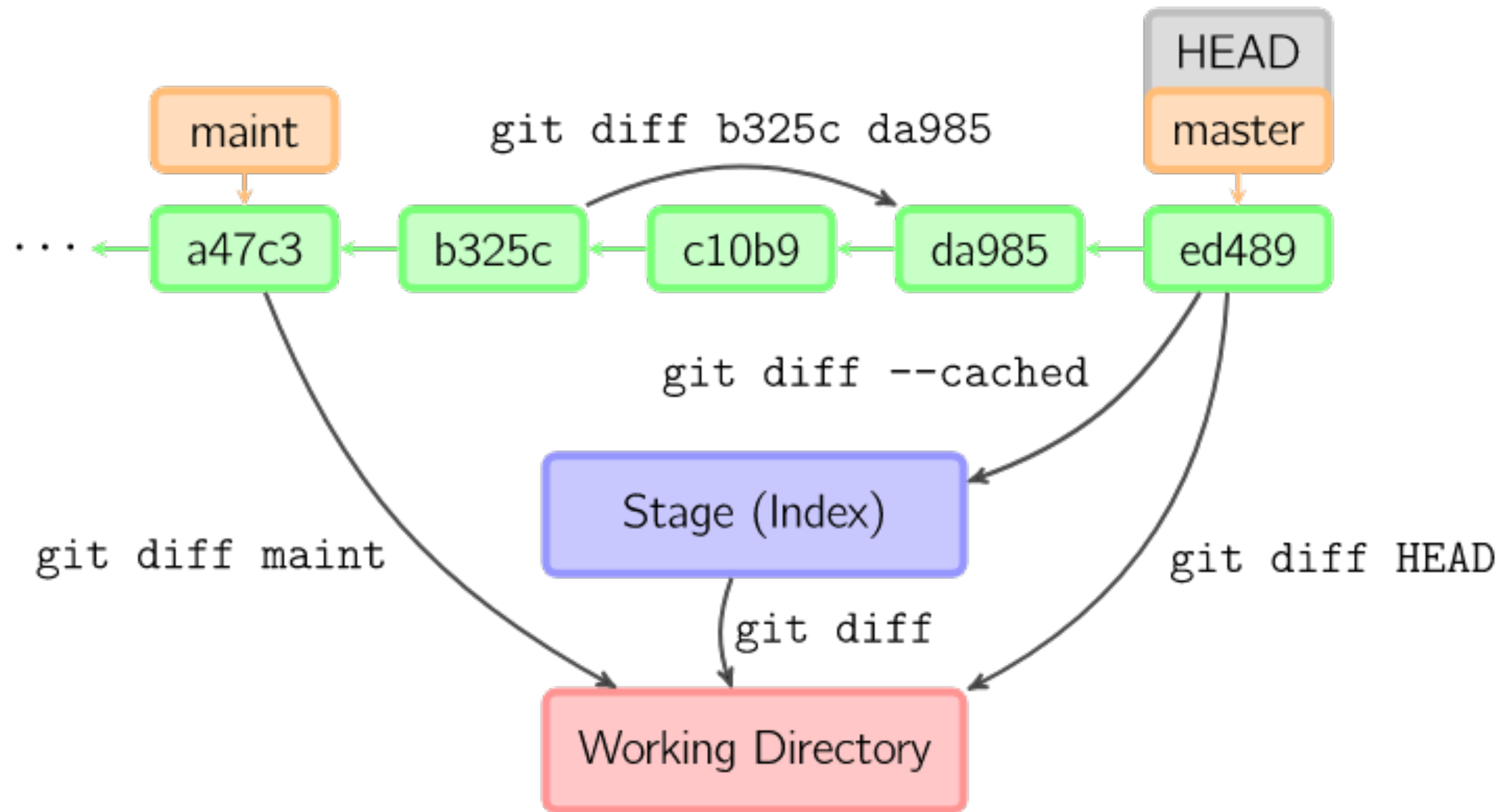
# git workflow



# git add

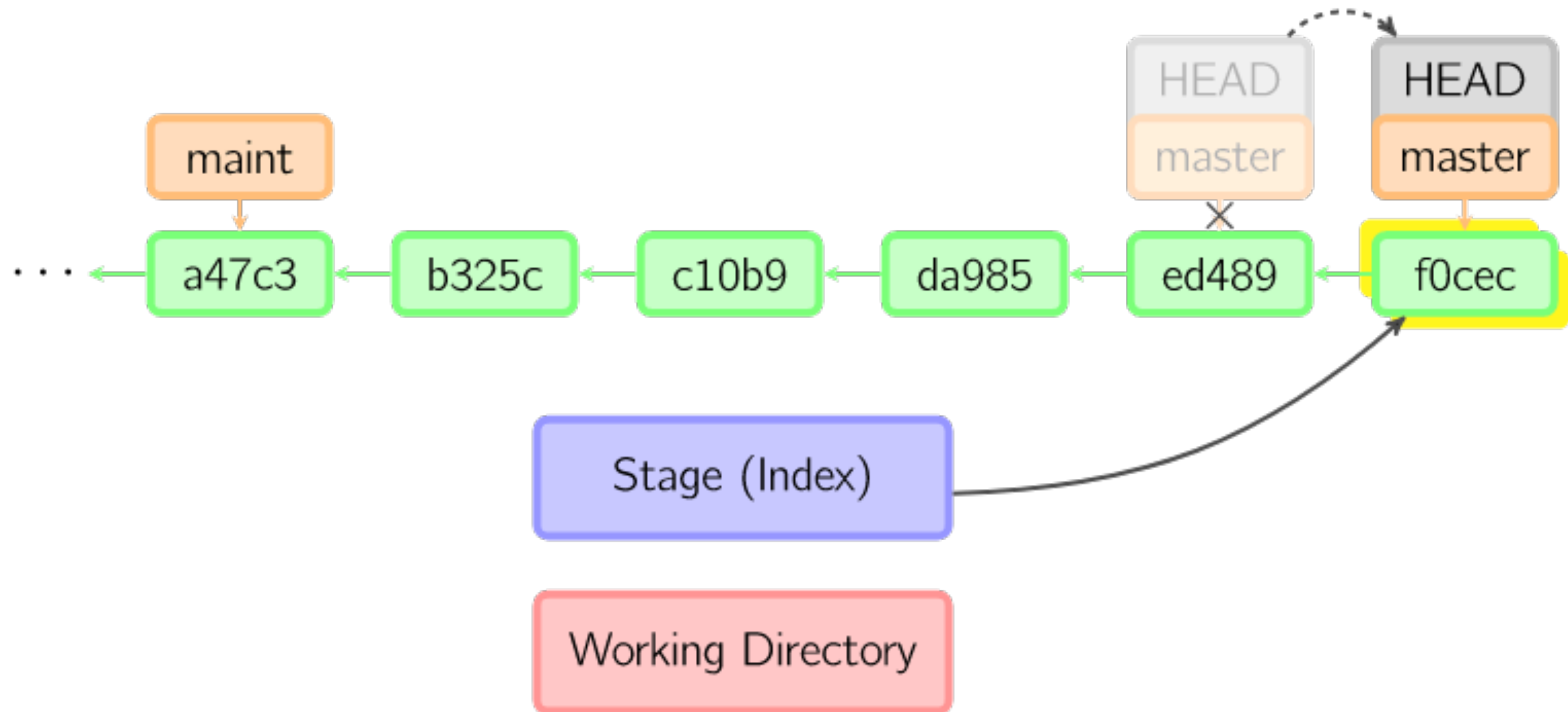


# git diff



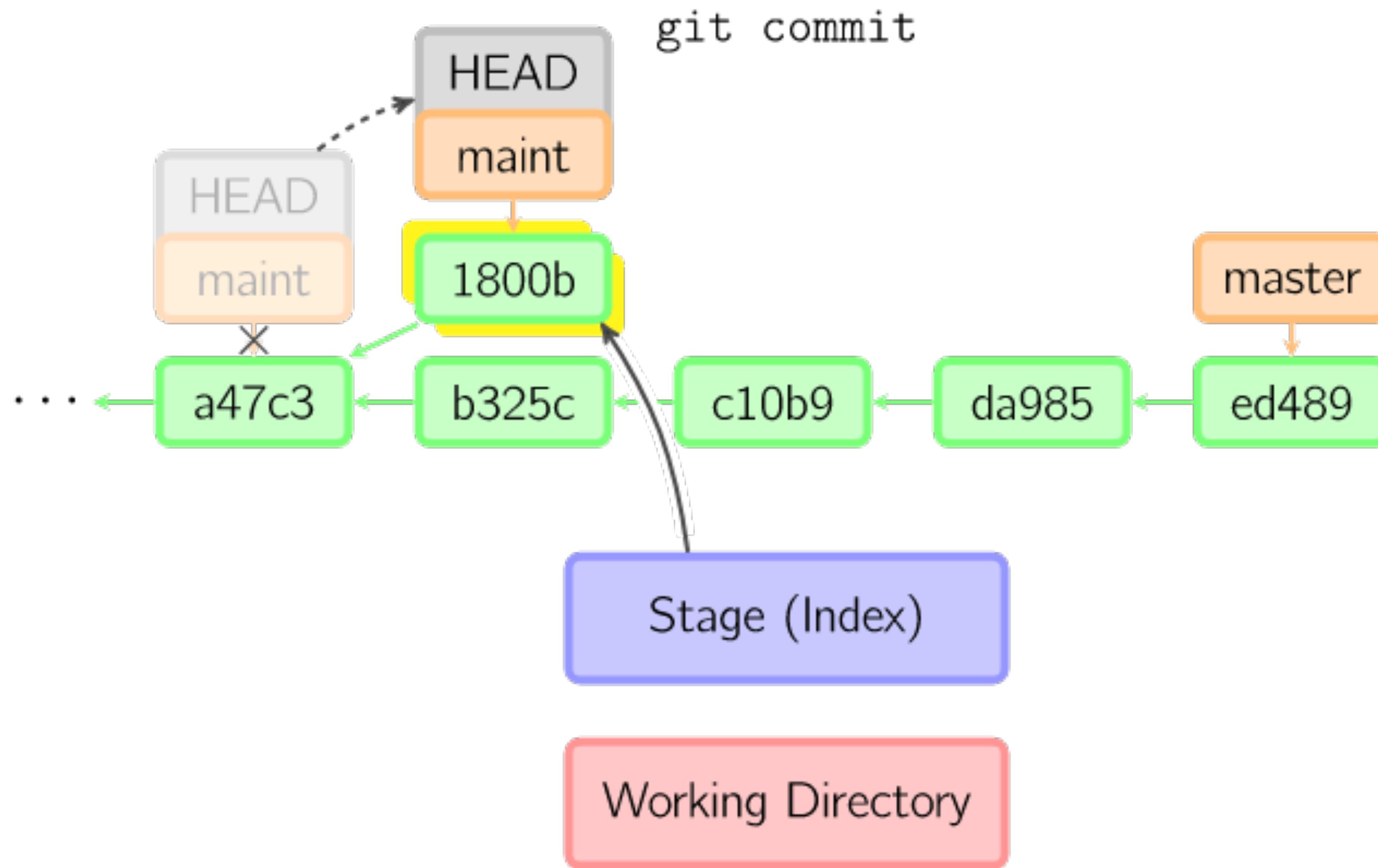
