

# 编程基础 VI - 习题课

# Outline

- 迭代和递归
- Lambda验算
- 过程建模和数据建模
- 高阶函数
- 函数式编程
- Git II

# 递归和迭代

# + 加法

- 下面几个过程定义了一种加起来两个正整数的方法，他们都是基于过程inc（他将参数加1）和dec（他将参数减少1）
- (define (+ a b)
- (if (= a 0)
- b
- (inc (+ (dec a) b))))
- (define (+ a b)
- (if (= a 0)
- b
- (+ (dec a) (inc b))))
- 求值(+ 4 5)时的计算过程

Lamdar演算

# Lambda> SUB FOUR TWO

- **SUCC** =  $\lambda n. \lambda f. \lambda x. f (n f x)$
- **PLUS** =  $\lambda m. \lambda n. m \text{ SUCC } n$
- **PRED** =  $\lambda n. \lambda f. \lambda x. n (\lambda g. \lambda h. h (g f)) (\lambda u. x) (\lambda u. u)$
- **SUB** =  $\lambda m. \lambda n. n \text{ PRED } m$
- Lambda> SUB FOUR TWO
- $\backslash f. \backslash x. f (f x)$

# IF (EQ ONE TWO) a b

- Lambda> TRUE =  $\lambda x.\lambda y.x$
- Lambda> FALSE =  $\lambda x.\lambda y.y$
- Lambda> AND =  $\lambda p.\lambda q.p\ q\ p$
- Lambda> OR =  $\lambda p.\lambda q.p\ p\ q$
- Lambda> NOT =  $\lambda p.\lambda a.\lambda b.p\ b\ a$
- Lambda> IF =  $\lambda p.\lambda a.\lambda b.p\ a\ b$
- Lambda> ISZERO =  $\lambda n.n\ (\lambda x.FALSE)\ TRUE$
- Lambda> LEQ =  $\lambda m.\lambda n.ISZERO\ (SUB\ m\ n)$
- Lambda> EQ =  $\lambda m.\lambda n.\ AND\ (LEQ\ m\ n)\ (LEQ\ n\ m)$
  
- Lambda> IF (EQ ONE TWO) a b
- b

# LENGTH NIL

- Lambda> Y
- $\lambda g.(\lambda x.g (x x)) \lambda x.g (x x)$
- $YF = \beta F(YF)$ //Y的定义带入F
- Lambda> CONS =  $\lambda x.\lambda y.\lambda f. f x y$
- Lambda> CAR =  $\lambda p.p$  TRUE
- Lambda> CDR =  $\lambda p.p$  FALSE
- Lambda> NIL =  $\lambda x. TRUE$
- Lambda> NULL =  $\lambda p.p (\lambda x.\lambda y.FALSE)$
- Lambda> LENGTH =  $Y (\lambda g.\lambda c.\lambda x. NULL x c (g (SUCC c) (CDR x))) ZERO$
- Lambda> LENGTH NIL
- $\lambda f.\lambda x.x$

# 过程建模

# 累 + \* cons

- (define (accumulate op initial sequence)
- (if (null? sequence)
- initial
- (op (car sequence)
- (accumulate op initial (cdr sequence))))))
- (accumulate + 0 (list 1 2 3 4 5))
- 15
- (accumulate \* 1 (list 1 2 3 4 5))
- 120
- (accumulate cons nil (list 1 2 3 4 5))
- (1 2 3 4 5)

# 多项式求值

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

$$(\cdots (a_n x + a_{n-1})x + \cdots + a_1)x + a_0$$

- 想想 <??>里面填什么
- (define (horner-eval x coefficient-sequence)
- (accumulate (lambda (this-coeff higher-terms) <??>)
- 0
- coefficient-sequence))

# 多项式求值 $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$

$$(\dots (a_n x + a_{n-1})x + \dots + a_1)x + a_0$$

- (define (horner-eval x coefficient-sequence)
- (accumulate (lambda (this-coeff higher-terms)
- (+ this-coeff (\* x higher-terms)))
- 0
- coefficient-sequence))
- (assert-equal 10 (horner-eval 0 '(10)))
- (assert-equal 13 (horner-eval 10 '(3 1)))
- (assert-equal 79 (horner-eval 2 '(1 3 0 5 0 1)))

# accumulate-n

- (define (accumulate-n op init seqs)
- (if (null? (car seqs)) null
- (cons (accumulate op init (map car seqs))
- (accumulate-n op init (map cdr seqs)))))
- (assert-equal '(22 26 30) (accumulate-n + 0 '((1 2 3)
- (4 5 6)
- (7 8 9)
- (10 11 12)))))

# 数据建模

# 矩阵

- $\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix} \quad \begin{smallmatrix} + & - \\ - & + \end{smallmatrix}$

- $\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix} \quad \begin{smallmatrix} | & 1 & 2 & 3 & 4 & | \end{smallmatrix}$

- $\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix} \quad \begin{smallmatrix} | & 4 & 5 & 6 & 6 & | \end{smallmatrix}$

- $\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix} \quad \begin{smallmatrix} | & 6 & 7 & 8 & 9 & | \end{smallmatrix}$

- $\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix} \quad \begin{smallmatrix} + & - \\ - & + \end{smallmatrix}$

- $\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix}$

- $\begin{smallmatrix} \cdot & \cdot \\ \cdot & \cdot \end{smallmatrix}$  is represented as the sequence  $((1\ 2\ 3\ 4)\ (4\ 5\ 6\ 6)\ (6\ 7\ 8\ 9))'$ .

