**工作区和暂存区：**

**工作区（Working Directory）:**

就是存放git源码的目录。

**版本库（Repository）:**

工作区有一个隐藏目录.git，这是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的**暂存区**，还有Git自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



往Git版本库里加文件，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到**暂存区**；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支，也就是说commit针对的是暂存区的内容。

**放弃未提交的文件修改：**

场景1：改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令

git checkout -- <file> //注意，--和<file>之间必须有空格。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

**git删除文件：**

git rm可以删除一个文件，并从git仓库中移除。注意：要执行commit后才真正提交。

git rm <file> //删文件

git rm -r <folder> //删文件夹

git rm –cached <file> //删除已经提交到暂存区的文件，但不会对文件本身做改动

git rm的误删的文件被提交之前可以用checkout恢复。

**git分支管理：**

git branch <new branch> //创建新分支

git checkout <branch name> //切换分支

git checkout -b <new branch> //创建新分支并切换过去

git checkout -b <new branch> --track <remote>/<branch> //创建新分支并切换过去，并指定对应的远端分支

git branch -d <branch name> //删除分支

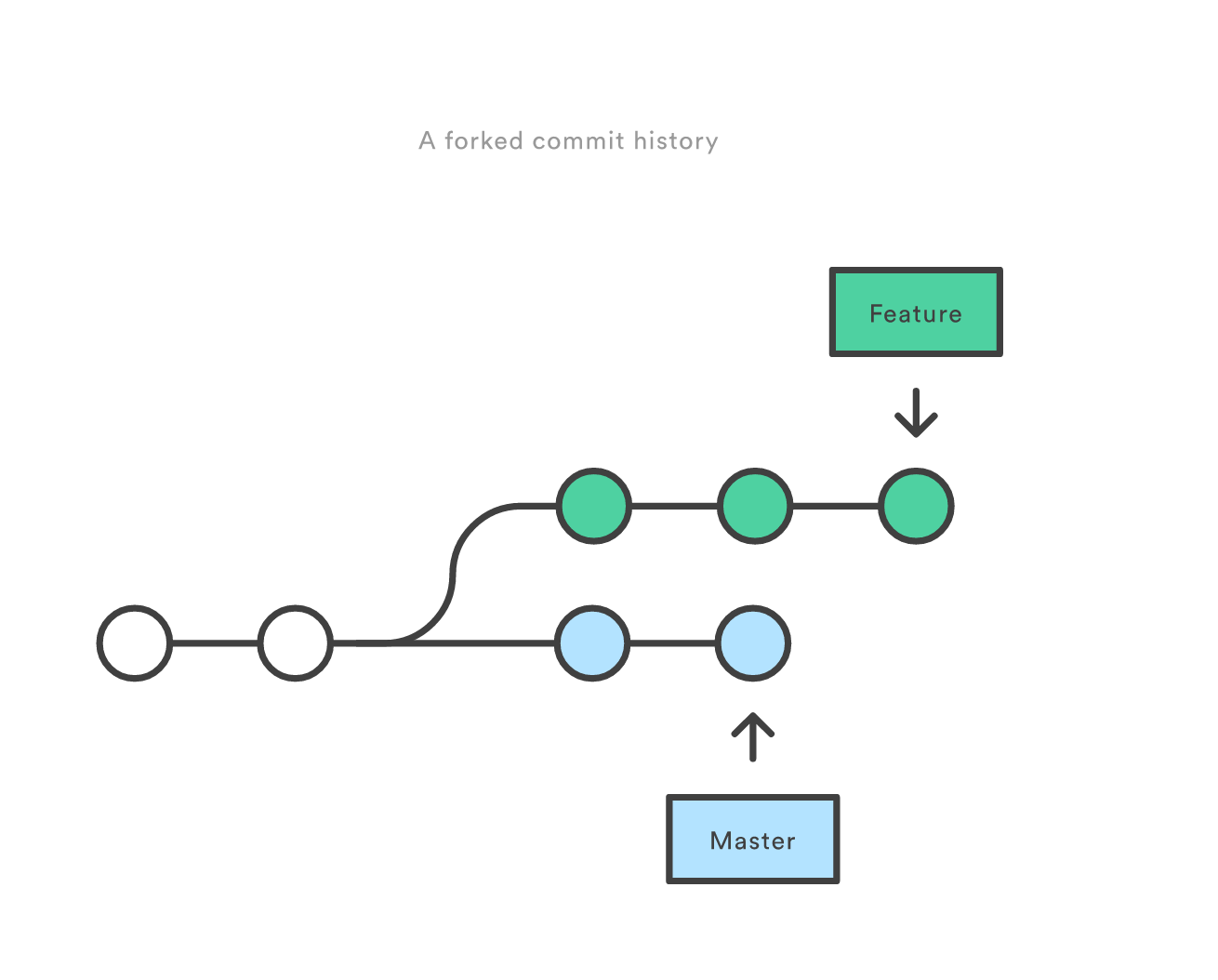
git branch //查看本地分支

git branch -a //查看本地和远端分支

git branch -r //查看远端分支

**git rebase 和 git merge 的区别：**

在一个feature分支进行新特性的开发，与此同时，master 分支的也有新的提交。



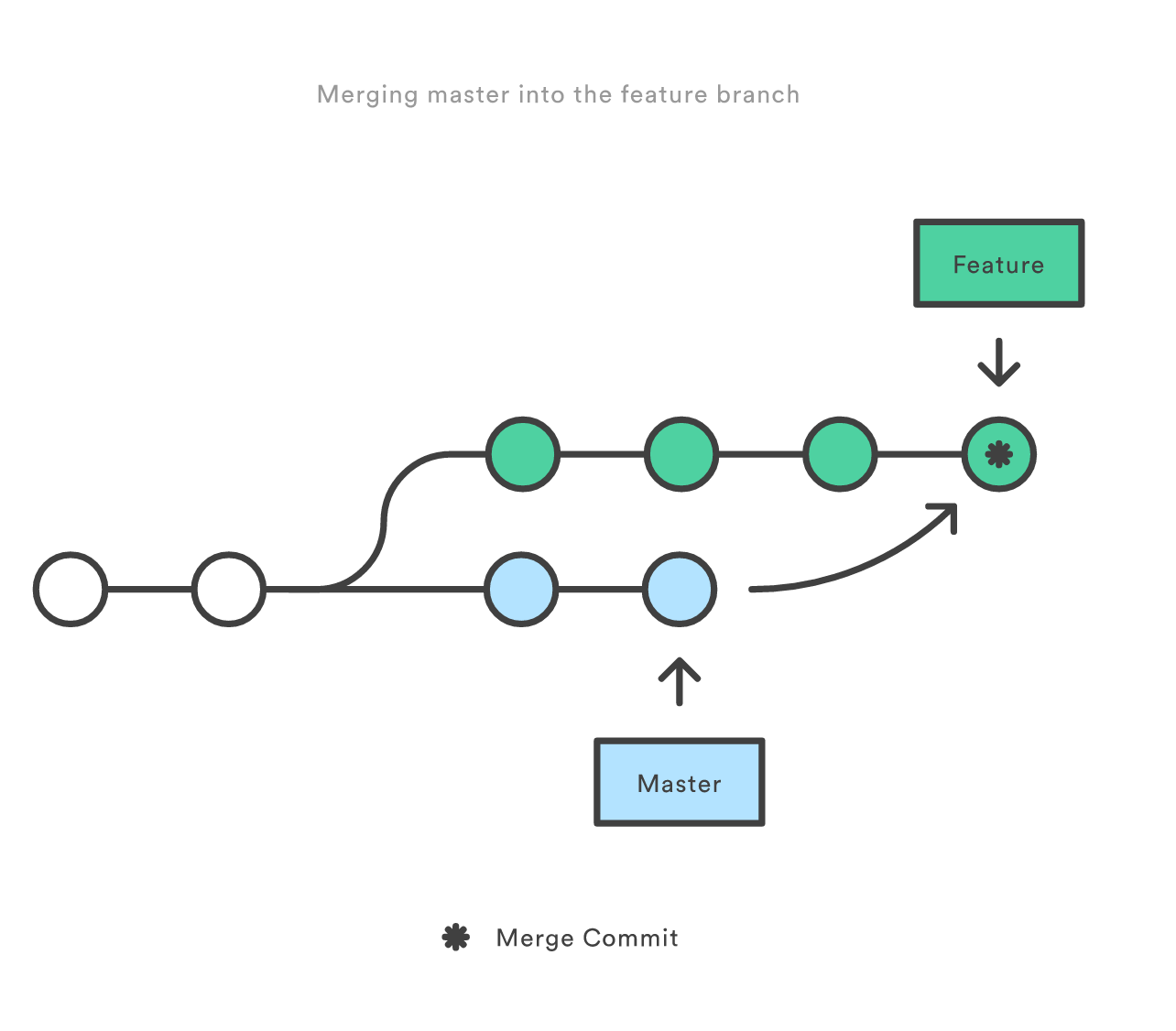
为了将master 上新的提交合并到你的feature分支上，有两种选择：merging or rebasing：

merge:

git checkout feature

git merge master

此时在feature上git 自动会产生一个新的commit(merge commit):