# git commit

git commit



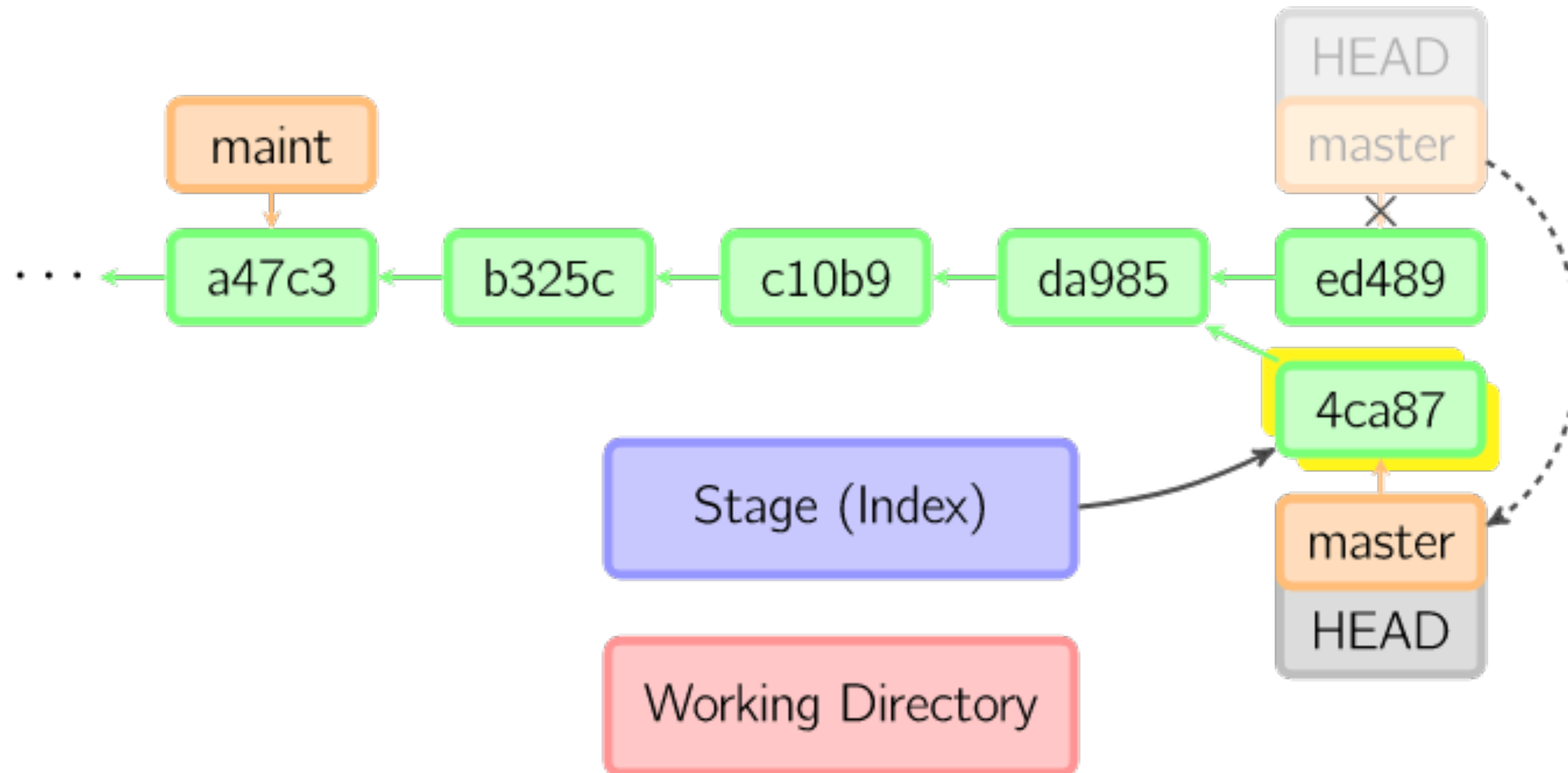


# git commit



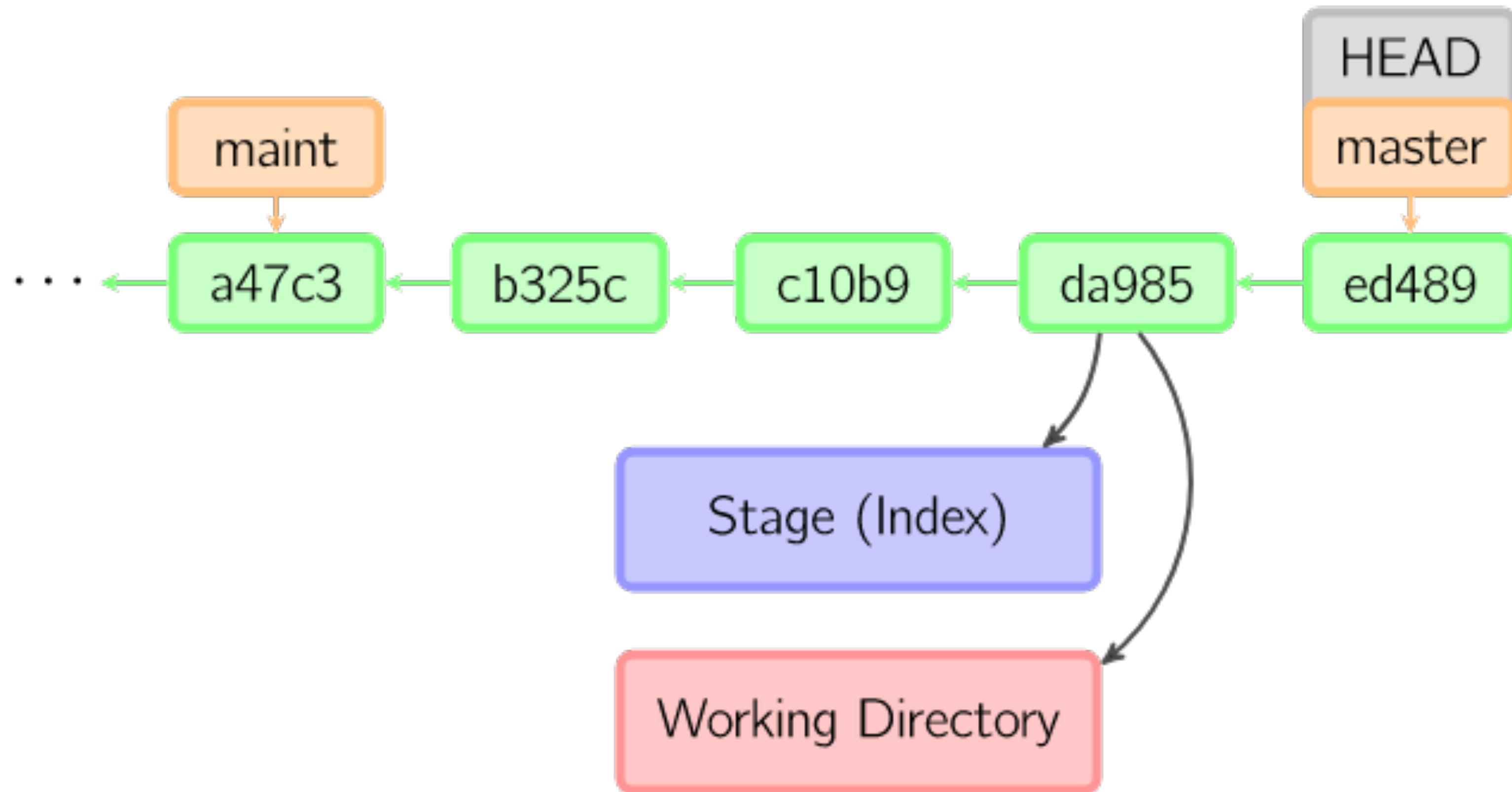
# git commit

```
git commit --amend
```