# Map

- (define (map proc items)
- (if (null? items)
- null
- (cons (proc (car items))
- (map proc (cdr items)))))

# Map

- Scheme standardly provides a map procedure that is more general than the one described here. This more general map takes a procedure of n arguments, together with n lists, and applies the procedure to all the first elements of the lists, all the second elements of the lists, and so on, returning a list of the results. For example:

- `(map + (list 1 2 3) (list 40 50 60) (list 700 800 900))`

- `(741 852 963)`

- `(map (lambda (x y) (+ x (* 2 y)))`

- `(list 1 2 3)`

- `(list 4 5 6))`

- `(9 12 15)`

# dot-product

- `(define (dot-product v w)`
- `(accumulate + 0 (map * v w)))`

# 矩阵运算

- (define (identity-n . x) x)
- (define (matrix-\*-vector m v)  
  (map (lambda (row) (dot-product v row)) m))
- (define (transpose mat)  
  (apply map cons (identity-n mat)))
- (define (transpose-slow mat)  
  (accumulate-n cons '() mat))
- (define (matrix-\*-matrix m n)  
  (let ((cols (transpose n)))  
    (map (lambda (row) (matrix-\*-vector cols row)) m)))

# 结果

- (assert-equal 7 (dot-product '(1 2 3) '(-1 1 2)))
- ;; [ 2 -1 1 ]
- ;; [ 0 -2 1 ] \* [1 2 3] = [3 -1 -3]
- ;; [ 1 -2 0 ]
- (define v '(1 2 3))
- (define m '((2 -1 1) (0 -2 1) (1 -2 0)))
- (assert-equal '(3 -1 -3) (matrix-\*-vector m v))
- (assert-many (lambda (f)
  - (assert-equal '((2 0 1) (-1 -2 -2) (1 1 0)) (f m)))
  - transpose-slow
  - transpose))
- ;; test data stolen from aja because I'm tired.
- (assert-equal '((19 22) (43 50)) (matrix-\*-matrix '((1 2) (3 4))
  - '((5 6) (7 8)))))

# 高阶函数

# 计算

$$\int_a^b f = \left[ f\left(a + \frac{dx}{2}\right) + f\left(a + dx + \frac{dx}{2}\right) + f\left(a + 2dx + \frac{dx}{2}\right) + \cdots \right] dx$$

- 还记得
  - (define (sum term a next b)
  - (if (> a b)
  - 0
  - (+ (term a)
  - (sum term (next a) next b))))

# 计算

$$\int_a^b f = \left[ f\left(a + \frac{dx}{2}\right) + f\left(a + dx + \frac{dx}{2}\right) + f\left(a + 2dx + \frac{dx}{2}\right) + \cdots \right] dx$$

- (define (integral f a b dx)
- (define (add-dx x) (+ x dx))
- (\* (sum f (+ a (/ dx 2.0)) add-dx b)
- dx))

- (integral cube 0 1 0.01)
- .24998750000000042
- (integral cube 0 1 0.001)
- .2499998750000001

# 函数式编程

# 例题 1

- 统计一本书中的所有长单词
- 长单词
  - 单词长度超过12

# 命令式实现

- `int count = 0;`
- `for(String w: words){`
- `if(w.length()>12) count++;`
- `}`

# 函数式实现

- `long count = words.stream().filter(w->w.length()>12).count();`

# 例题2

- 假如给定一个名称列表，其中一些名称包含一个字符。系统会要求您在一个逗号分隔的字符串中返回名称，该字符串中不包含单字母的名称，每个名称的首字母都大写。

# 命令式实现

- `public class TheCompanyProcess {`
- `public String cleanNames(List<String> listOfNames) {`
- `StringBuilder result = new StringBuilder();`
- `for(int i = 0; i < listOfNames.size(); i++) {`
- `if (listOfNames.get(i).length() > 1) {`
- `result.append(capitalizeString(listOfNames.get(i))).append(",");`
- `}`
- `}`
- `return result.substring(0, result.length() - 1).toString();`
- `}`
- `public String capitalizeString(String s) {`
- `return s.substring(0, 1).toUpperCase() + s.substring(1, s.length());`
- `}`
- `}`

# 函数式实现 - Java

- `public String cleanNames(List<String> names) {`
- `return names`
- `.stream()`
- `.filter(name -> name.length() > 1)`
- `.map(name -> capitalize(name))`
- `.collect(Collectors.joining(","));`
- `}`
  
- `private String capitalize(String e) {`
- `return e.substring(0, 1).toUpperCase() + e.substring(1, e.length());`
- `}`

# 函数式实现 - Java

- 并行版本
- `public String cleanNamesP(List<String> names) {`
- `return names`
- `.parallelStream()`
- `.filter(n -> n.length() > 1)`
- `.map(e -> capitalize(e))`
- `.collect(Collectors.joining(","));`
- `}`

# 函数式实现 - Scala

- 伪代码
  - `listOfEmps -> filter(x.length > 1) -> transform(x.capitalize) -> convert(x, y -> x + "," + y)`
- Scala版
  - `val employees = List("neal", "s", "stu", "j", "rich", "bob")`
  - `val result = employees`
  - `.filter(_.length() > 1)`
  - `.map(_.capitalize)`
  - `.reduce(_ + "," + _)`
- Scala并行版
  - `val parallelResult = employees`
  - `.par`
  - `.filter(f => f.length() > 1)`
  - `.map(f => f.capitalize)`
  - `.reduce(_ + "," + _)`
  - `.par` 方法返回后续操作依据的集合的并行版本。

