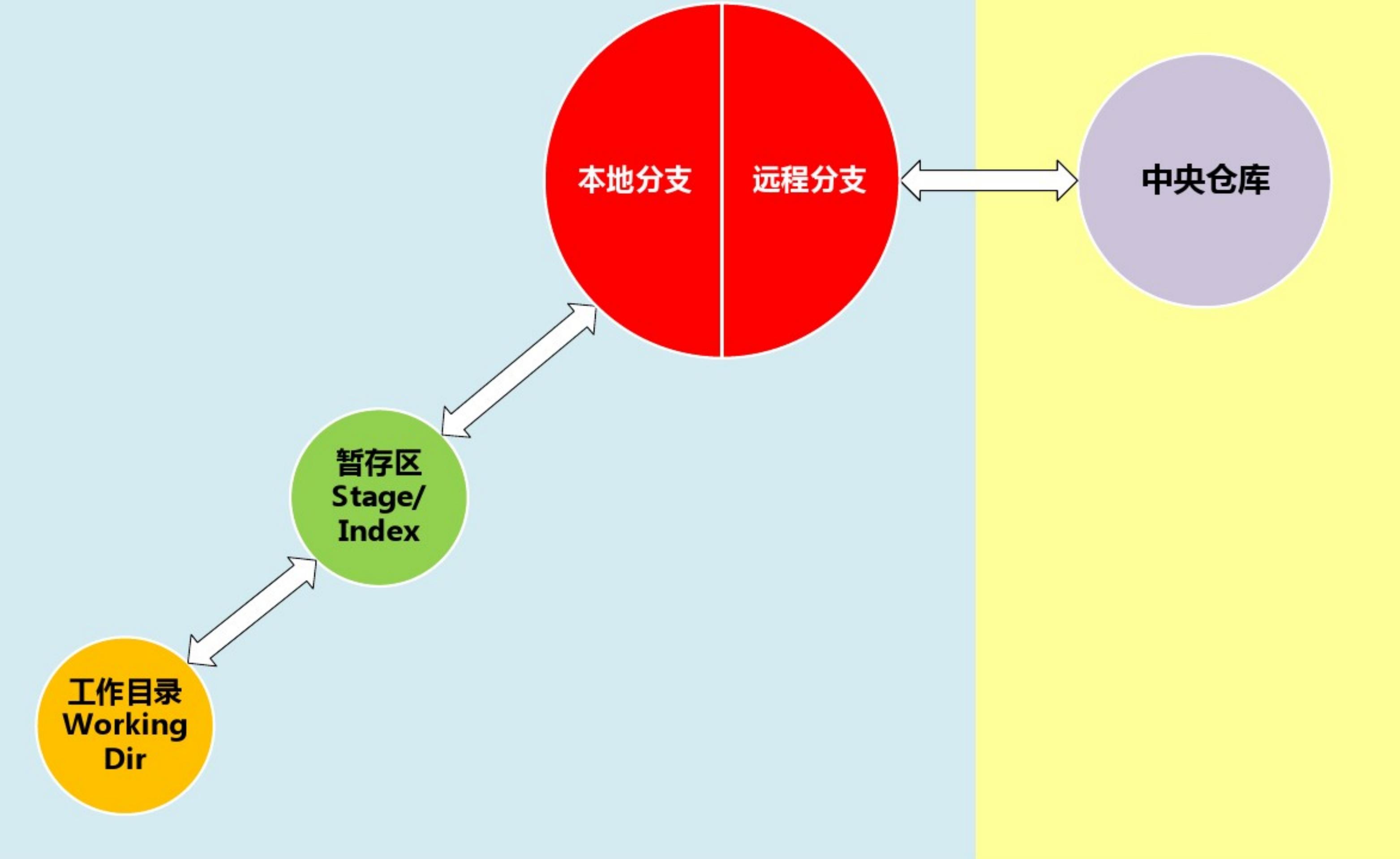


GIT 使用指南

GIT优势： 玩转分支

- 基本概念
- 常用操作
- 注意事项
- 典型场景
- 深入理解

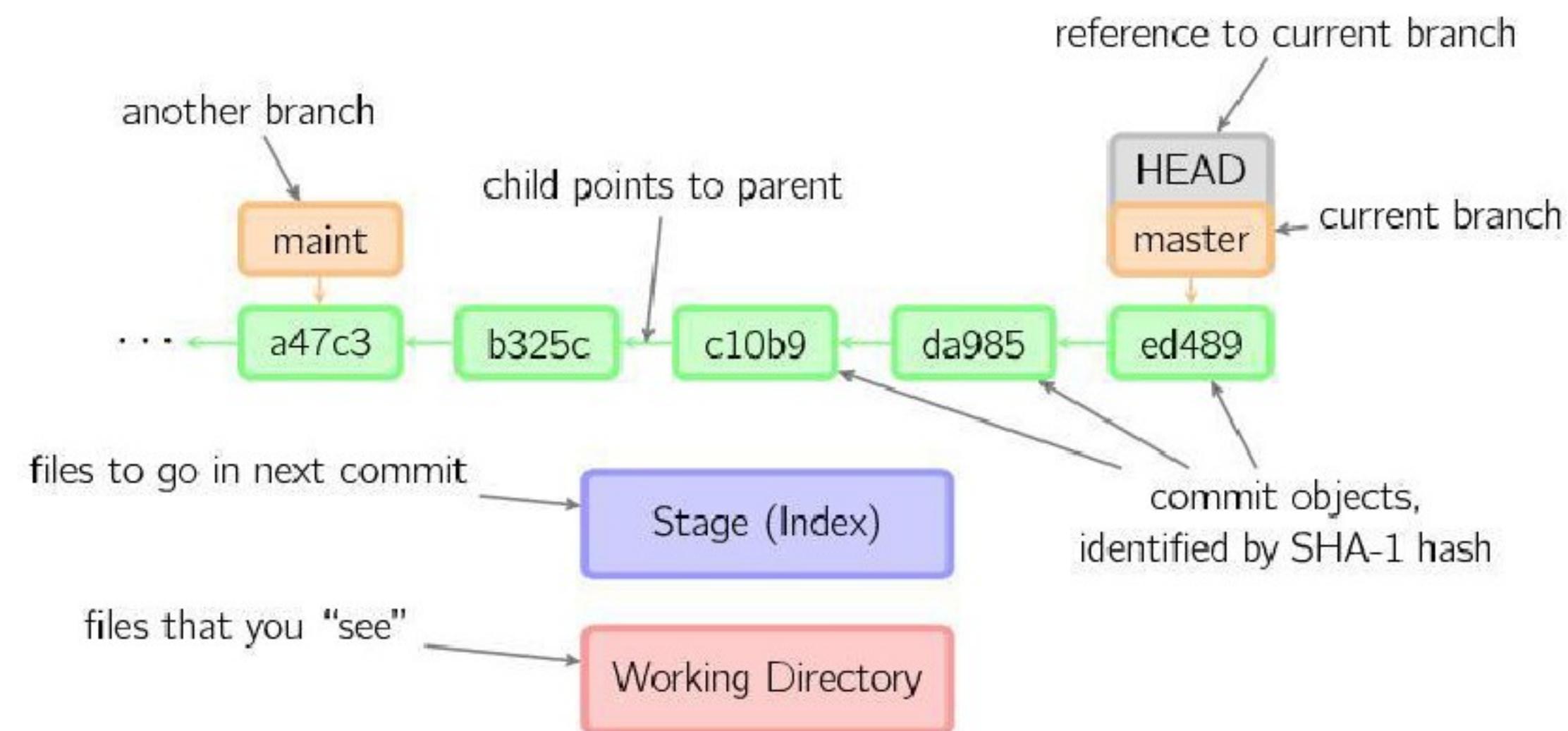
基本概念



引用 (Ref)

- GIT中的分支名、远程分支名、tag等都是指向某个commit的引用。
- 比如master分支,origin/master远程分支,v1.2tag等都是引用，它们通过保存某个commit的SHA-1值，从而指向某个commit。
- 分支名指向的commit，就是这个分支上最新的commit。

引用 (Ref)



常用操作

下载代码

- 【SVN】

```
svn checkout http://svn.sh.gj.com/gsw/S304/trunk/platform
```

- 【GIT】

```
git clone ssh://zhangdingli@10.58.100.6:29418/S304 [--depth=版本数] // 复制远程仓库到本地，可以选择需要复制的版本数，减少本地仓库占用空间
```

更新代码

- 【SVN】

`svn update`

- 【GIT】

`svn pull` // 更新代码到本地远程仓库、本地仓库、暂存区和工作目录

提交代码

- 【SVN】

```
svn add <files>
```

```
svn del <files>
```

```
svn commit -m ""
```

- 【GIT】

```
git add <paths> // 保存文件或目录到缓存区
```

```
git rm <paths> // 从缓存区删除文件或目录
```

```
git mv <paths> <paths> // 重命名缓存区中的文件或目录
```

