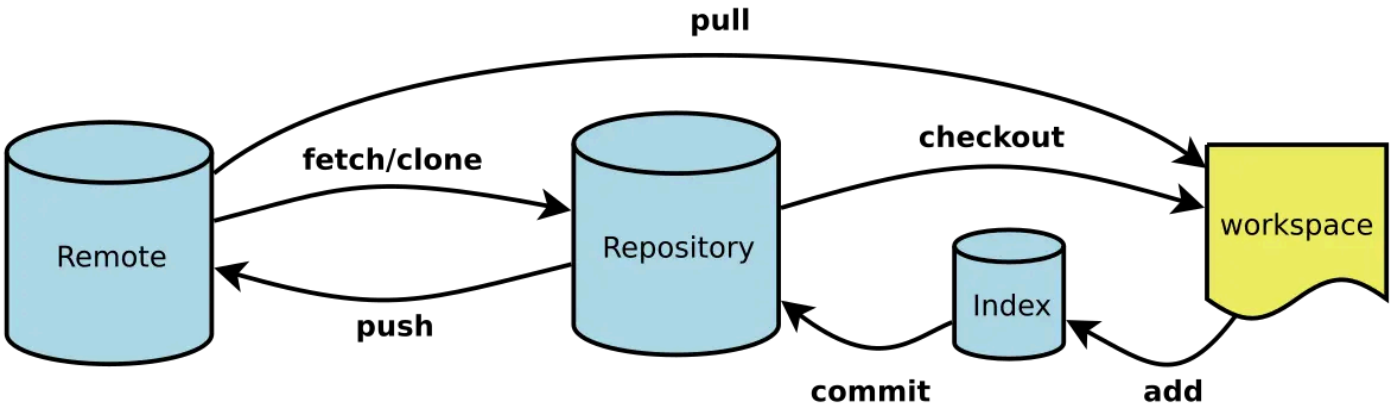


git 两大功能：版本管理/ 分布式协作

版本控制

git 原理

- git 首先指定一个目录作为工作区
- git 会自动在工作区生成一个.git 文件夹，作为版本库。
  - 存储版本变化信息。
- 同时，.git 中会有一个 index 文件夹，作为暂存区
  - 做的所有改动都会被自动记录，此时 git add 会储存到暂存区中

git 仓库的 SSH 传输

git 默认支持两种传输协议：`SSH`&`HTTPS`。

基本指令

从远程 cLone 一个仓库 `git clone <remote url>`

显示当前工作目录下的提交文件状态 `git status`

将指定文件 stage(标记为将要被提交的文件) `git add <path>`

将指定文件 unstage（取消将被标记的文件） `git reset <path>`

创建要给提交并提供提交信息 `git commit -m "提交信息"