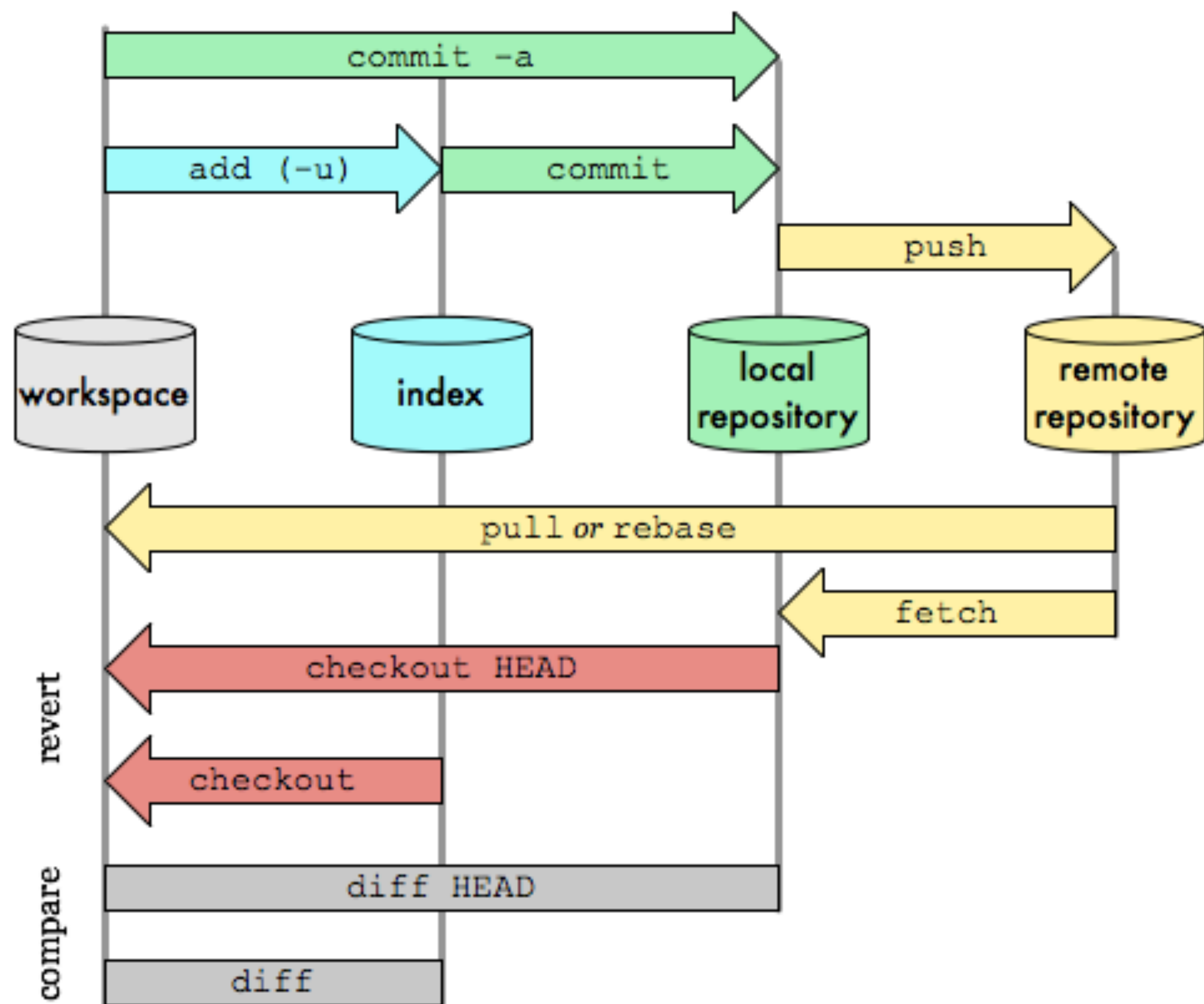
# 函数式编程 - python

- `from functools import reduce`
- `bigmuls = lambda xs,ys: filter(lambda t:t[0]*t[1] > 25, combine(xs,ys))`
- `combine = lambda xs,ys: zip(xs*len(ys), dupelms(ys,len(xs)))`
- `dupelms = lambda lst,n: reduce(lambda s,t:s+t, map(lambda l,n=n: [l]*n, lst))`

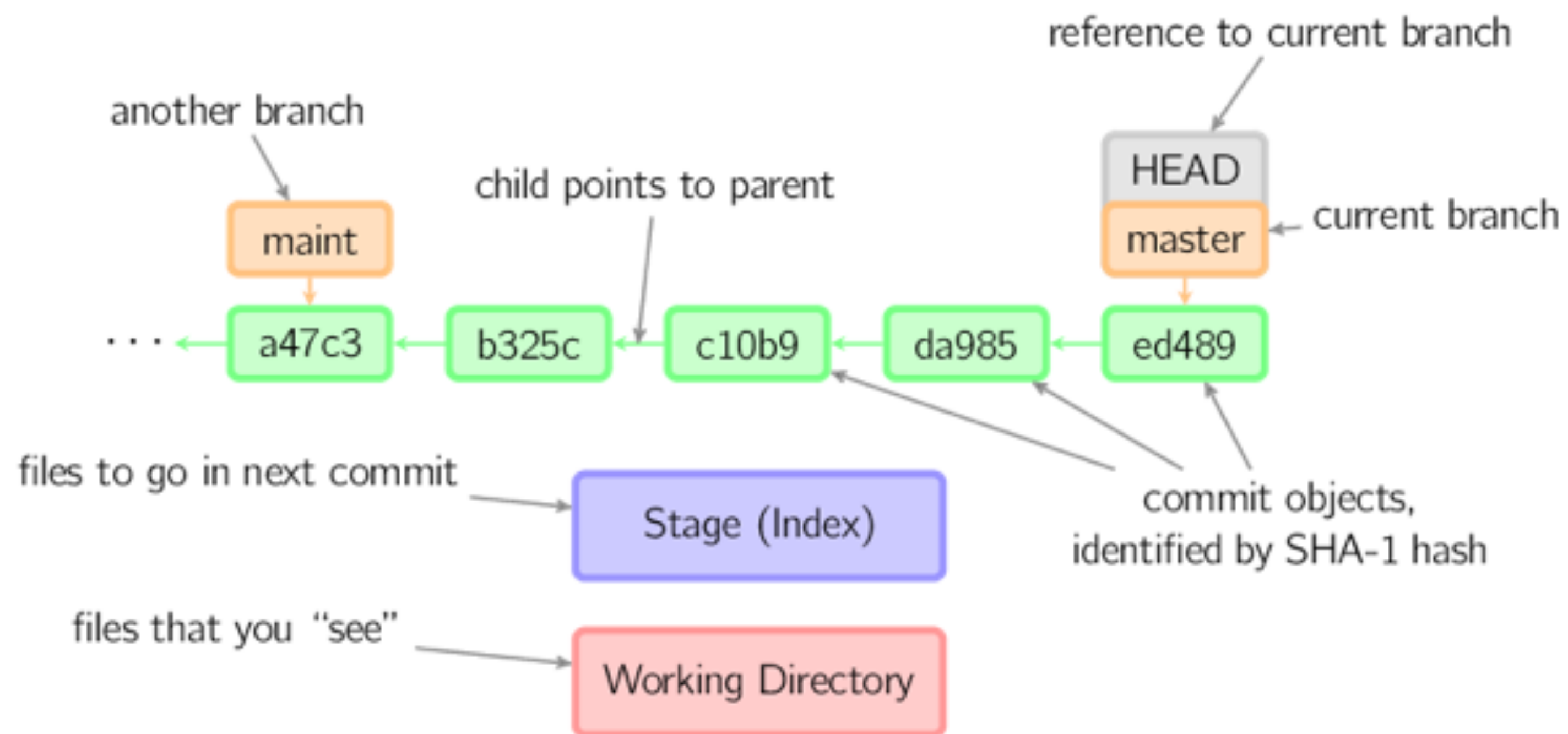
- `list1 = dupelms([10,15,3,22],len([1,2,3,4]))`
  - `print("dupelms:"+str([i for i in list1])+"\n")`
  - `list2 = combine([1,2,3,4],[10,15,3,22])`
  - `print("combine:"+str([i for i in list2])+"\n")`
  - `list3 = list(bigmuls([1,2,3,4],[10,15,3,22]))`
  - `print("bigmuls"+str([i for i in list3]))`
  - `print("for:"+str([(x,y) for x in (1,2,3,4) for y in (10,15,3,22) if x*y > 25]))`
- 
- `dupelms:[10, 10, 10, 10, 15, 15, 15, 15, 3, 3, 3, 3, 22, 22, 22, 22]`
  - `combine:[(1, 10), (2, 10), (3, 10), (4, 10), (1, 15), (2, 15), (3, 15), (4, 15), (1, 3), (2, 3), (3, 3), (4, 3), (1, 22), (2, 22), (3, 22), (4, 22)]`
  - `bigmuls[(3, 10), (4, 10), (2, 15), (3, 15), (4, 15), (2, 22), (3, 22), (4, 22)]`
  - `for:[(2, 15), (2, 22), (3, 10), (3, 15), (3, 22), (4, 10), (4, 15), (4, 22)]`