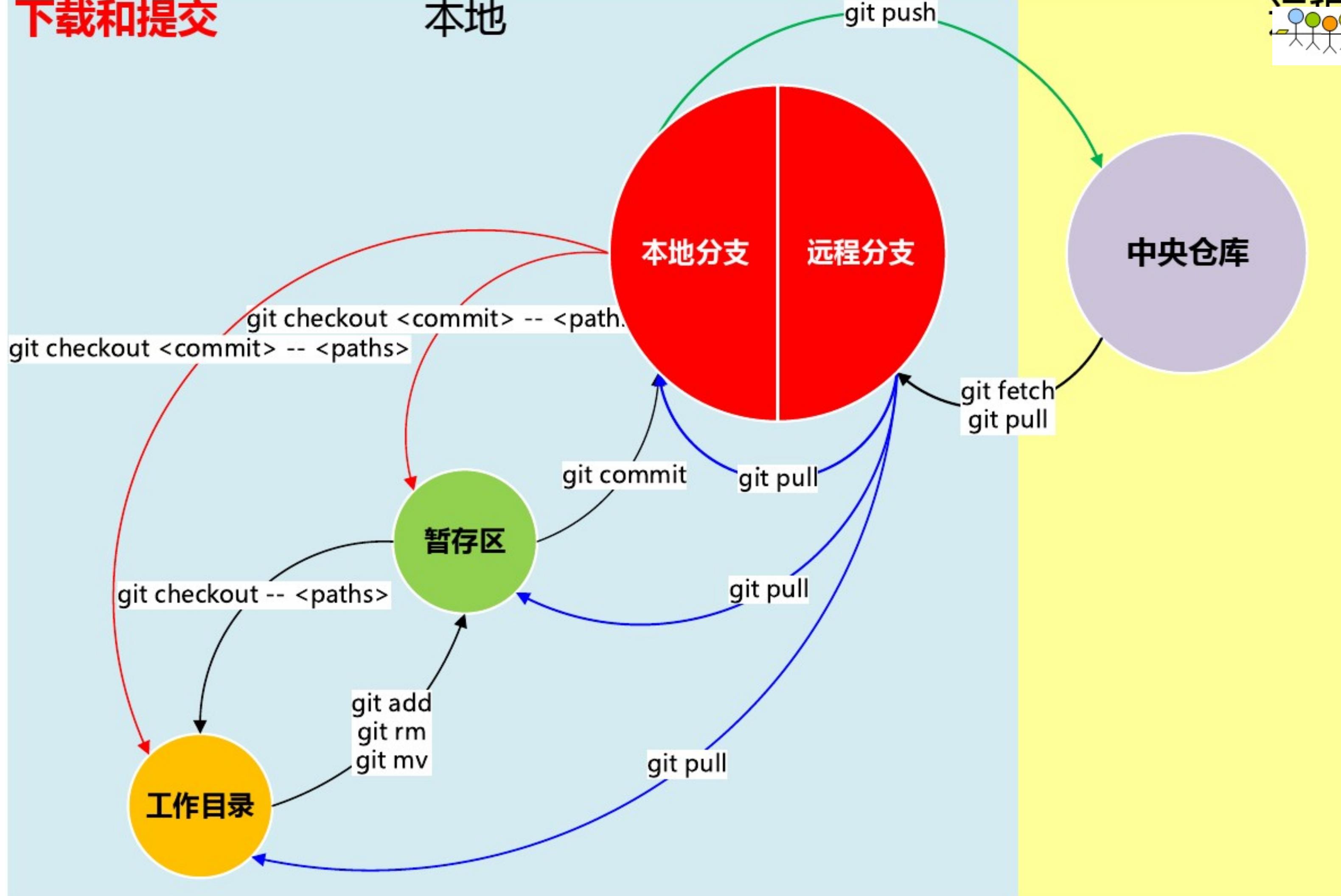
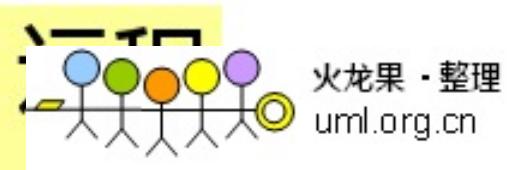
```
git commit -m "代码提交信息" // 将缓存区提交到本地仓库
```

```
git commit --amend // 合并最后一次提交
```

```
git push origin master // 将本地仓库主干推送到中央仓库
```