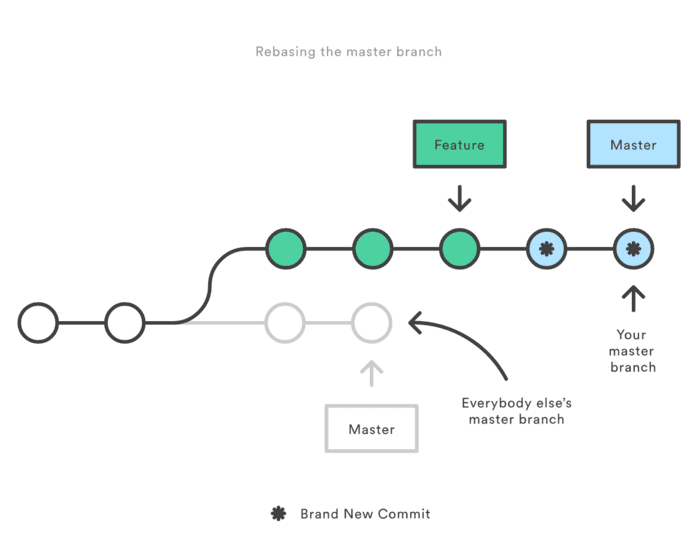
merge 特点：自动创建一个新的commit  
如果合并的时候遇到冲突，仅需要修改后重新commit  
优点：记录了真实的commit情况，包括每个分支的详情  
缺点：因为每次merge会自动产生一个merge commit，所以在使用一些git 的GUI tools，特别是commit比较频繁时，看到分支很杂乱。

rebase:

rebase本质是变基。

git checkout feature

git rebase master



rebase 将所有master的commit移动到你的feature 的顶端。

合并时如果出现冲突需要按照如下步骤解决：

1. 修改冲突部分
2. git add 不要在git add 之后习惯性的执行 git commit命令
3. git rebase --continue

如果第三步无效可以执行 git rebase --skip

**git add详解：**

-u, --update update tracked files. 已经被tracked的文件的所有修改加入暂存区。

-A, --all add changes from all tracked and untracked files

--ignore-removal 添加新文件和被修改的文件，忽略被删除的文件。

**为本地的分支设置远端分支：**

git branch --set-upstream <local branch> <remote branch>

创建本地分支的同时指定对应的远端分支：

git checkout -b <local branch> <remote branch>

第一次向远端推送时指定对应的远端branch：

git push –u/--set-upstream origin <remote branch>

**修改上一次提交的信息：**

git commit –amend

**比较两个分支：**

git diff <branch a> <branch b>

或：

git diff <branch a>..<branch b>

比较两个分支之间某个文件的不同：

git diff <branch a>..<branch b> -- 文件名

**git fetch与git pull的区别：**

**FETCH\_HEAD**： 是一个版本链接，记录在本地的一个文件.git/FETCH\_HEAD中，指向着目前已经从远程仓库取下来的分支的最新commit。

**git fetch有四种基本用法:**

1. git fetch 这将更新git remote中所有的远程repo 所包含的所有分支的最新commit-id, 将其记录到.git/FETCH\_HEAD文件中

2. git fetch remote\_repo 这将更新名称为remote\_repo 的远程repo上的所有branch的最新commit-id，将其记录。

3. git fetch remote\_repo remote\_branch 这将这将更新名称为remote\_repo 的远程repo上的分支： remote\_branch

4. git fetch remote\_repo remote\_branch:local\_branch 这将这将更新名称为remote\_repo 的远程repo上的分支： remote\_branch，并在本地创建local\_branch本地分支保存远端分支的所有数据。

**用fetch合并将远端master分支的改动合并到本地dev分支的流程：**

git fetch origin master:temp //在本地新建一个temp分支，并将远程origin仓库的master分支代码下载到本地temp分支

git checkout dev

git merge temp

git branch –d temp

git pull**的工作流程:**

首先，基于本地的FETCH\_HEAD记录，比对本地的FETCH\_HEAD记录与远程仓库的版本号，然后git fetch 获得当前指向的远程分支的后续版本的数据，然后再利用git merge将其与本地的当前分支合并。所以可以认为git pull是git fetch和git merge两个步骤的结合。

小结：

与git pull相比git fetch相当于是从远程获取最新版本到本地，但不会自动merge。如果需要有选择的合并git fetch是更好的选择。否则git pull将更为快捷。