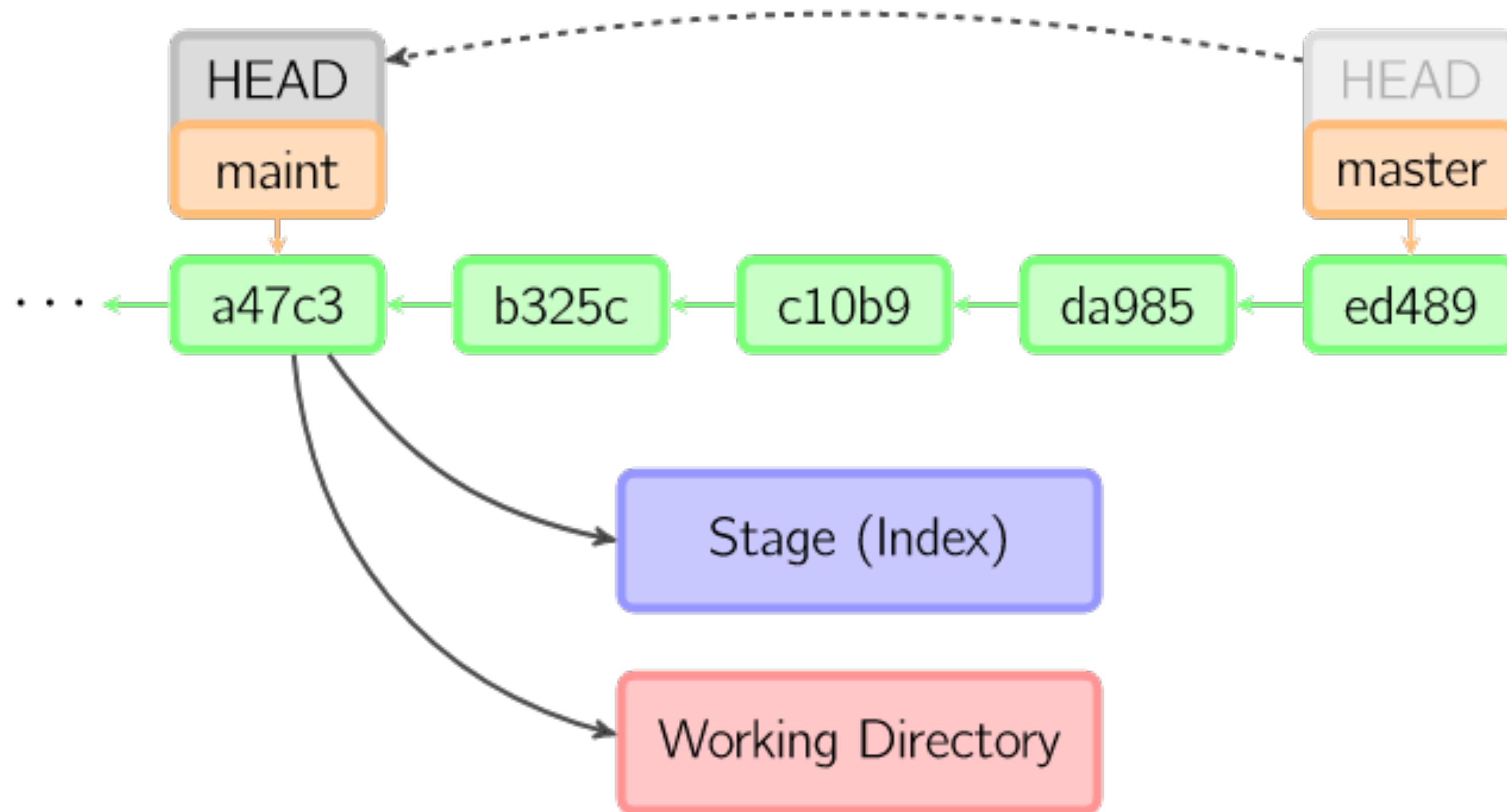
# git checkout

`git checkout HEAD~ files`



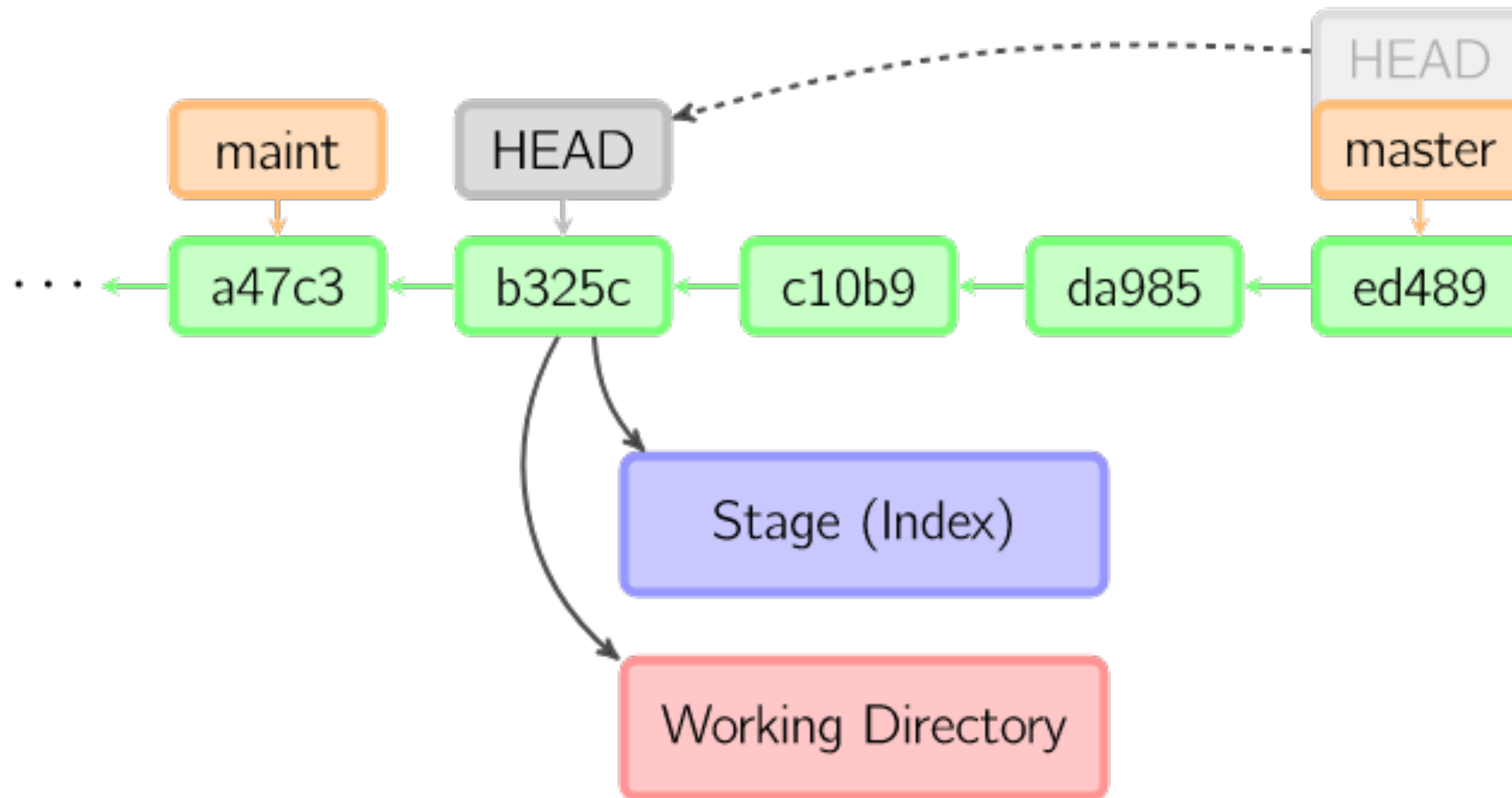
# git checkout

git checkout maint



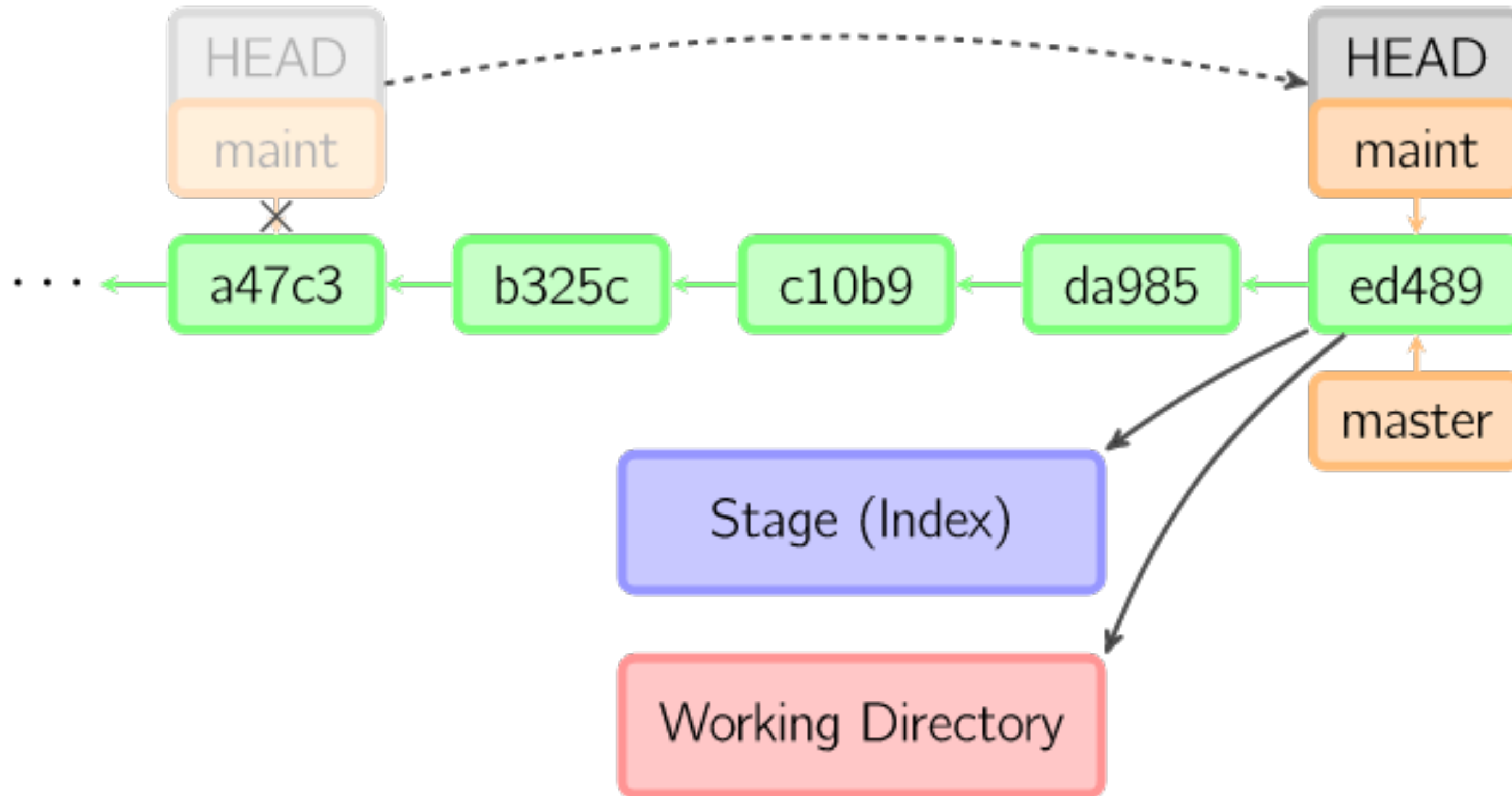
# git checkout

```
git checkout master~3
```