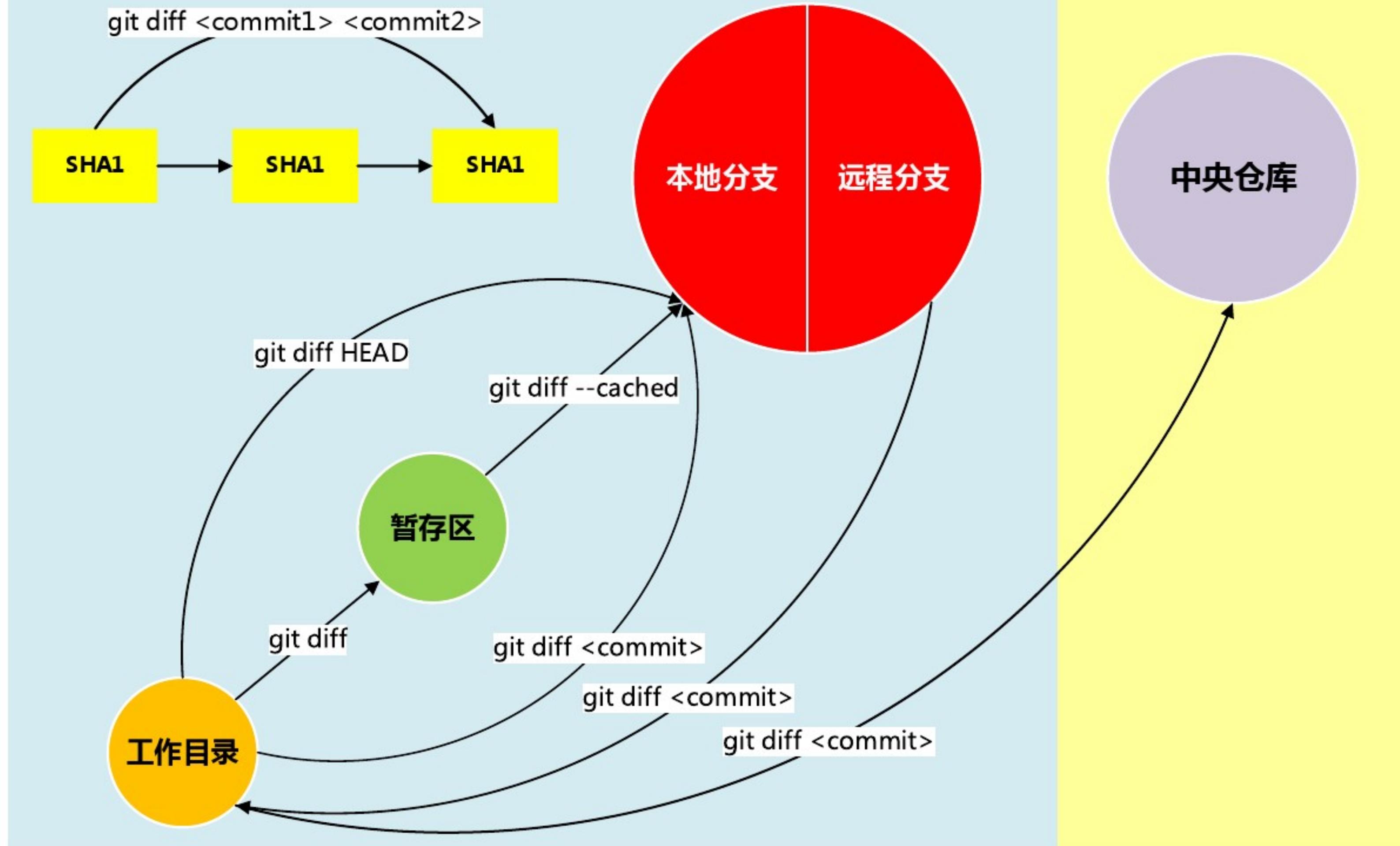
下载和提交

本地



比较操作

本地



恢复代码

- 【SVN】

```
svn revert -R
```

- 【GIT】

```
git reset HEAD -- <paths> //从本地仓库恢复文件或目录
```

```
git checkout HEAD -- <paths> //从本地仓库恢复文件或目录
```

```
git checkout -- <paths> // 从暂存区恢复文件或目录
```

```
git clean -dfx [-- <paths>] // 删除所有没有被跟踪的文件
```

回滚代码

- 【SVN】

```
svn merge -r 大版本号:小版本号 .
```

```
svn commit
```

- 【GIT】

```
git reset <commit> [--hard] // 回滚本地仓库到历史提交
```

通过 *reset* 回滚代码可能导致不可恢复， 肾用！

合并代码

- 【SVN】

```
svn merge -r 小版本号:大版本号
```

- 【GIT】

```
git merge <b1> // 合并b1分支的改动到当前分支
```

```
git rebase <b1> // 重放b1分支的改动到当前分支
```

```
git cherry-pick <commit> // 合并指定提交到当前分支
```

```
git rebase HEAD~2 // 合并最近两次提交成一个提交
```

分支操作

- 【SVN】

创建分支: svn copy

切换分支: svn switch

- 【GIT】

创建分支: git checkout -b <b1>

切换分支: git checkout b1

工作进度

- `git stash [save “message...”]` // 保存工作进度（即保存工作目录和暂存区的快照）
- `git stash list` // 查看工作进度
- `git stash pop [--index] [<stash>]` // 恢复并删除最近的工作进度
- `git stash apply [--index] [<stash>]` // 恢复最近的工作进度
- `git stash drop [<stash>]` // 删除工作进度
- `git stash clear` // 清空所有工作进度

标签

- 在 git 中有两种最主要的标签—轻量级标签（**lightweight**）和带注释的标签（**annotated**）。
- 轻量级标签跟分枝一样，不会改变。它就是针对某个特定提交的指针。
- 带注释的标签是git仓库中的对象。它是一组校验和，包含标签名、email、日期，标签信息，GPG签名和验证。
- 一般情况下，建议创建带注释的标签，这样就会保留这些信息，但是如果你只是需要临时性标签或者某些原因你不想在标签中附带上面说的这些信息，**lightweight**标签更合适些。

标签操作

- `git tag [-l XXX]` // 查看标签
- `git tag v1.4-lw` // 创建轻量级标签
- `git tag -a v1.4 -m 'version 1.4'` // 创建带注释的标签
- `git show v1.4` // 查看标签详细信息
- `git push -tags` // 提交标签到中央仓库

其他常用操作

- `git status [-s]` // 查看状态
- `git log [--pretty=raw]` // 查看日志
- `git show` // 查看版本细节
- `git branch [-r]` // 查看分支
- `git ls-files -dm` // 插件文件
- `git reflog` // 查看分支HEAD操作记录
- `git revert` // 回滚某次提交

注意事项

隐藏目录.git尽量不要删！！！

- 执行clone命令后，远程仓库会被复制到本地目录，即本地目录下的隐藏目录.git，一旦本地仓库被删除，本地提交将全部丢失
- 推荐大家将实际的.git目录移到比较安全的地方，然后在本地目录下创建一个到实际.git目录的软链接，具体操作方法如下：
- `mkdir ~/.gitrepo`
- `mv .git ~/.gitrepo/S304`
- `ln -s ~/.gitrepo/S304 .git`

空目录

- 和SVN不同，GIT只会跟踪文件，不会跟踪目录，所以无法将一个空目录提交到GIT仓库
- 有一种变通的方法来提交空目录，即在空目录下面新建要给空文件.gitignore，然后把.gitignore提交到GIT仓库