# Git II

# 工作流程



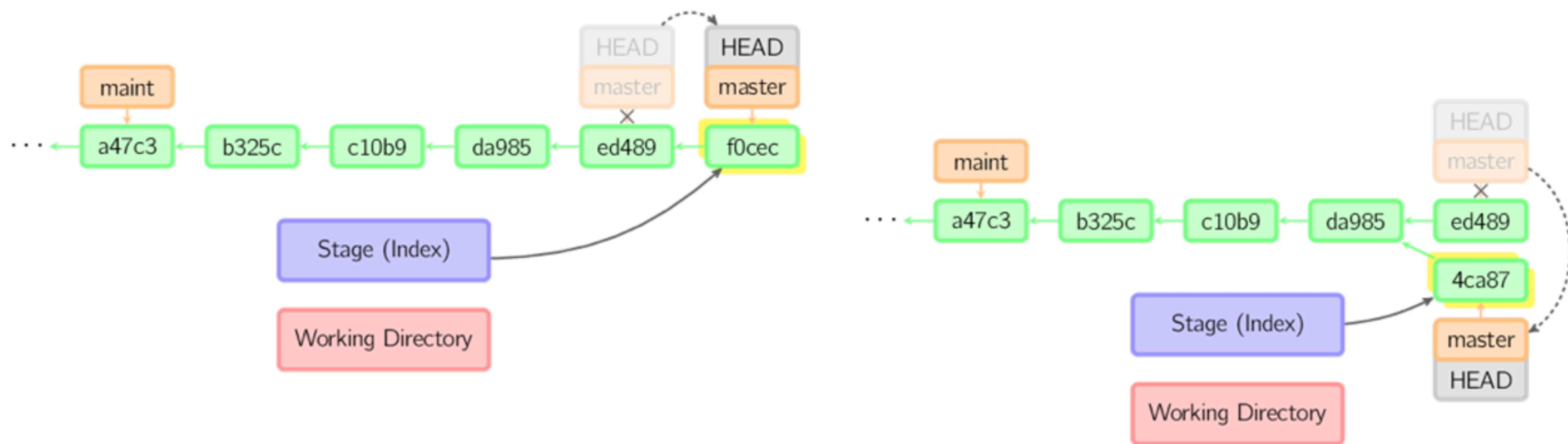
# 图例



# Commit

## commit

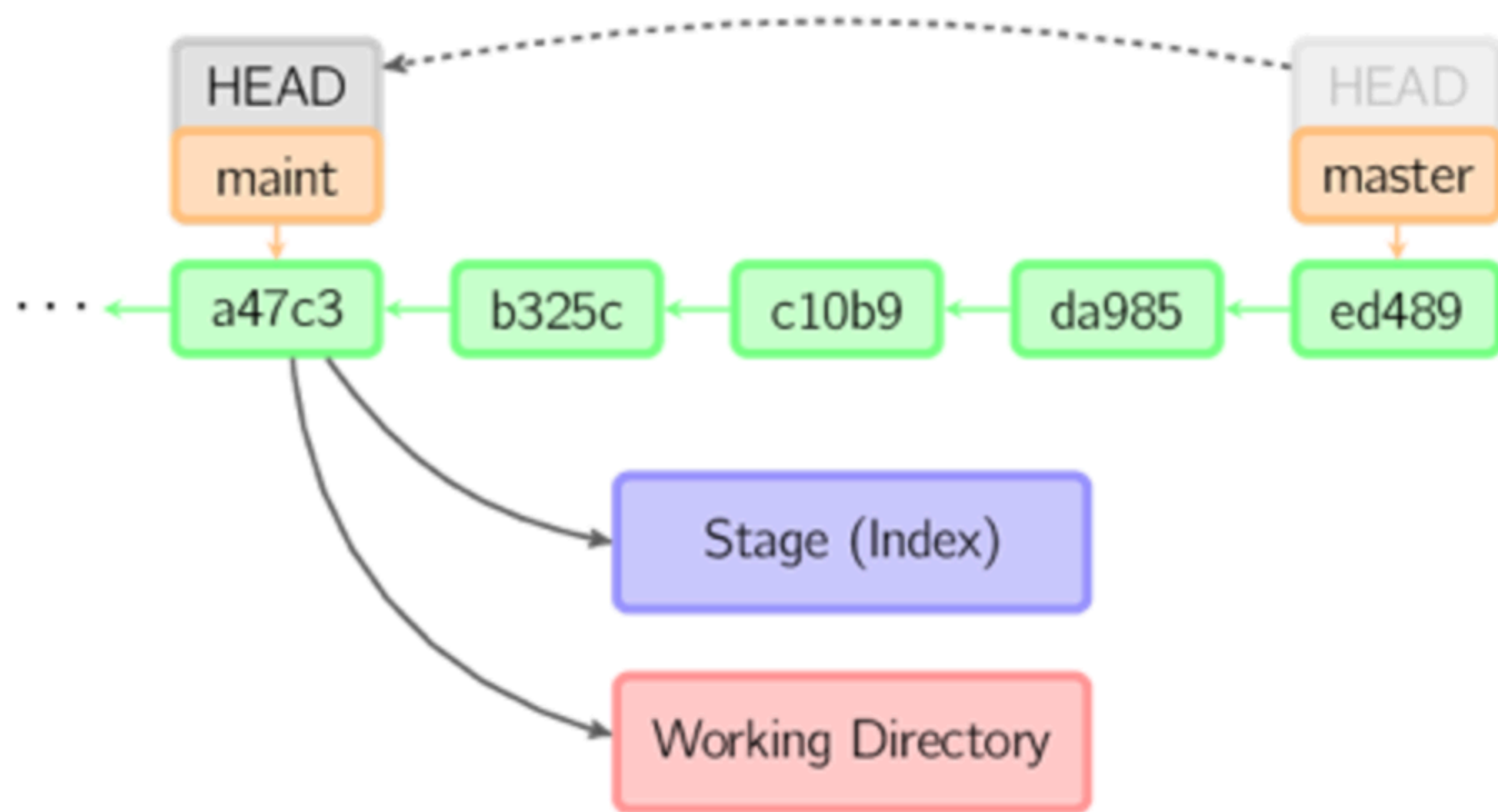
commit把暂存区的内容存入到本地仓库，并使得当前分支的HEAD向后移动一个提交点。如果对最后一次commit不满意，可以使用 `git commit --amend` 来进行撤销，修改之后再提交。如图所示的，ed489被4ca87取代，但是git log里看不到ed489的影子，这也正是amend的本意：原地修改，让上一次提交不露痕迹。