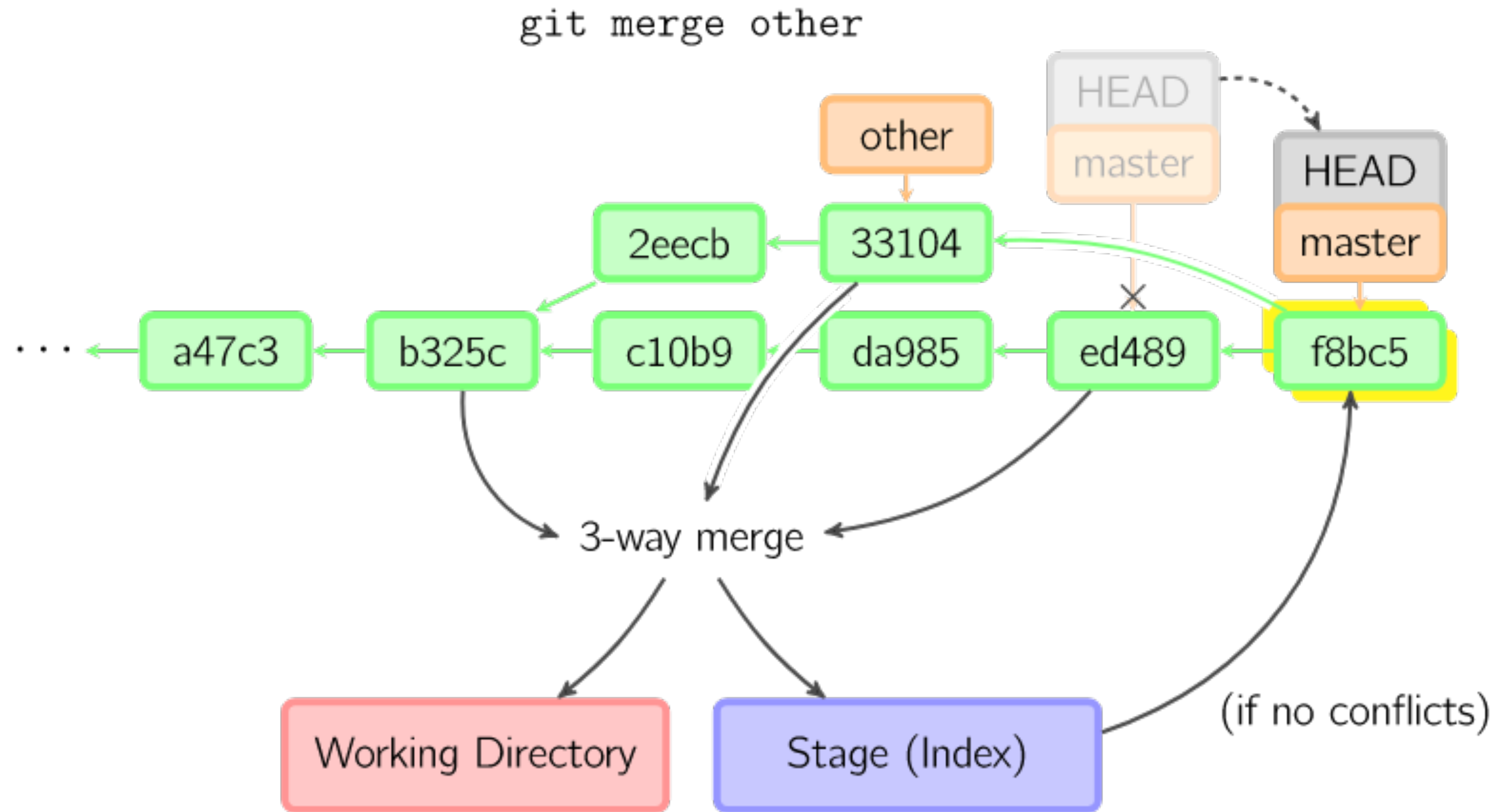


# git merge

`git merge master`



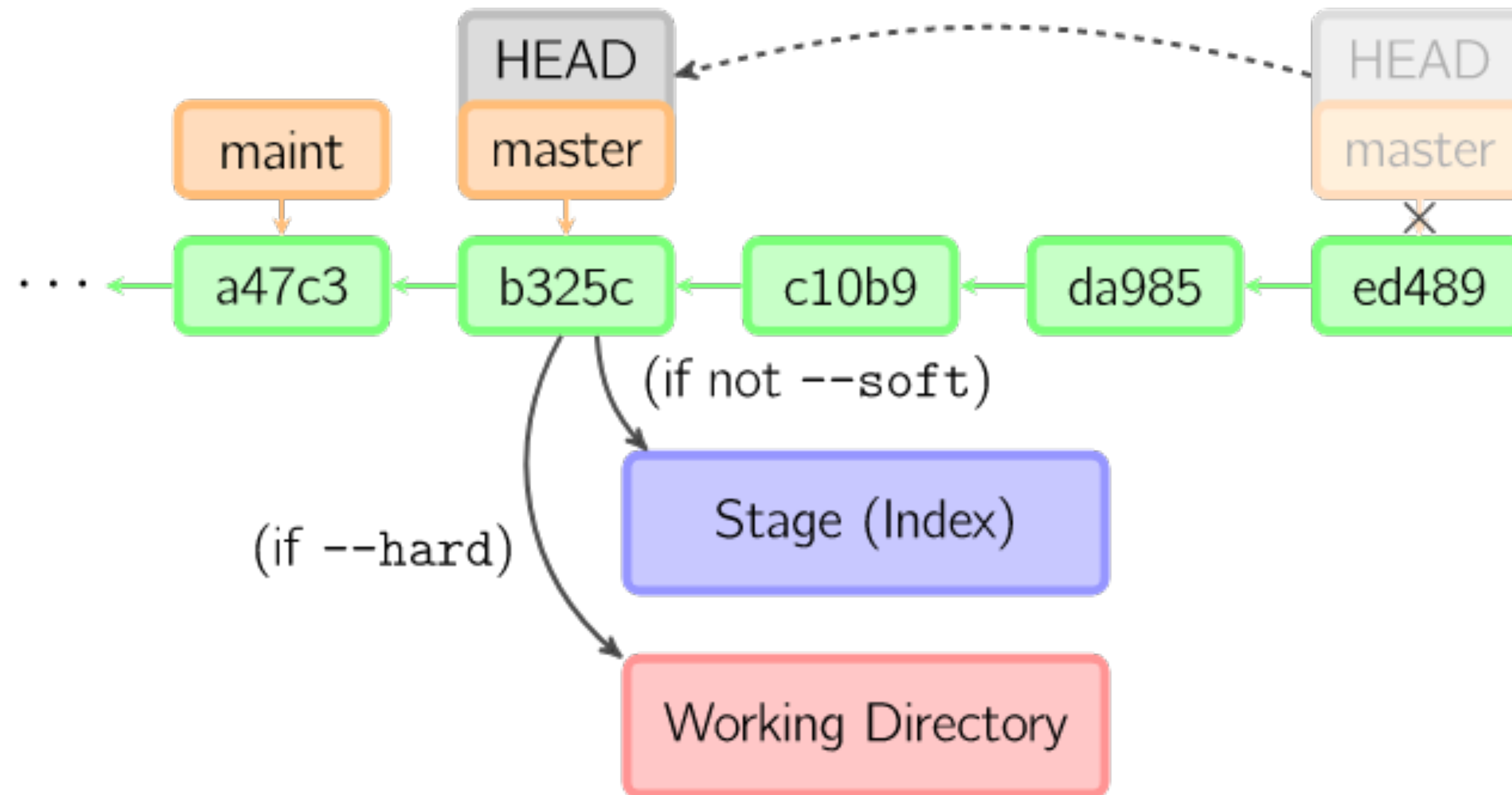
# git merge





# git reset

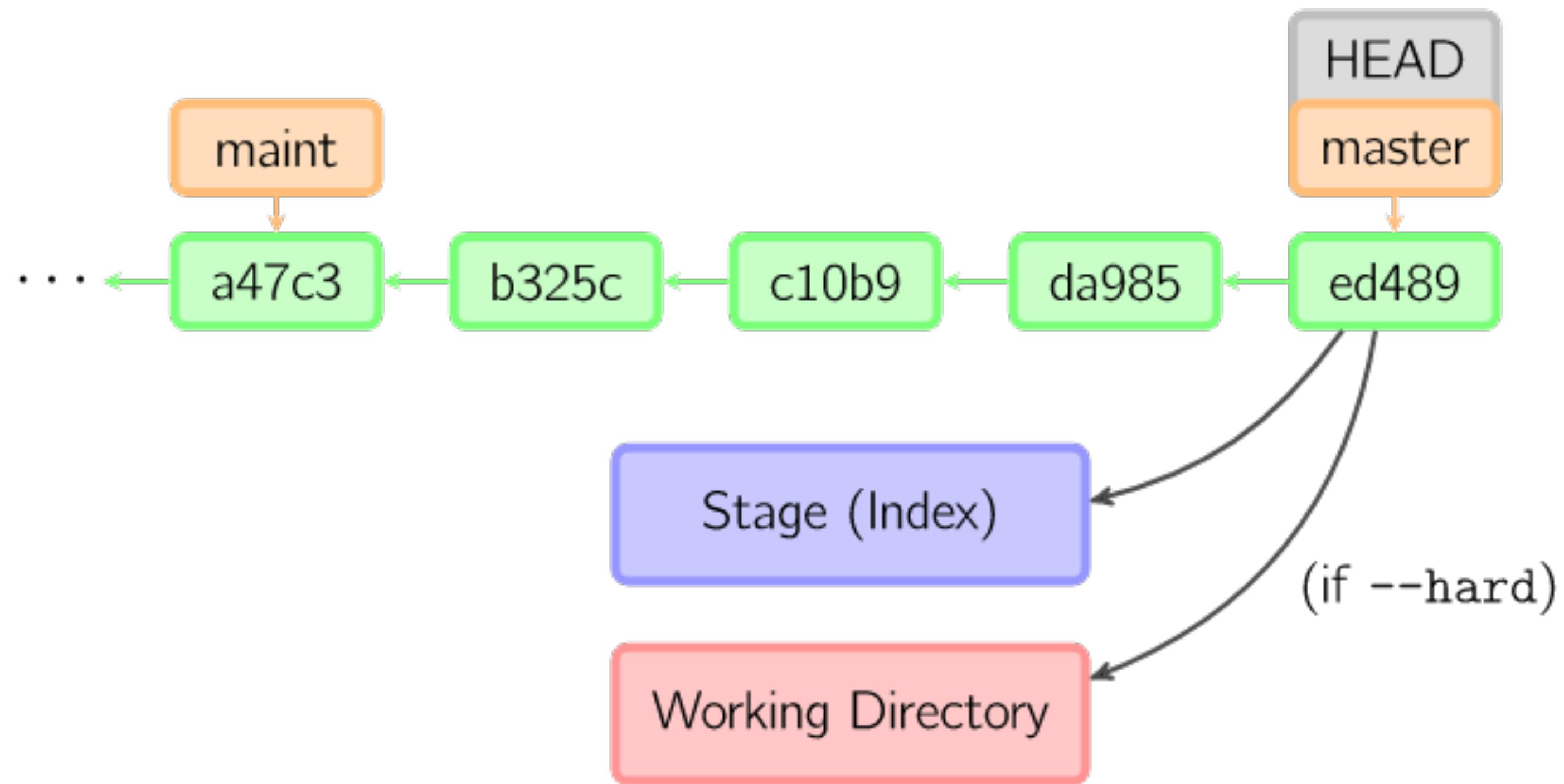
```
git reset HEAD~3
```





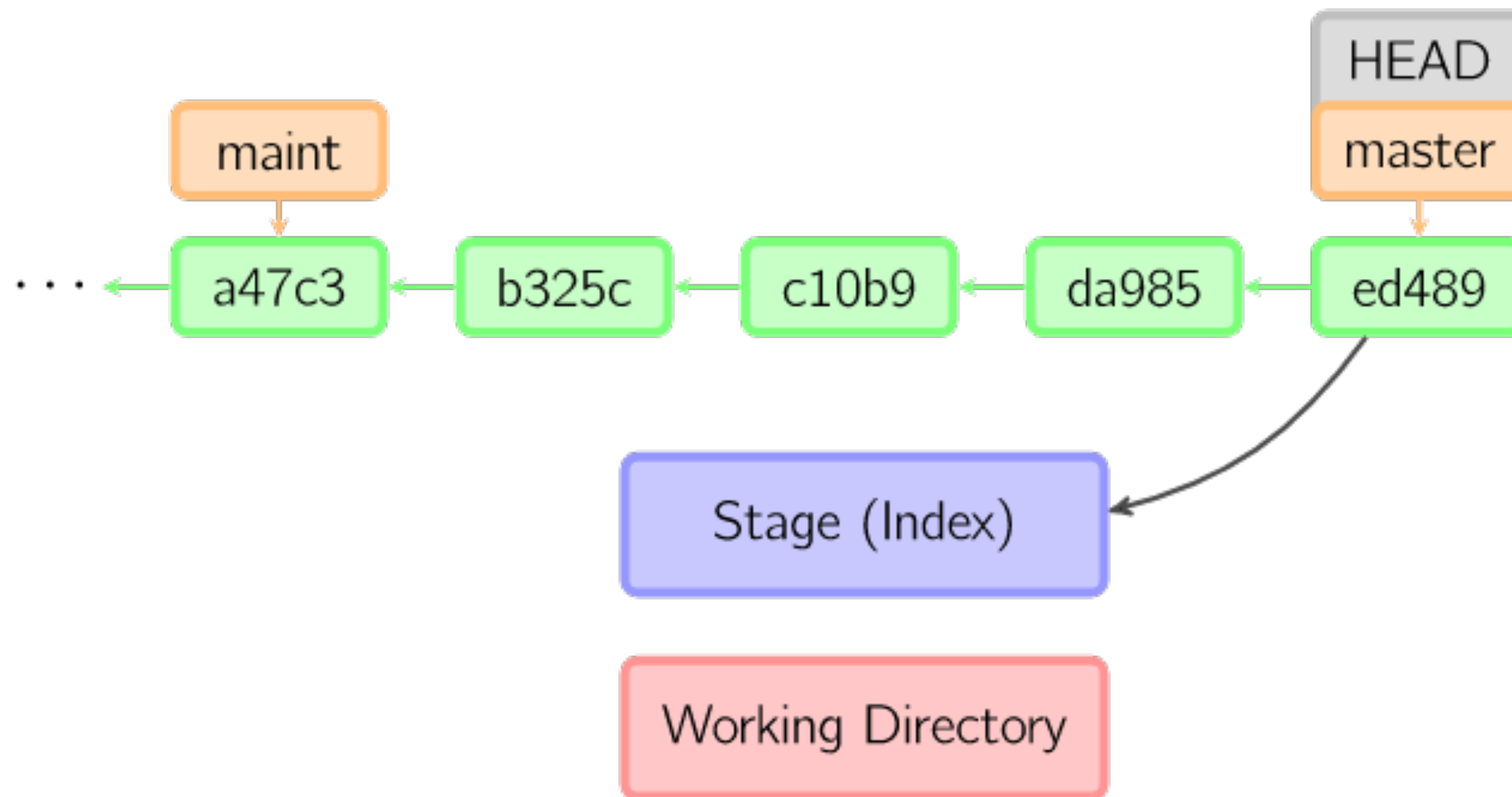
# git reset

git reset

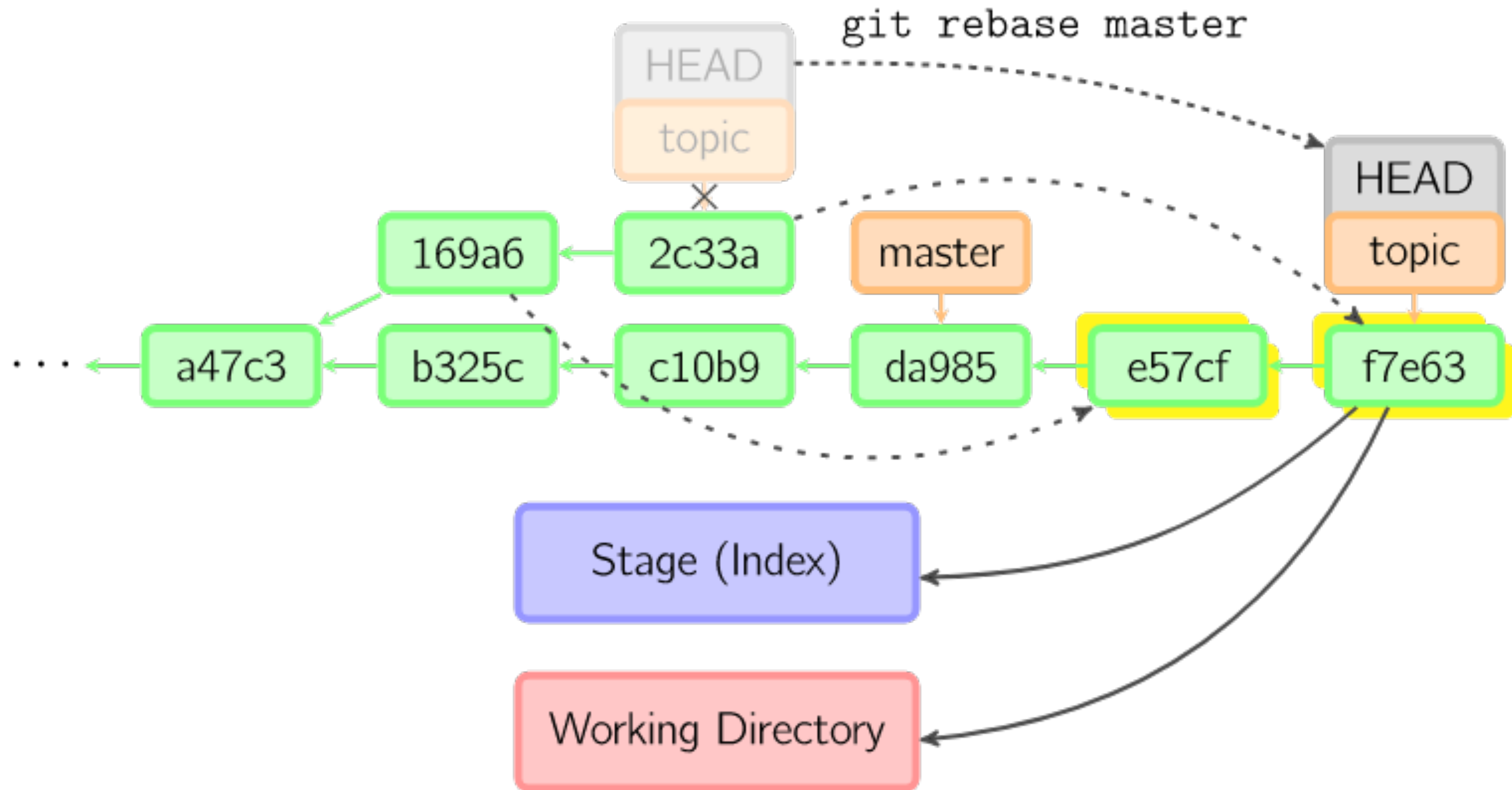


# git reset

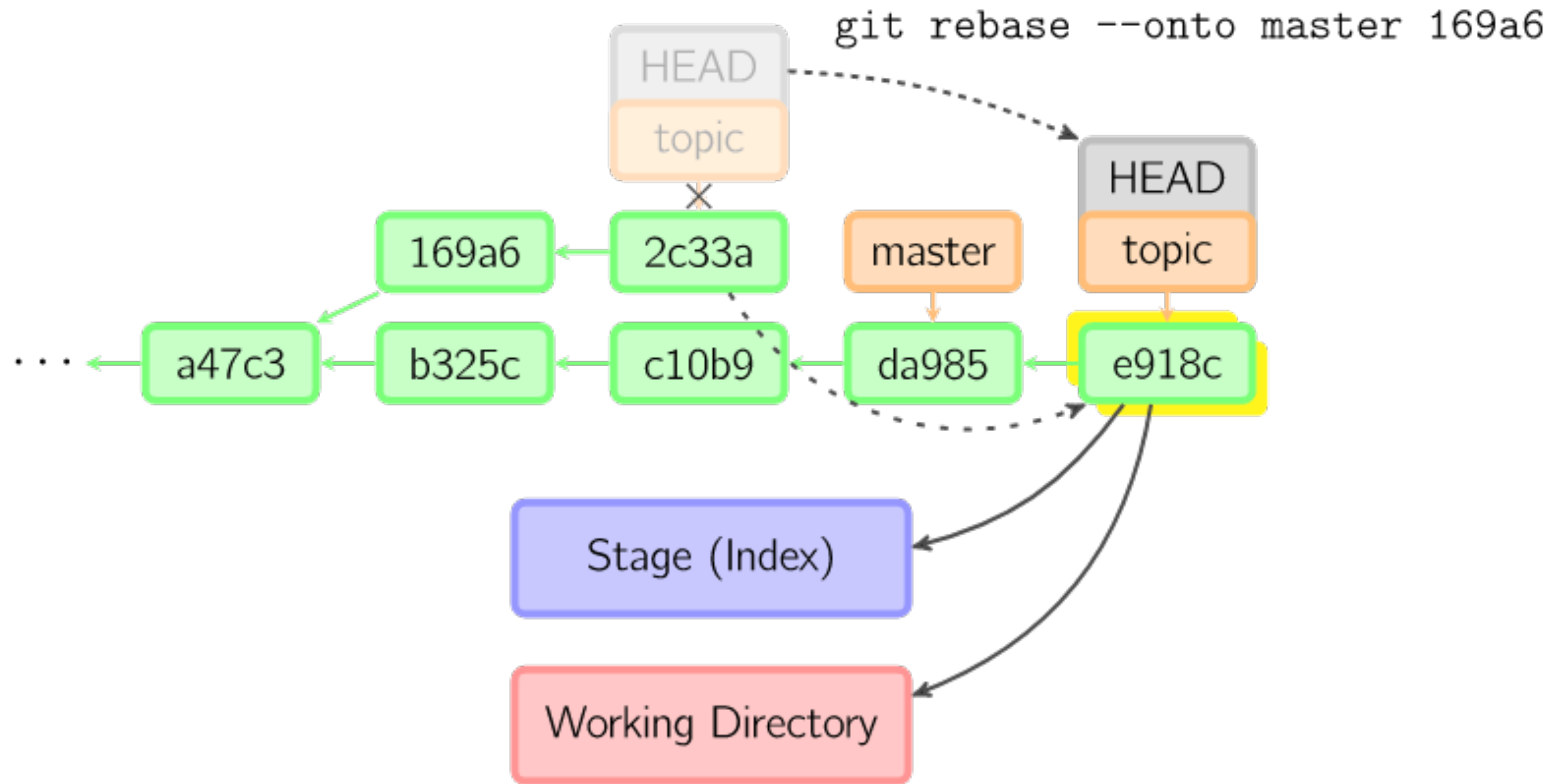
```
git reset -- files
```



# git rebase

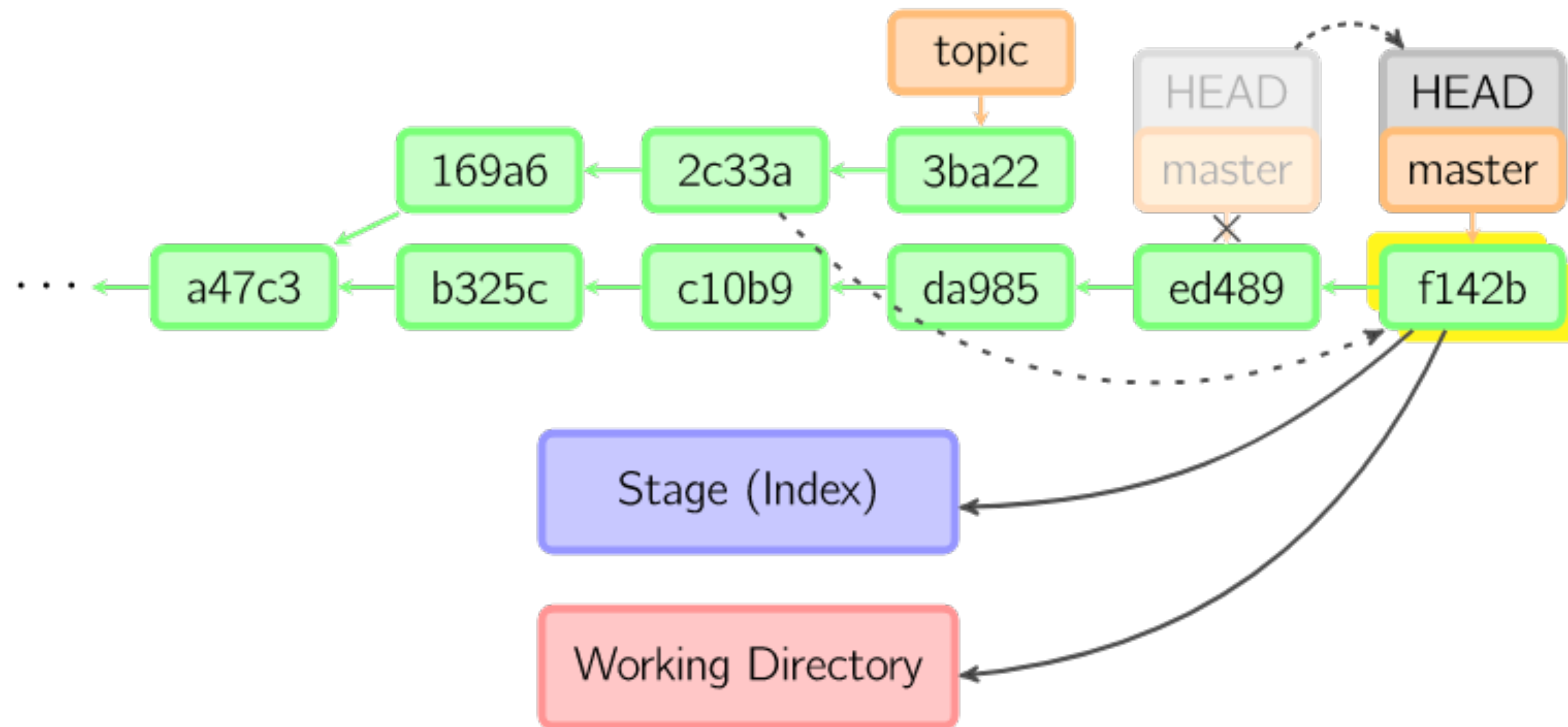