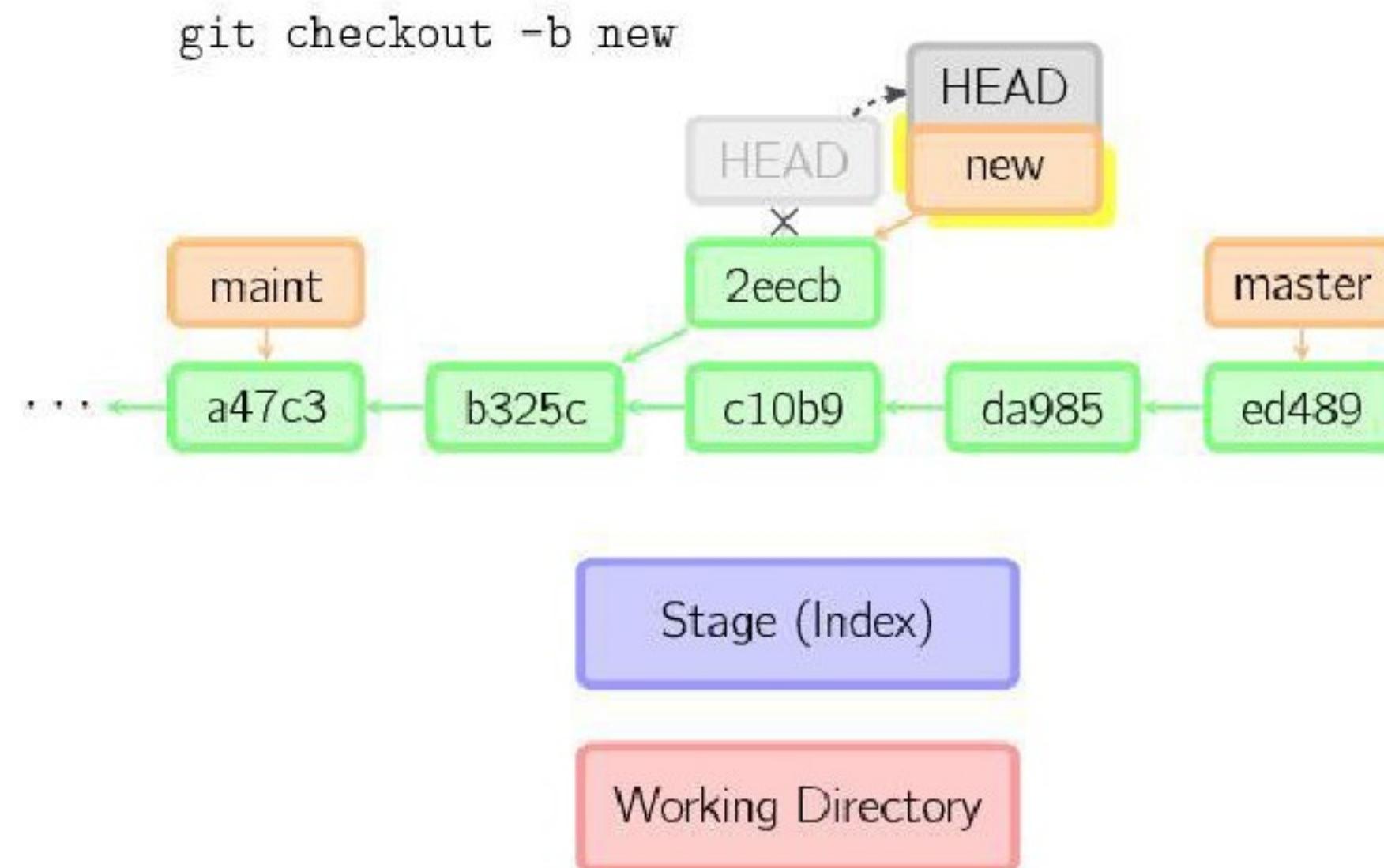
常用场景

编译发布版本的一般步骤

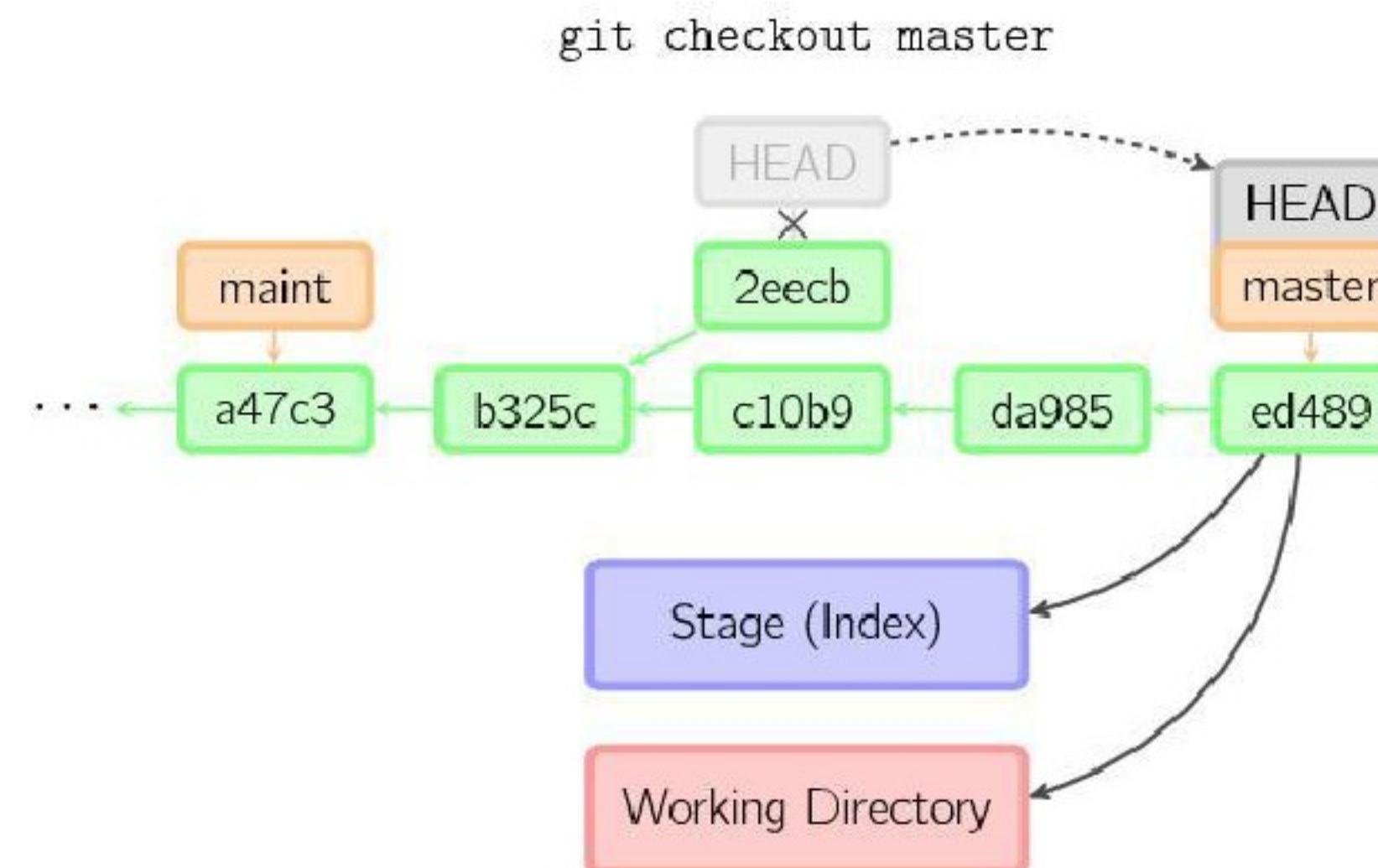
- `git stash //` 保存当前工作进度
- `git reset –hard -- .../products //` 恢复platform和products的本地修改
- `git clean –dfx -- .../products //` 清空platform和products目录下新生成的文件
- `git status –s //` 检查本地文件状态
- `git stash pop //` 恢复当前工作进度