# Checkout

## checkout

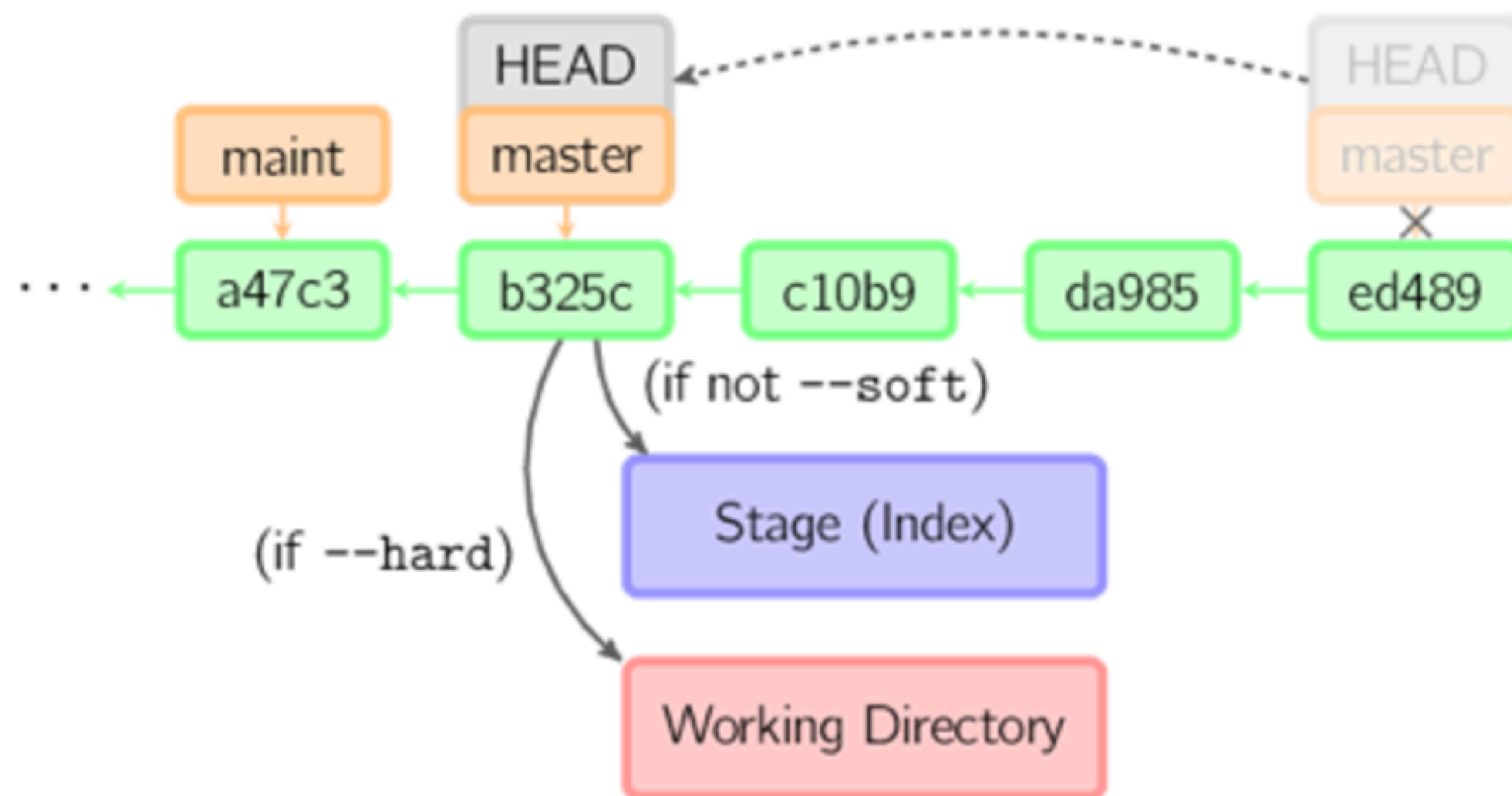
checkout用来检出并切换分支。checkout成功后，HEAD会指向被检出分支的最后一次提交点。对应的，工作目录、暂存区也都会与当前的分支进行匹配。下图是执行`git checkout maint`后的结果：



# Reset

## reset

reset命令把当前分支指向另一个位置，并且相应的变动工作目录和索引。如下图，执行`git reset HEAD~3`后，当前分支相当于回滚了3个提交点，由ed489回到了b325c：