# git rebase



# git cherry-pick

git cherry-pick 2c33a

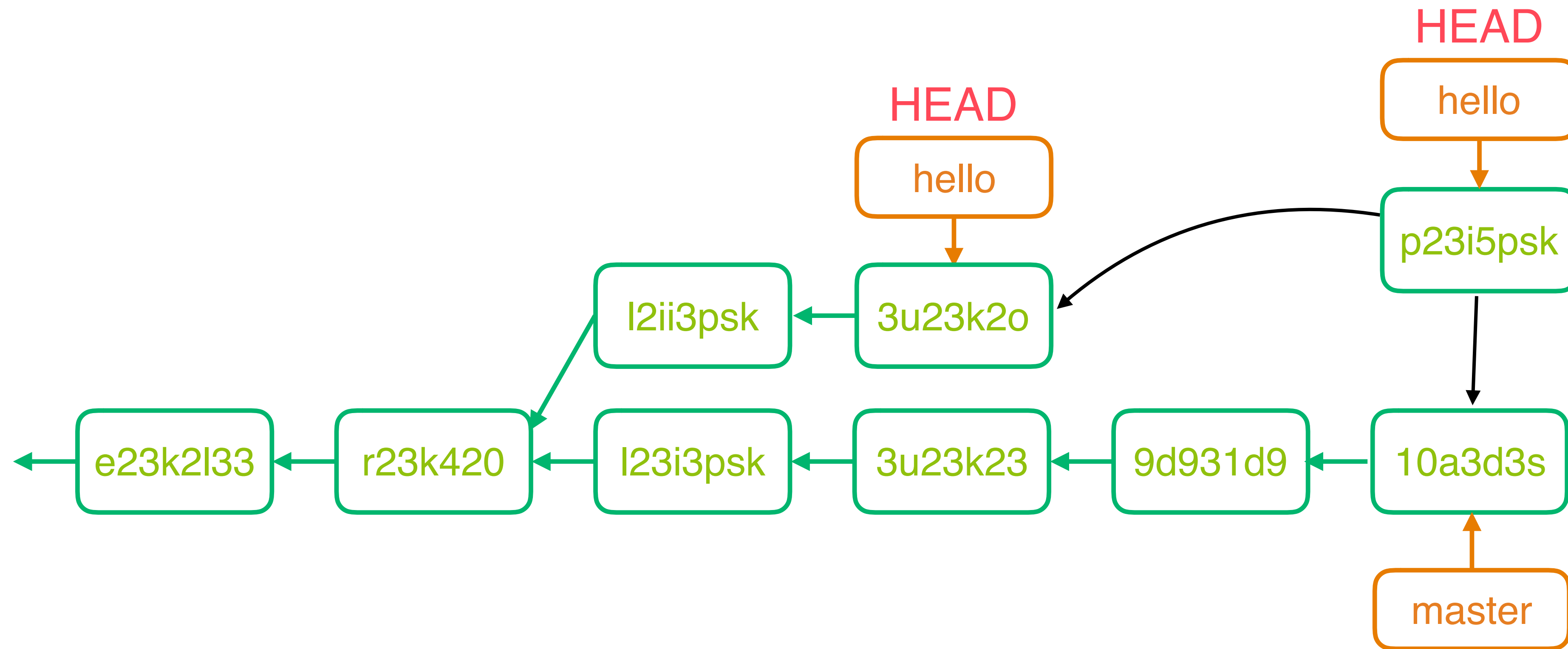




# Git高级应用 与最佳实践

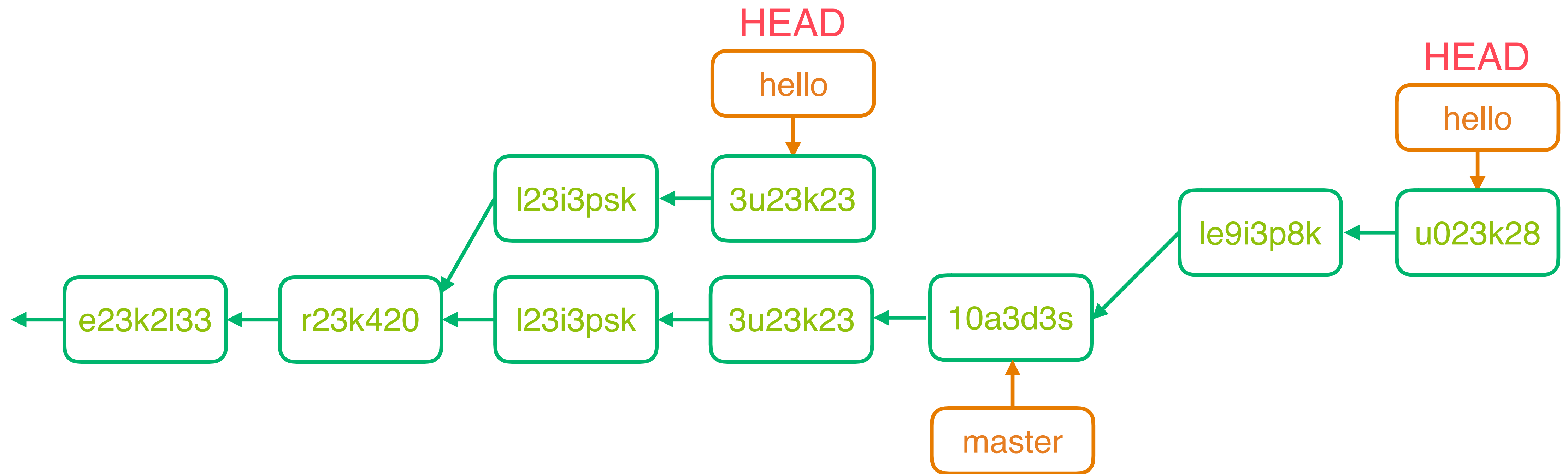
3

# Merge vs Rebase



git merge master

# Merge vs Rebase



git rebase master



# Merge vs Rebase

解决同样的问题，只是方法不同

额外创建一个提交

保留分支历史，看到分支结构

非破坏性操作，避免风险

随着项目增加，不便于理解

不会多生成一个提交

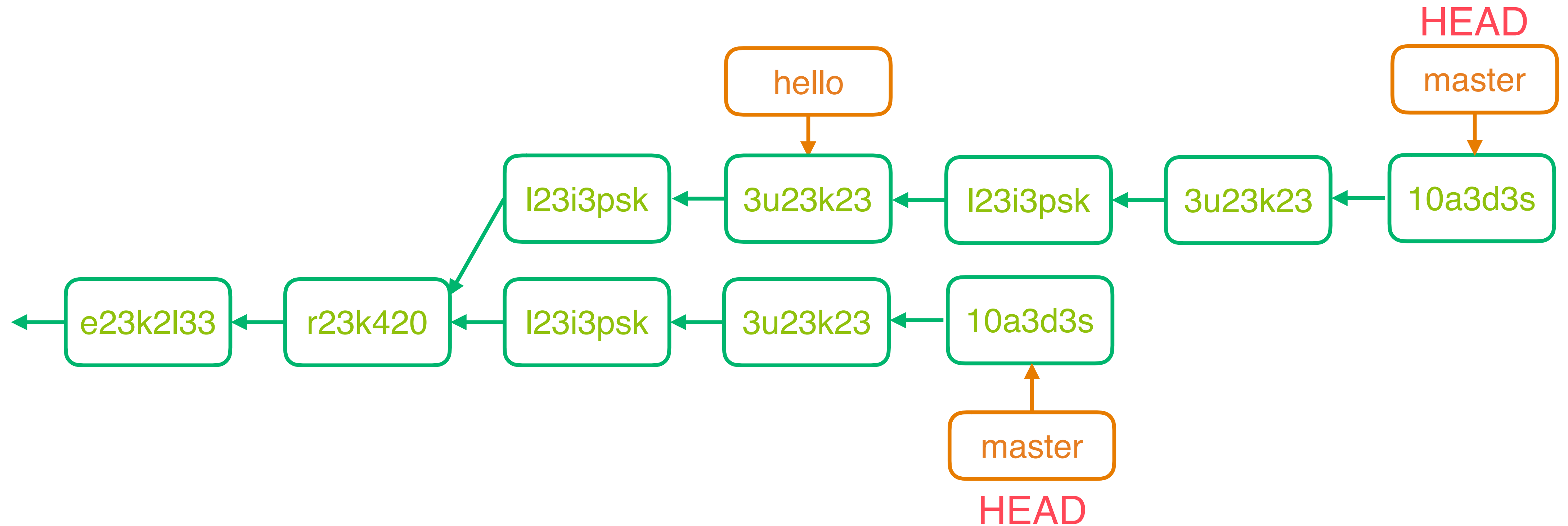
线性化合并分支，破坏分支历史

破坏性操作，存在潜在风险

需要遵守**黄金法则**



# Diverged Rebase



“

“不要在公共分支上使用  
REBASE”



# 最佳实践之 分支管理

1. 创建一个新分支
2. 每个组（个人）工作在自己的分支上
3. 小步提交
4. 经常同步主分支
5. 重复3，4直到完成该功能

```
git checkout -b 'branch-name'
```

```
git add .
```

```
git commit -m 'desc'
```

```
git pull origin master --rebase
```

```
git push origin 'branch-name'
```

# Reset vs Revert vs Checkout

命令	作用域	使用场景
git reset	Commit-level	私有分枝上放弃提交（commit），扔掉未提交（commit）的改变
git reset	File-level	删除暂存区（stage）
git checkout	Commit-level	切换分支，查看历史提交
git checkout	File-level	删除工作区（working place）
git revert	Commit-level	放弃公共分之上提交（commit）
git revert	File-level	(N/A)