显示提交历史 `git log`

向远程仓库推送 `git push`

从远程仓库拉取 `git pull`

进阶

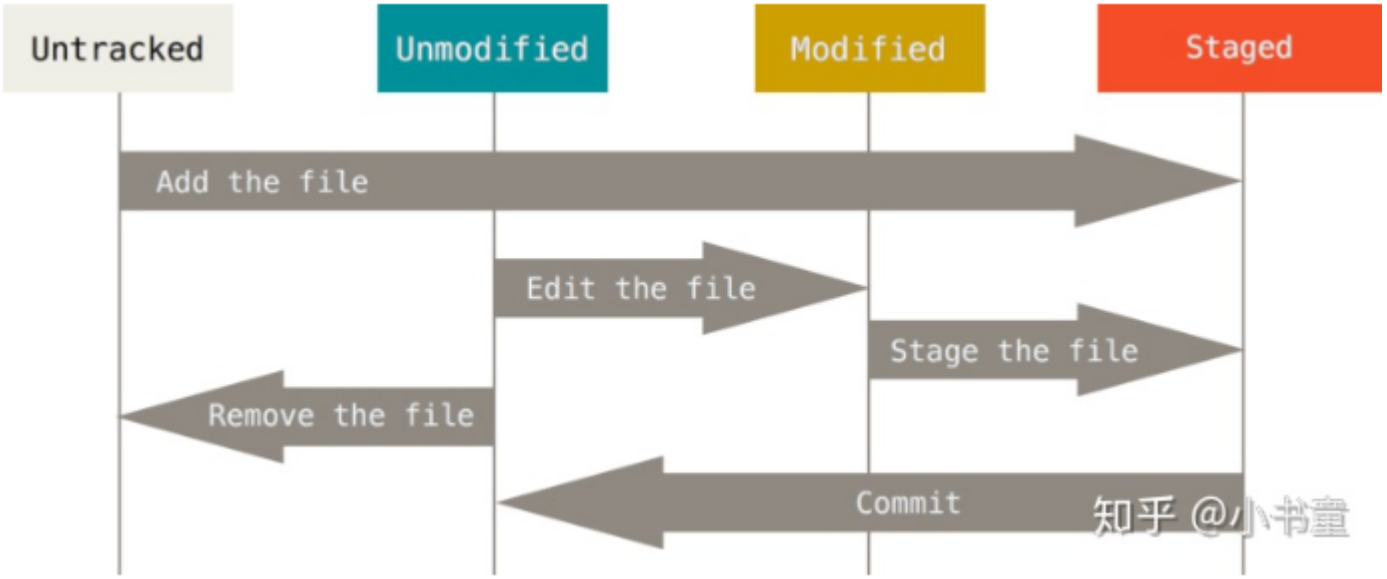
改写{amend} 对已提交的信息进行修改 （最好对本地提交使用）

提交代码在默认 master 上，创建分支

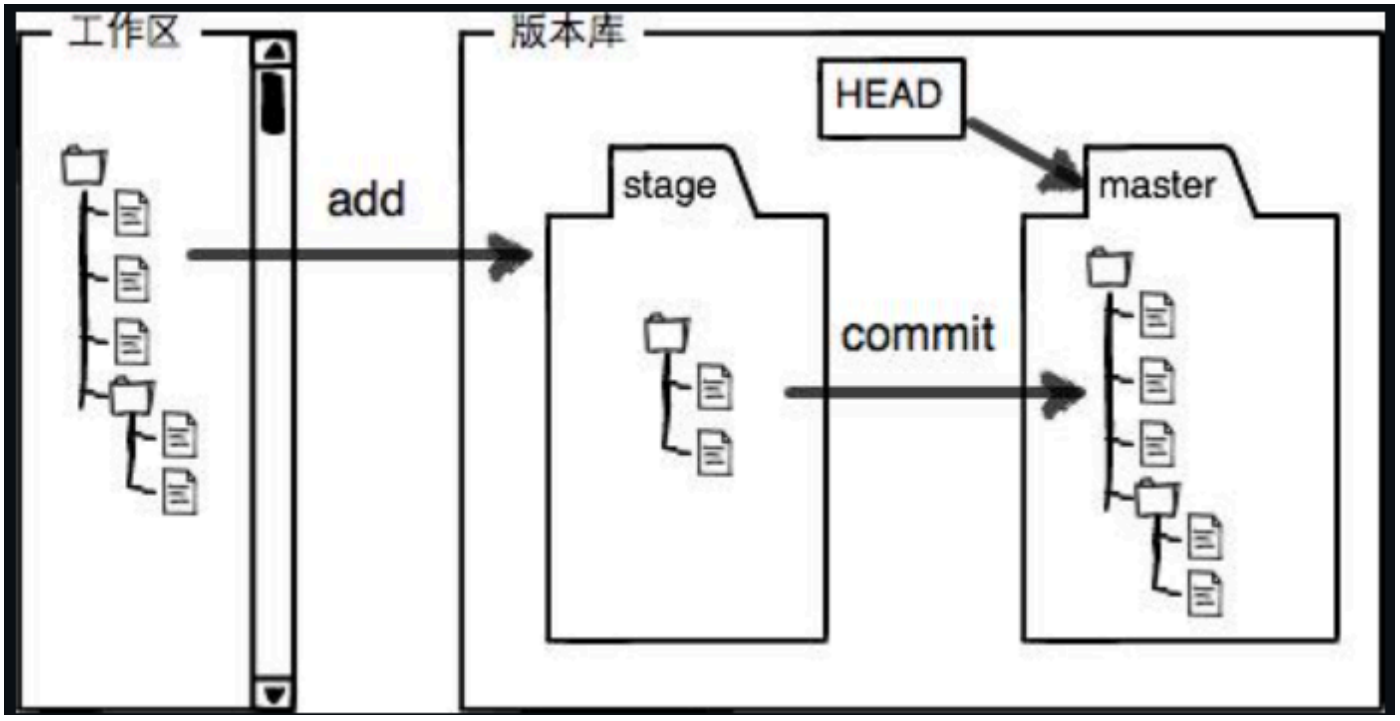
分支合并（merge） 若多人修改同一行代码，会冲突 conflict，compare 修改完后。