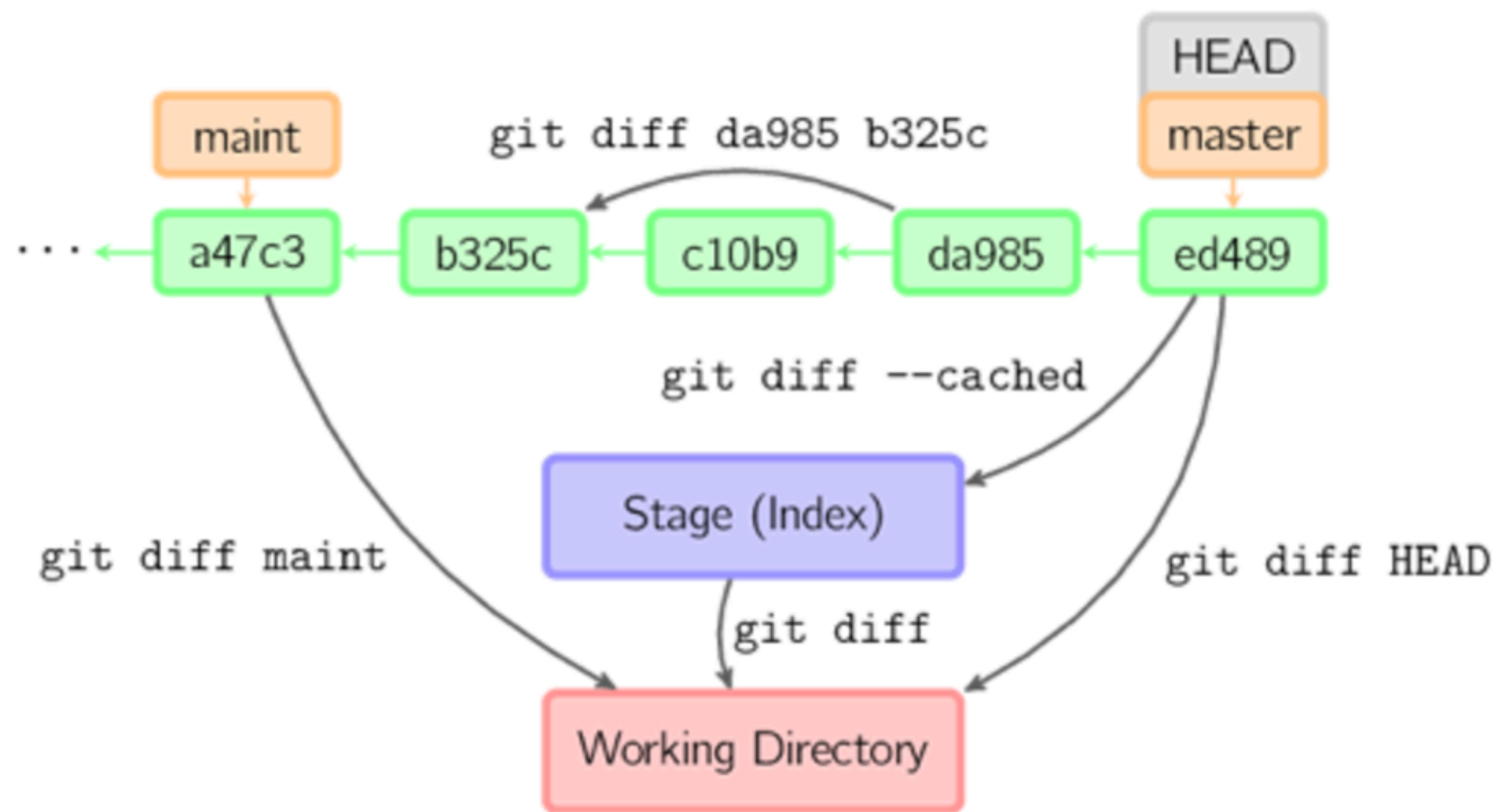
reset有3种常用的模式：

- soft，只改变提交点，暂存区和工作目录的内容都不改变
- mixed，改变提交点，同时改变暂存区的内容。这是默认的回滚方式
- hard，暂存区、工作目录的内容都会被修改到与提交点完全一致的状态

# Diff

## diff

我们在commit、merge、rebase、打patch之前，通常都需要看看这次提交都干了些什么，于是diff命令就派上用场了：



来比较下上图中5种不同的diff方式：

比较不同的提交点之间的异同，用 `git diff 提交点1 提交点2`

比较当前分支与其他分支的异同，用 `git diff 其他分支名称`

在当前分支内部进行比较，比较最新提交点与当前工作目录，用 `git diff HEAD`

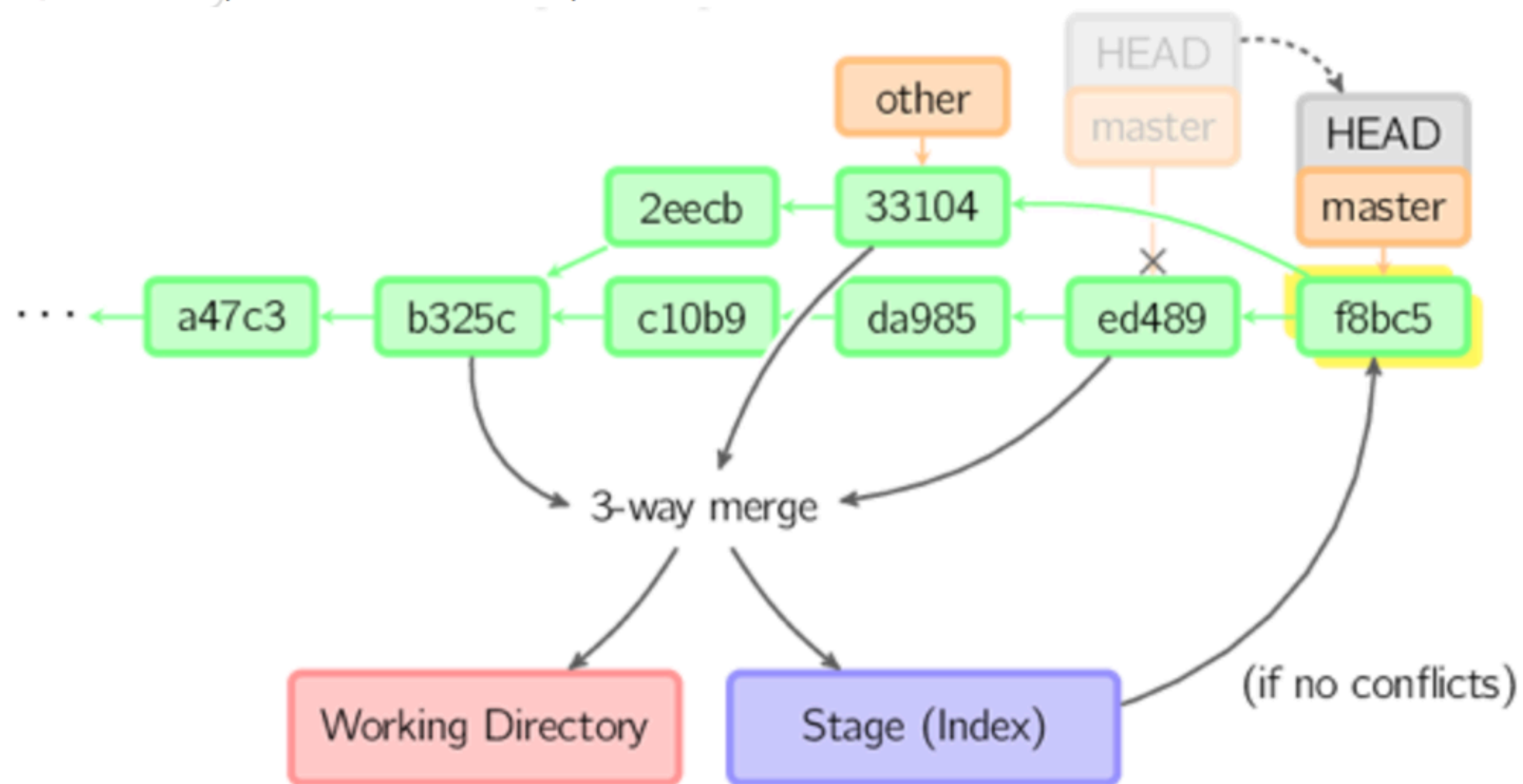
在当前分支内部进行比较，比较最新提交点与暂存区的内容，用 `git diff --cached`