# 提交级别之 Reset

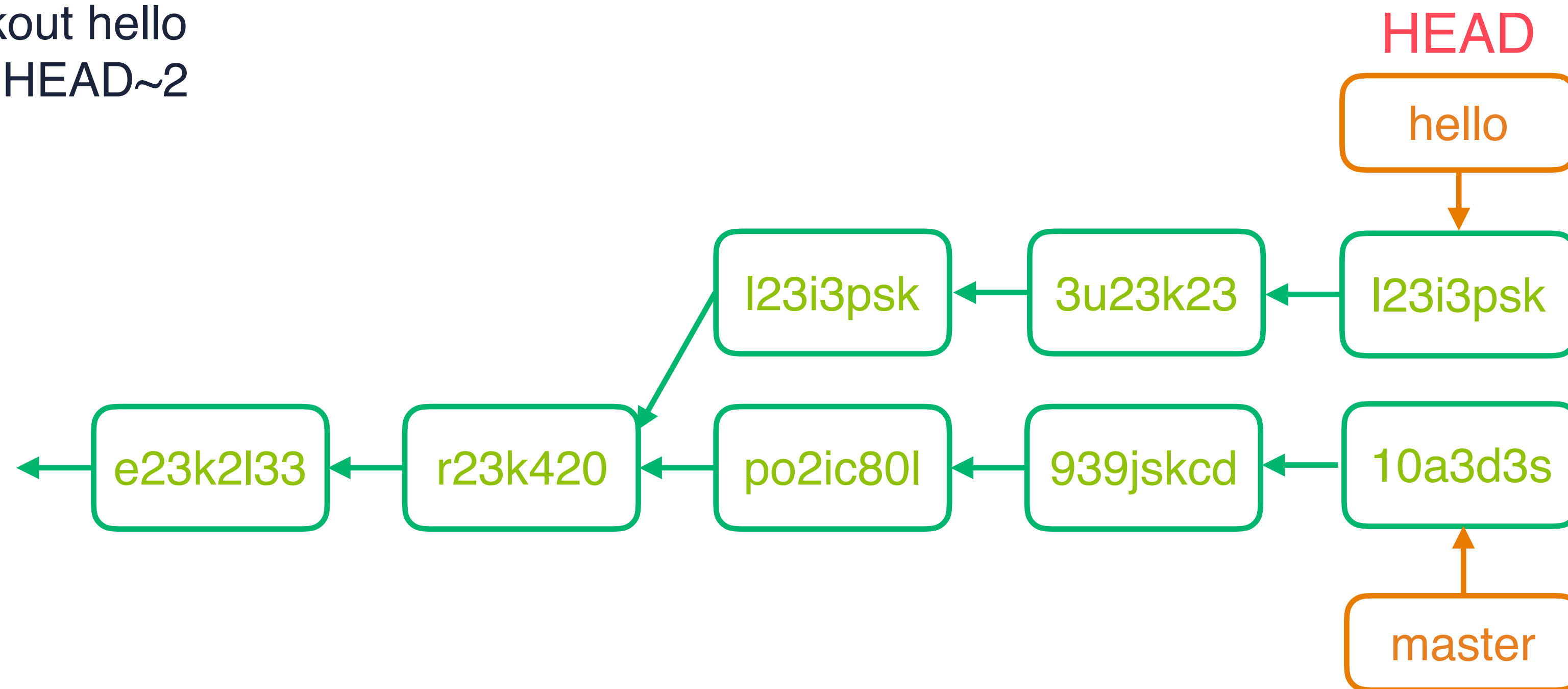
用于在当前分支上删除提交

悬挂提交或孤儿提交会在垃圾回收阶段被GIT删除

—soft工作目录和阶段快照都不变； —mixed默认，工作目录不变，快照改变； —hard工作目录和快照都改变

```
git checkout hello
```

```
git reset HEAD~2
```



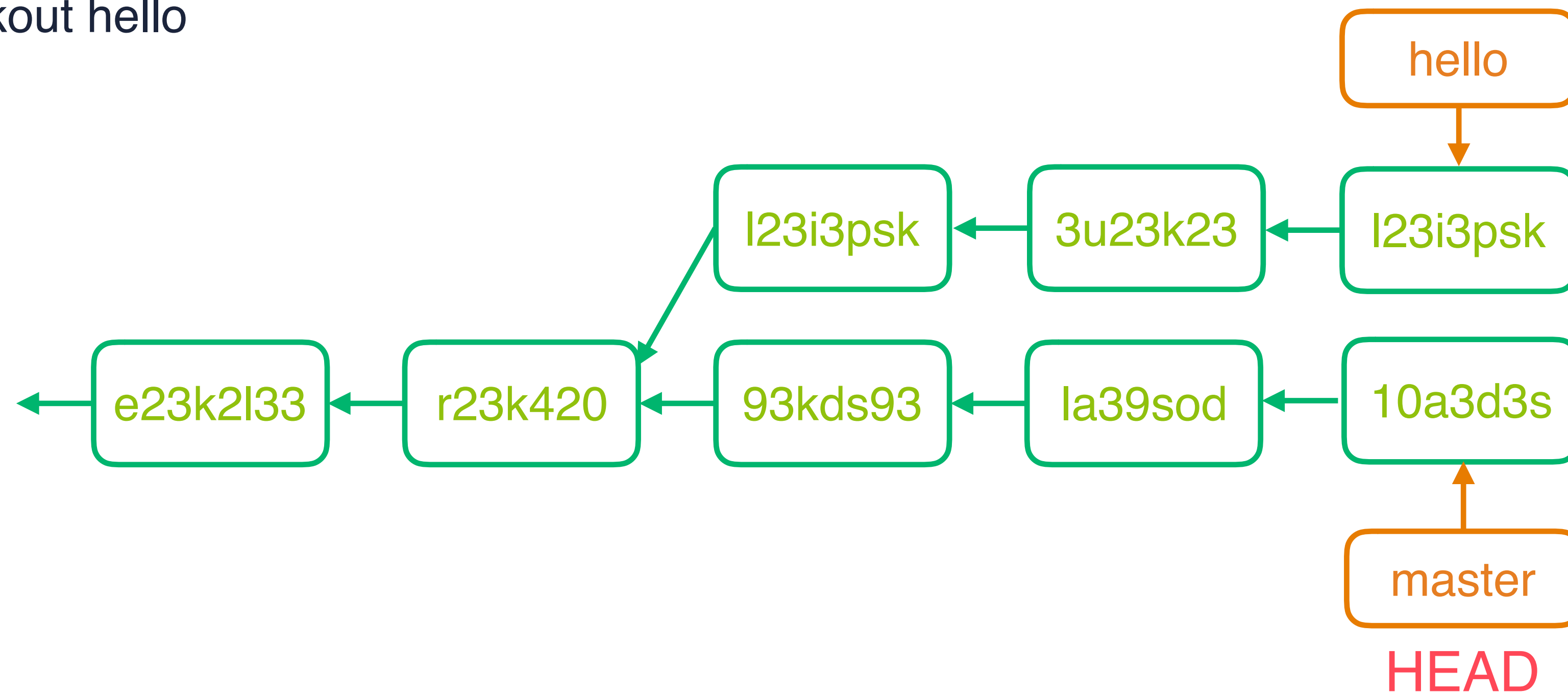
# 提交级别之 Checkout

抽象：用于更新代码仓库（本地）状态到指定点

通俗：切换分支-移动HEAD到指定分支，更新工作目录内容；

和Reset不同之处，Checkout时候Git会提示commit或者stash未提交内容，它不会移动分支

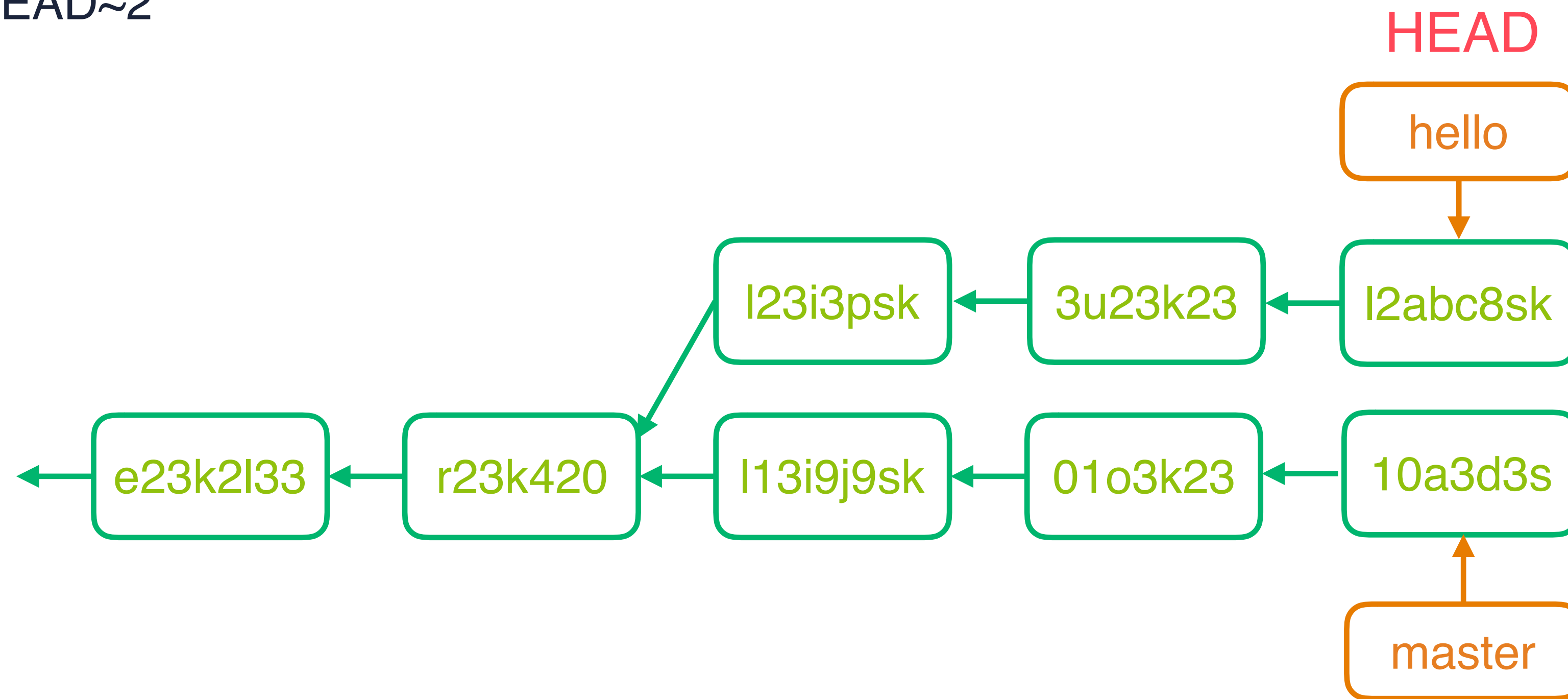
git checkout hello



# 提交级别之 Checkout

通过commit hash, 或者移动距离, 可以checkout到指定提交点, 用于查看历史代码提交内容  
HEAD处于分离状态, 如果这时做提交, 首先创建分支, 再提交; 否则, 提交后, 一旦切换回其它分支, 提交丢失

git checkout HEAD~2



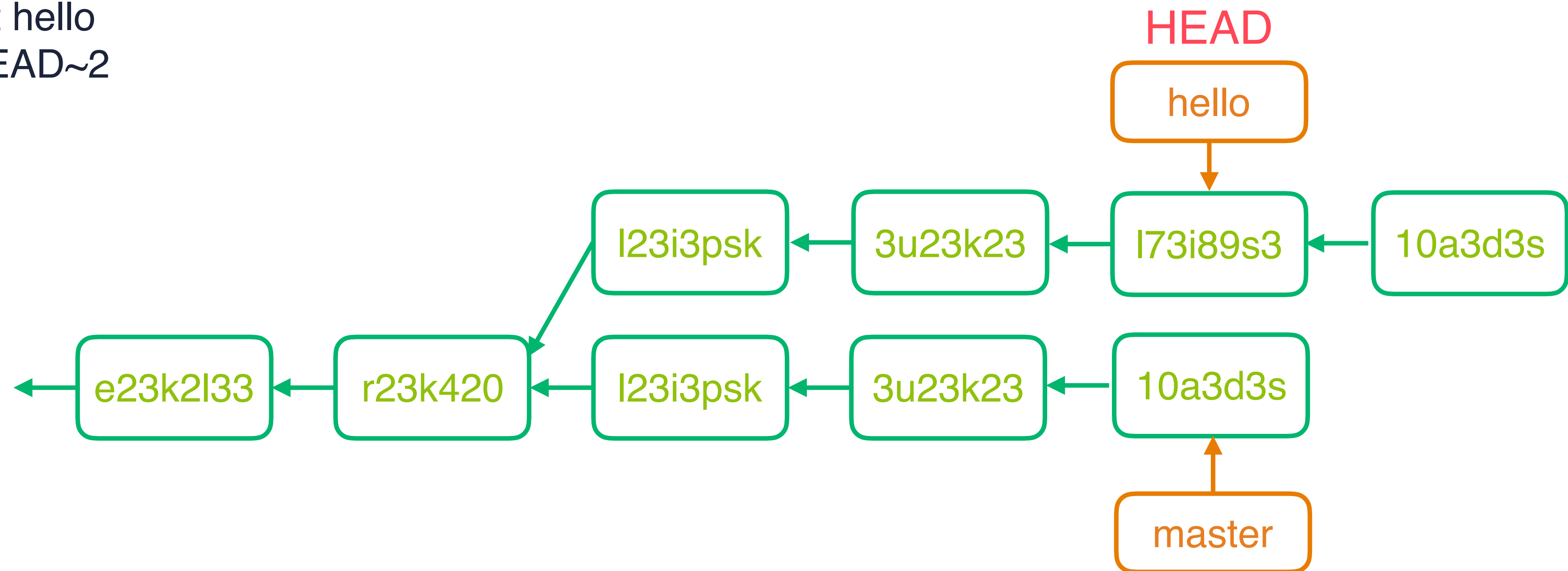


# 提交级别之 Revert

通过创建一个新提交来撤销之前的提交，这样它不会修改提交历史

相比reset, revert用于撤销公共分支提交(undo committed), reset用于撤销私有分支提交(undo uncommitted)  
和checkout一样，它会提示commit或stash未提交改变

```
git checkout hello  
git revert HEAD~2
```