git 文件四种状态

untracked unmodified modified staged



暂存区 stage / index:



git checkout --file - 丢弃工作区的修改

场景：乱改了工作区某个文件的内容，想直接丢弃工作区的修改时.

自然，你是不会犯错的。不过现在是凌晨两点，你正在赶一份工作报告，你在`readme.txt`中添加了一行：

```
$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
My stupid boss still prefers SVN.
```

在你准备提交前，一杯咖啡起了作用，你猛然发现了`stupid boss`可能会让你丢掉这个月的奖金！

既然错误发现得很及时，就可以很容易地纠正它。你可以删掉最后一行，手动把文件恢复到上一个版本的状态。如果用`git status`查看一下：

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

你可以发现，Git会告诉你，`git checkout -- file`可以丢弃工作区的修改：

```
$ git checkout -- readme.txt
```

命令`git checkout -- readme.txt`意思就是，把`readme.txt`文件在工作区的修改全部撤销，这里有两种情况：

一种是`readme.txt`自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是`readme.txt`已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

总之，就是让这个文件回到最近一次`git commit`或`git add`时的状态。

git reset HEAD <file> - 撤销掉（unstage）暂存区的修改，重新放回工作区

场景：不仅改了工作区某文件内容，还添加到暂存区时，想丢弃修改，两步：

- git reset HEAD <file>
- git checkout --file