深入理解

创建分支



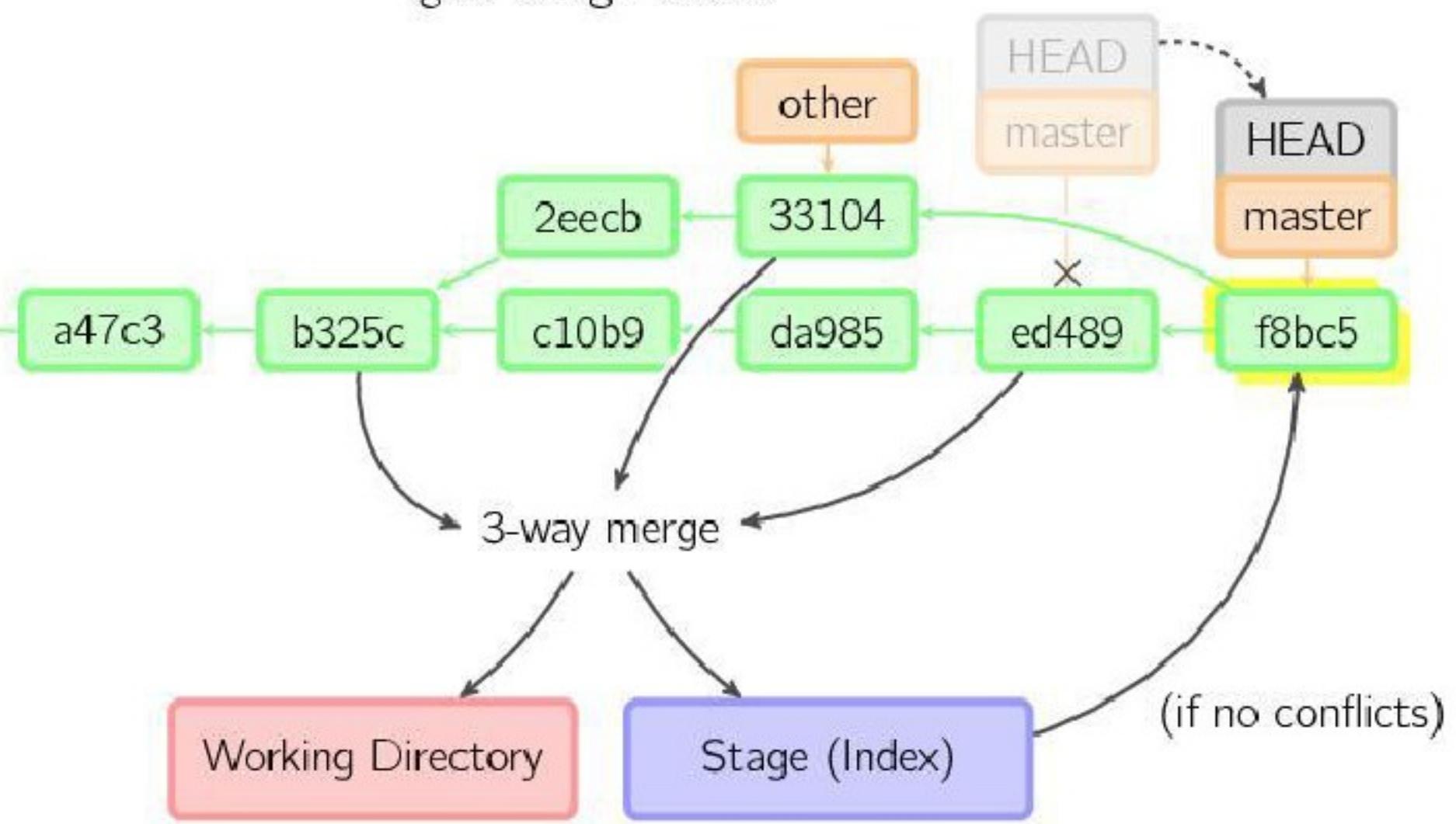
切换分支



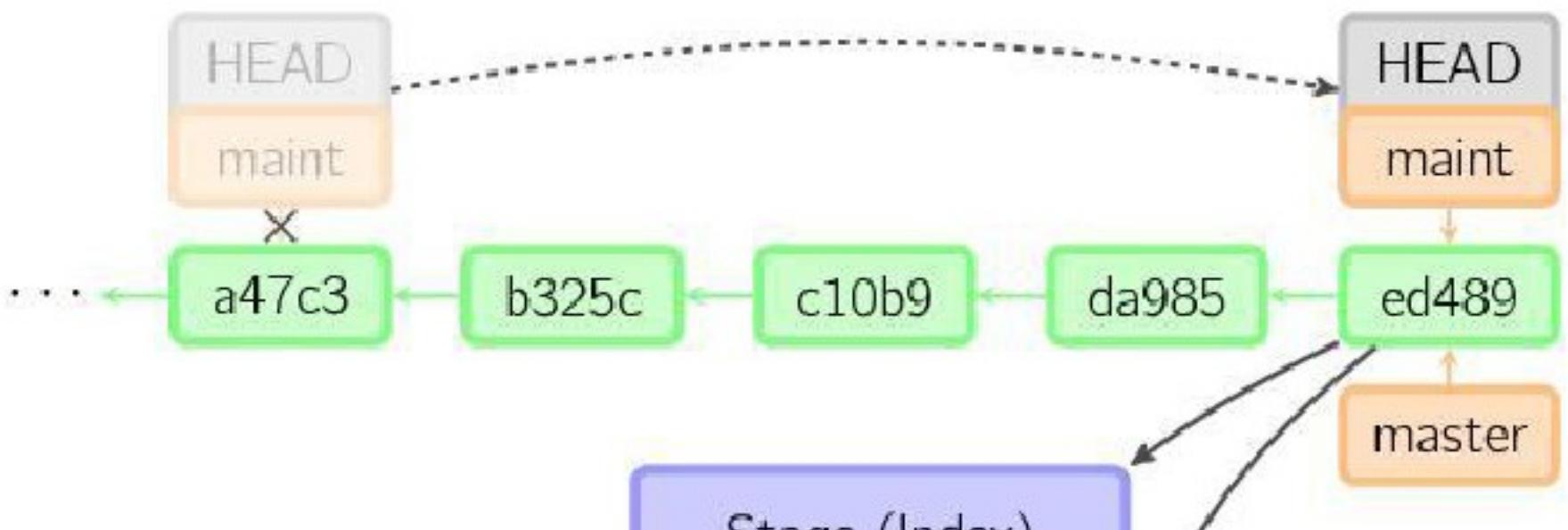
git merge master

合并 (Merge)