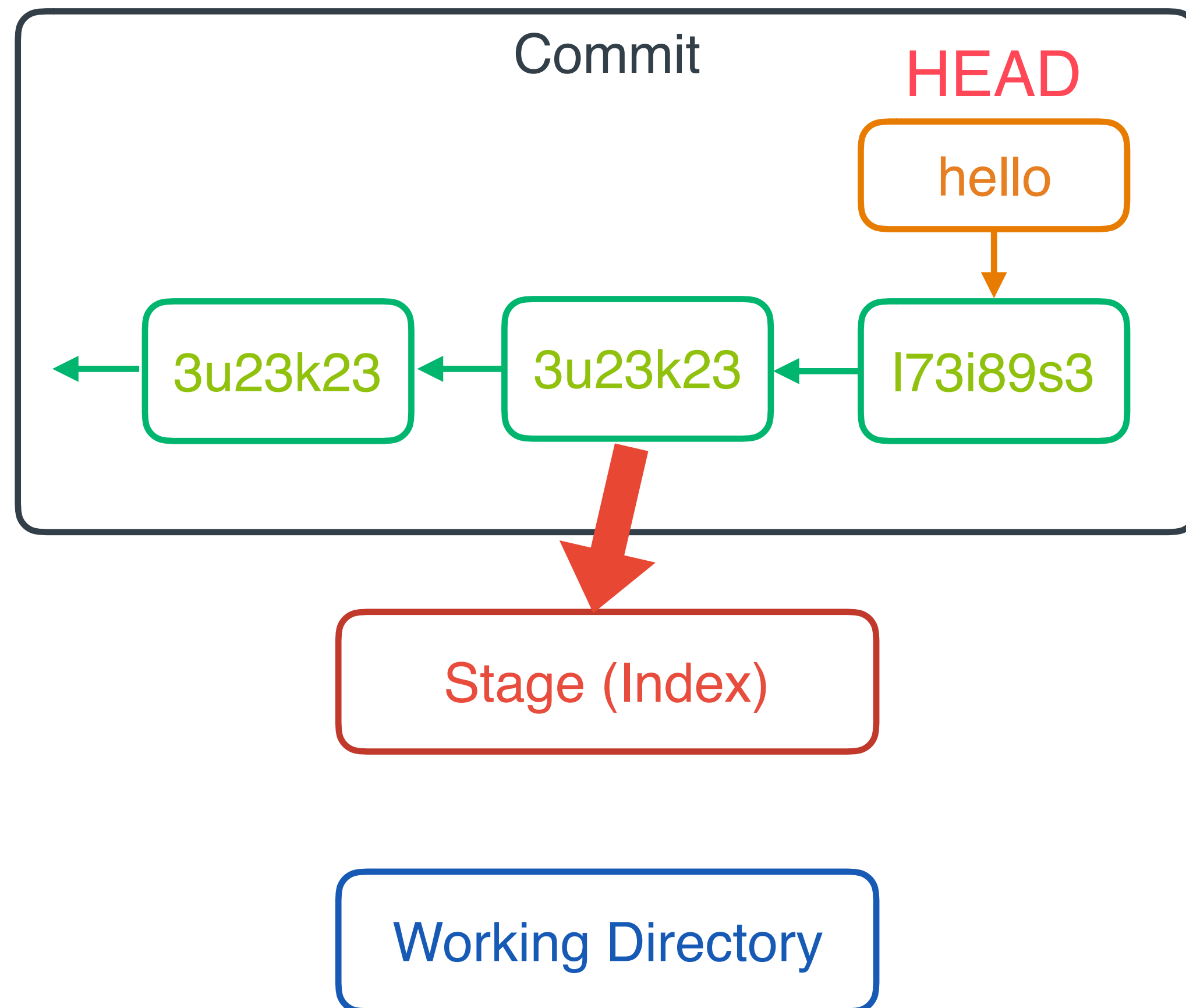


# 文件级别之 Reset vs Checkout

`git reset HEAD~2 dharma.java`

用指定提交替换阶段快照，工作目录保持不变

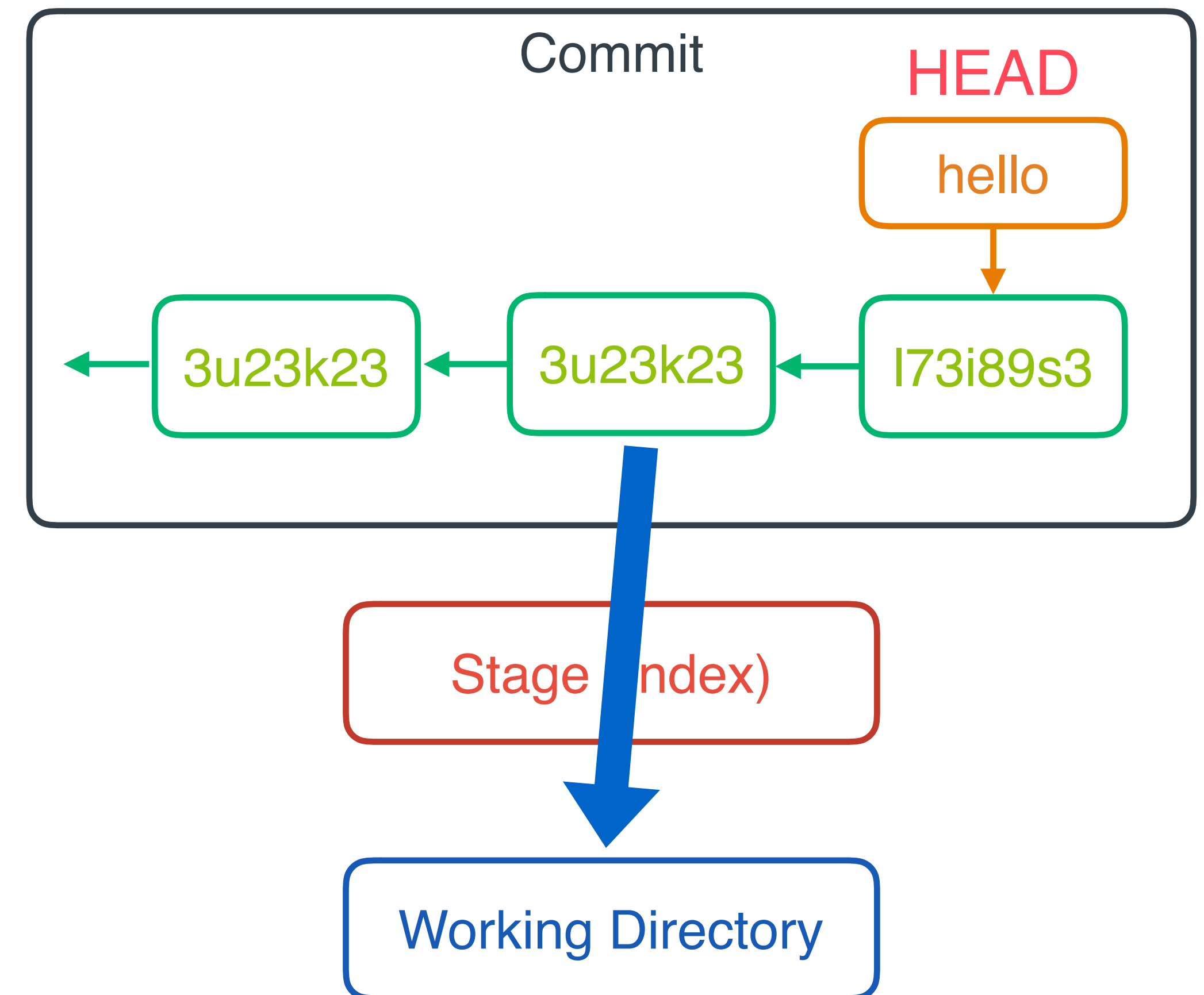
`git reset HEAD dharma.java` = 丢弃stage(`git add`)



`git checkout HEAD~2 dharma.java`

用指定提交替换工作目录内容，而阶段快照不变

`git checkout HEAD dharma.java` = 丢弃working内容



# 最佳实践之 PR



## Index page #1

**Merged** jeffer8a merged 2 commits into `master` from `index-page` 15 seconds ago

Conversation 0 Commits 2 Files changed 1



jeffer8a commented 5 minutes ago

Owner

Se ha creado la página index para el proyecto



jeffer8a added some commits 14 minutes ago



página de inicio de la aplicación

66f26c5



Merge branch 'master' of https://github.com/jcffer8a/StydcNet pull request

b33637d



jeffer8a merged commit 95e2423 into `master` 14 seconds ago

Revert



**Pull request successfully merged and closed**

You're all set—the `index-page` branch can be safely deleted

Delete branch



Write

Preview

Markdown supported

Edit in fullscreen

Leave a comment

Attach images by dragging & dropping, selecting them, or pasting from the clipboard.

Comment



# 最佳实践之 PR

Pull Request 是团队协作中代码审查的一种方式。PR用于告诉其他成员你希望做的改变，或者你正在做的功能已经完成，成员可以审查，同意或者添加评论表示质疑，在经过修改后达成一致，最终将代码合并。

# 最佳实践之 PR

1. 创建Pull Request
2. 填写PR内容：完成的功能，截图对比等
3. 团队Review代码：同意或评论
4. 代码修改，再次提交
5. 合并完成

The screenshot shows the GitHub 'Open a pull request' page. At the top, there's a navigation bar with tabs for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. Below this is a search bar with 'is:pr is:open' and buttons for 'Filters', 'Labels', 'Milestones', and a green 'New pull request' button. A summary bar shows '0 Open' and '145 Closed' pull requests, along with filters for Author, Labels, Projects, Milestones, Reviews, Assignee, and Sort. The main heading is 'Open a pull request' with a subtext: 'Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).' Below this is a comparison bar showing 'base: update-readme' and 'compare: modifications-to-143v' with a green checkmark and the text 'Able to merge. These branches can be automatically merged.' A yellow banner提醒用户提交前查看贡献指南。 The main content area has a title 'Modifications to 143v' and a 'Write' tab. Below the title is a large text area for comments with the placeholder 'Leave a comment'. At the bottom of the text area is a note: 'Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.' To the right of the text area is a sidebar with fields for 'Reviewers' (No reviews—request one), 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), and 'Milestone' (No milestone). At the bottom right is a green 'Create pull request' button.



# 作业

1. 在线练习 <https://try.github.io>
2. 学习Markdown语法 [参考链接](#)
3. 提交自己的一段代码
4. 创建自己的github页面 [Guide](#)





# Thanks!

Any questions?