在当前分支内部进行比较，比较暂存区与当前工作目录，用 `git diff`

# Merge

## merge

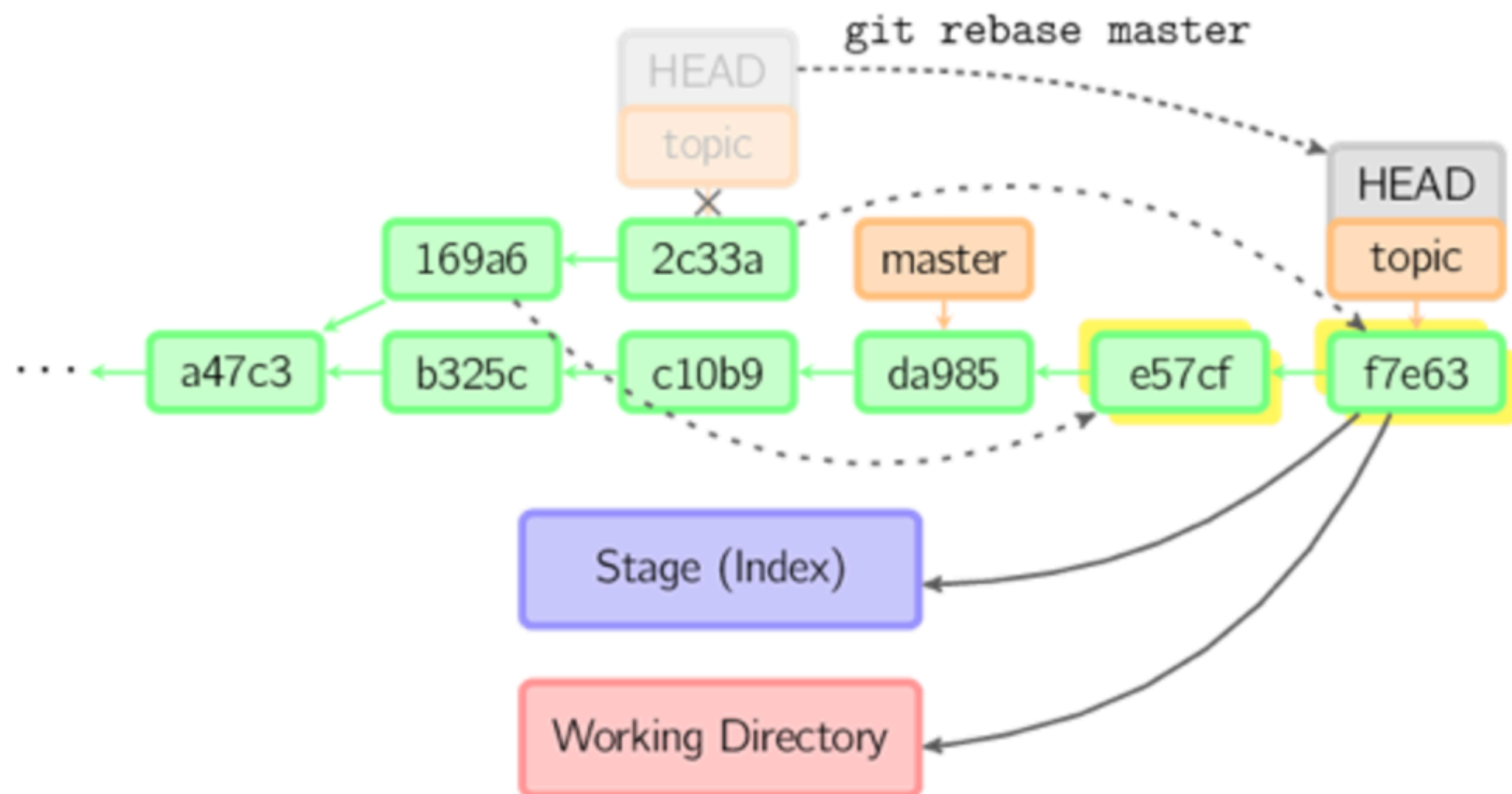
merge命令把不同的分支合并起来。如下图，HEAD处于master分支的ed489提交点上，other分支处于33104提交点上，项目负责人看了下觉得other分支的代码写的不错，于是想把代码合并到master分支，因此直接执行 `git merge other`，如果没有发生冲突，other就成功合并到master分支了。



# Rebase

## rebase

rebase又称为衍合，是合并的另外一种选择。merge把两个分支合并到一起进行提交，无论从它们公共的父节点开始(如上图，other分支与 master分支公共的父节点b325c)，被合并的分支(other分支)发生过多少次提交，合并都只会在当前的分支上产生一次提交日志，如上图的 f8bc5。所以merge产生的提交日志不是线性的，万一某天需要回滚，就只能把merge整体回滚。而rebase可以理解为verbosely merge，完全重演下图分支topic的演化过程到master分支上。如下图：