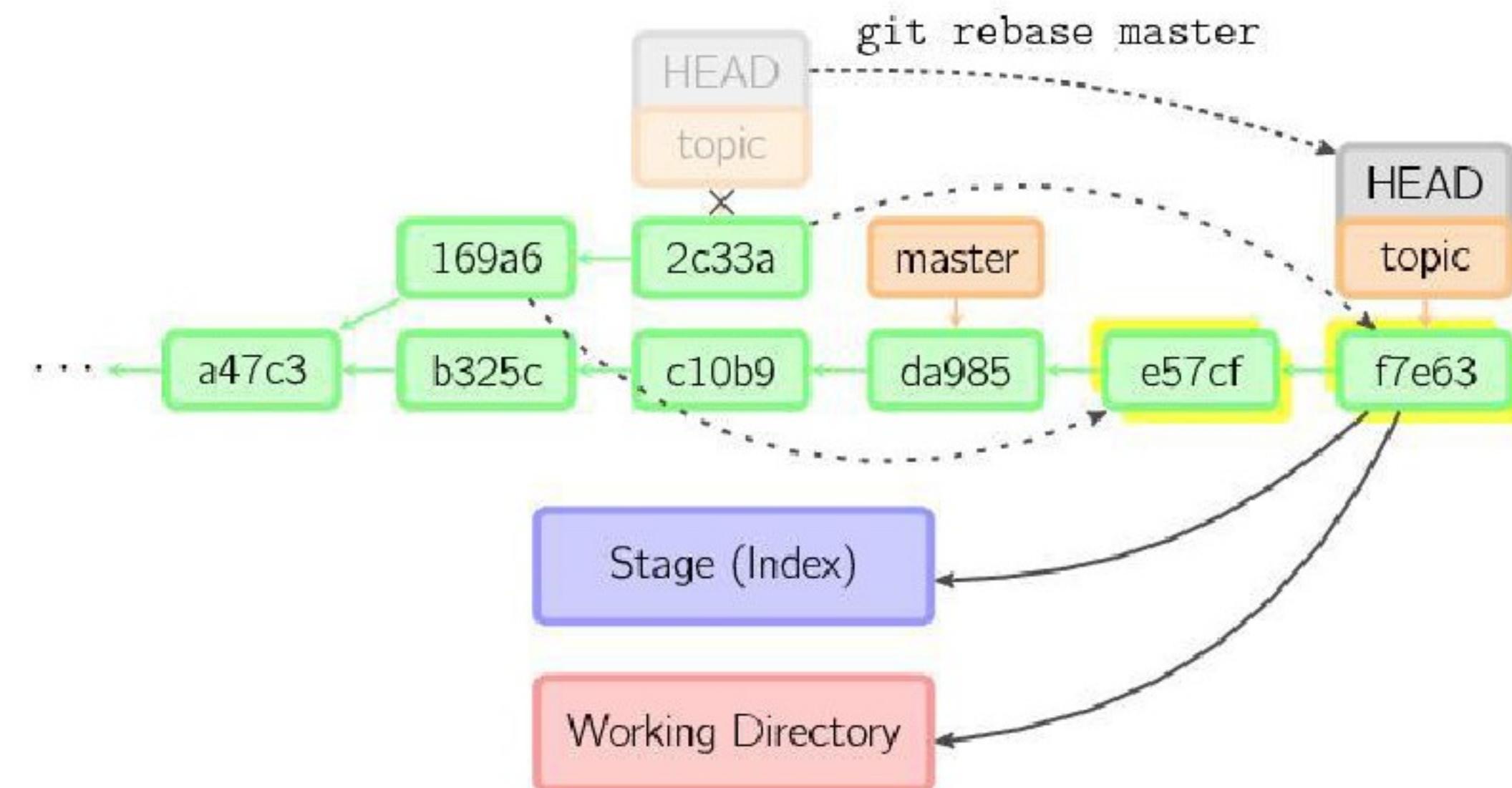
git merge other



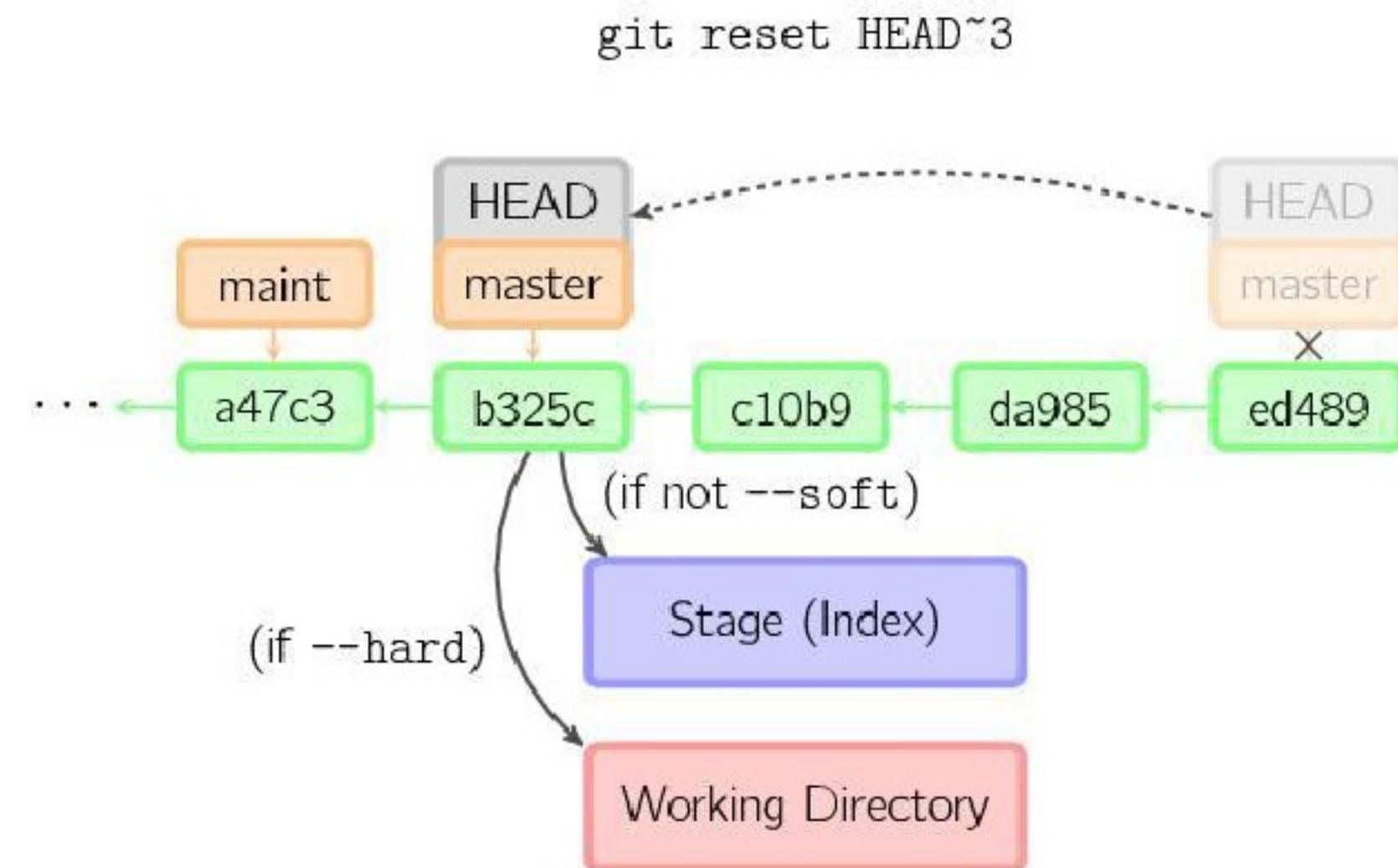
(if no conflicts)



合并 (Rebase)



回滚



参考资料

- <http://git-scm.com/book/zh/v1>
- <https://www.atlassian.com/git/tutorials/>