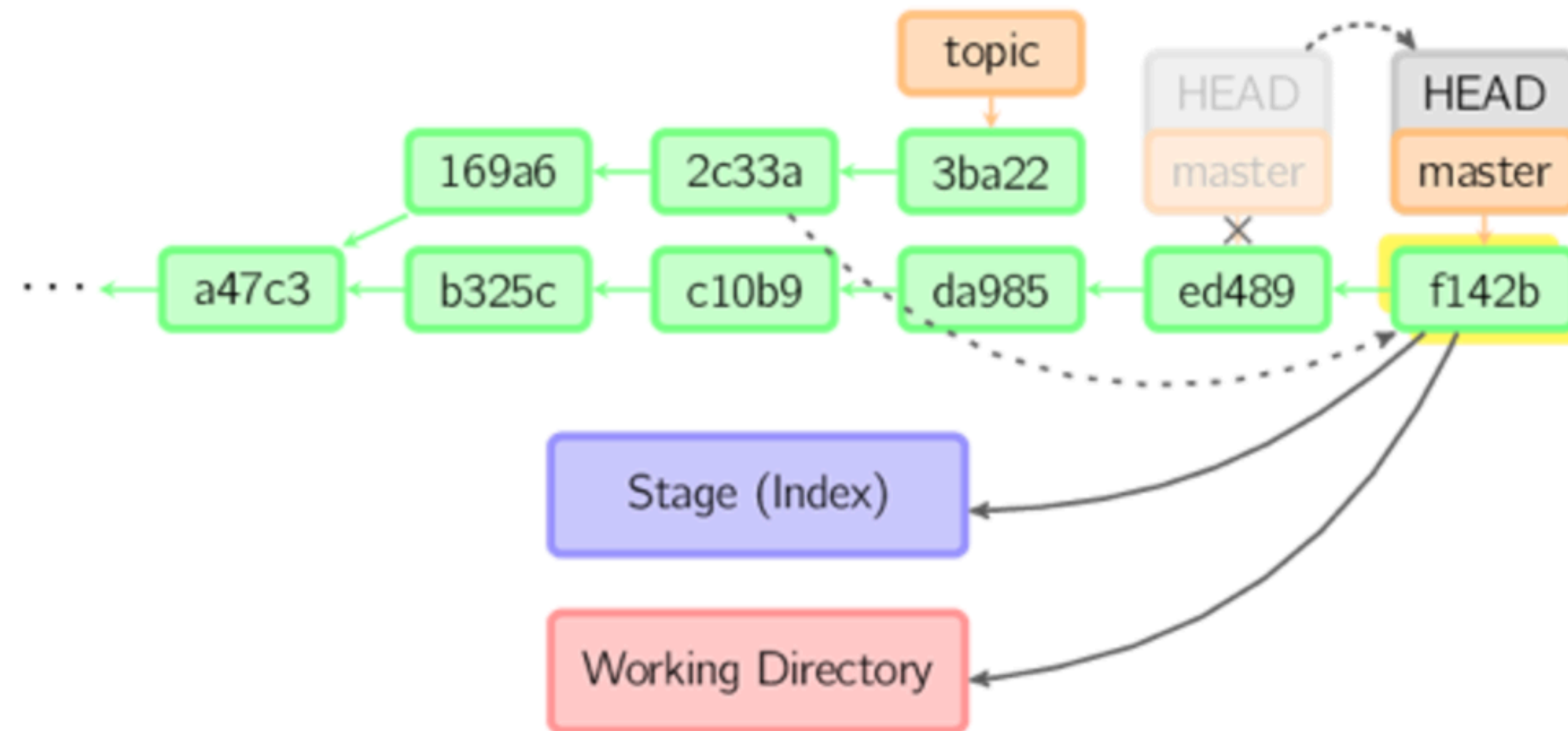
在开始阶段，我们处于topic分支上，执行 `git rebase master`，那么169a6和2c33a上发生的事情都在master分支上重演一遍，分别对应于master分支上的e57cf和f7e63，最后checkout切换回到topic分支。这一点与merge是一样的，合并前后所处的分支并没有改变。`git rebase master`，通俗的解释就是topic分支想站在master的肩膀上继续下去。

# Cherry-pick

## cherry-pick

cherry-pick命令复制一个提交点所做的工作，把它完整的应用到当前分支的某个提交点上。rebase可以认为是自动化的线性的cherry-pick。

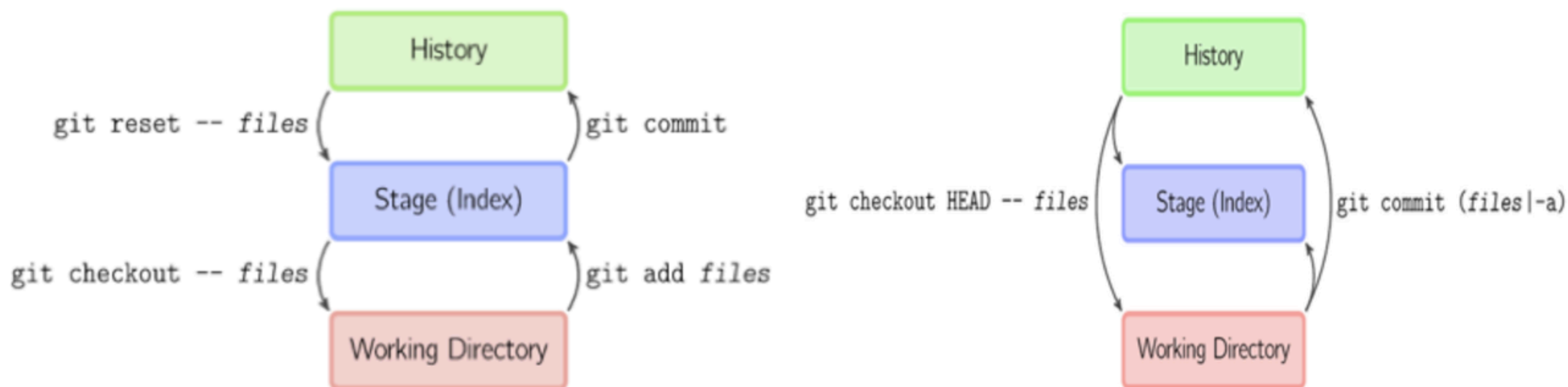
例如执行`git cherry-pick 2c33a`:



# 正反过程对比

## 正反过程对比

理解了上面最晦涩的几个命令，我们来从正反两个方向对比下版本在本地的3个阶段之间是如何转化的。如下图(history就是本地仓库)：



如果觉得从本地工作目录到本地历史库每次都要经过index暂存区过渡不方便，可以采用图形右边的方式，把两步合并为一步。