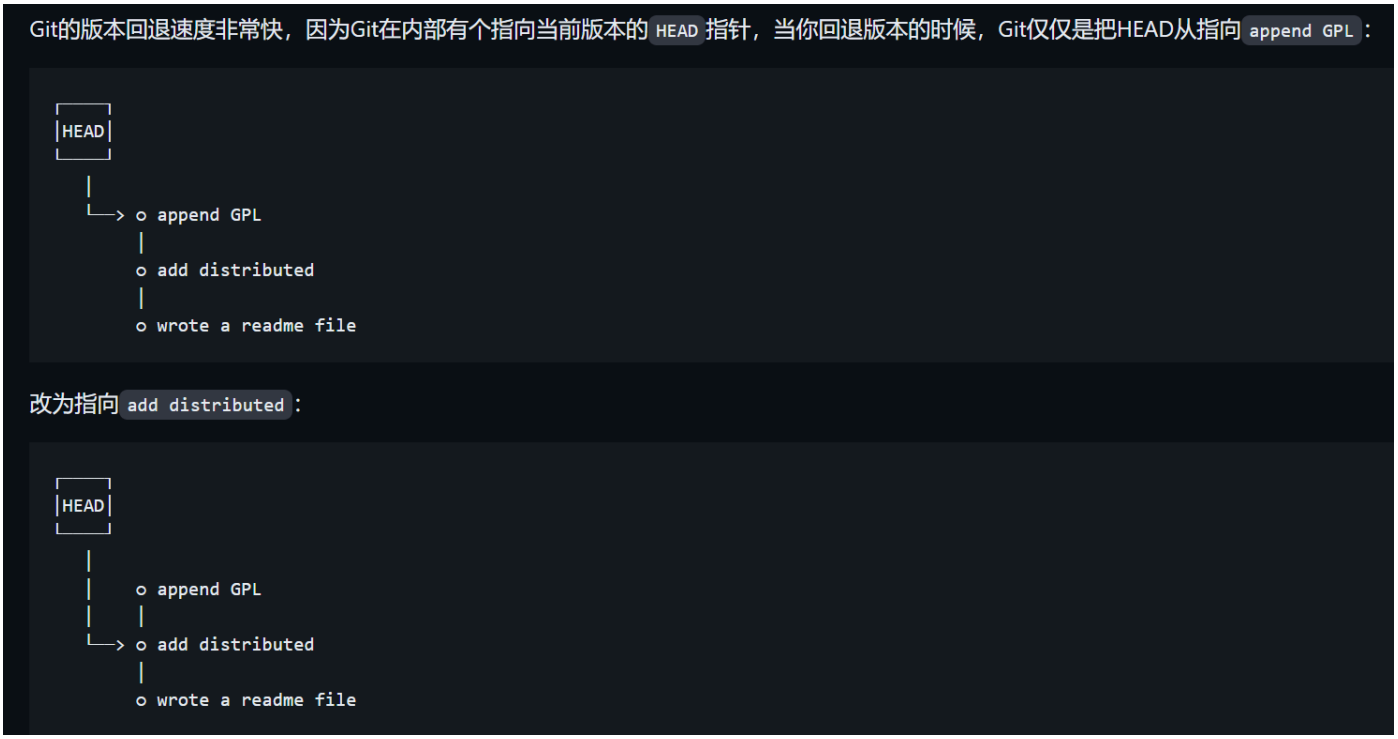
reset - 回退到上一个 commit 版本

场景：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

```
git reset --hard xx
```

回滚原理：

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的`HEAD`指针，当你回退版本的时候，Git仅仅是把HEAD从指向`append GPL`：



改为指向`add distributed`：

git reflog - 查看命令历史，确定要回到未来的哪个版本。

删除文件 两种情况——确实要删除和误删

- 确定删除 `git rm`
- 误删，想把误删文件恢复到最新版本。

```
git checkout -- file
```

git checkout 原理：用版本库里的版本替换工作区的版本

fetch

直接用 pull 会直接将远程仓库项目覆盖到工作区,所以应先用 fetch 将远程仓库项目添加到 repo,然后用 diff 对比区别，无问题再合并。

不需要提交（忽略的文件）

git 中有两个人同时对一个文件进行修改，需要进行合并操作（merge）

- git pull

项目中 git 提交

git > 提交 > 勾选更改 > 提交信息 > 提交（不选提交并推送）> git > 拉取 > 合并冲突

不提交直接拉取：

git>更新分支 > 合并到 > 拉取

未进行版本管理的文件。

连接 github

本地 git 仓库和 github 仓库之间的传输是通过 SSH 加密的。

分支管理

创建分支和合并分支

每次提交，git 将他们串成一条时间线，此时间线就是一个分支。

Git鼓励大量使用分支：

查看分支：`git branch`

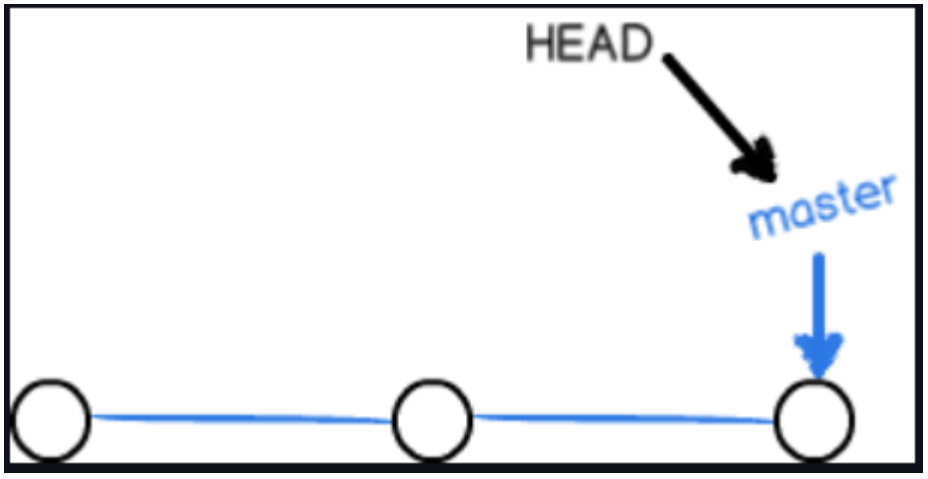
创建分支：`git branch <name>`

切换分支：`git checkout <name>` 或者 `git switch <name>`

创建+切换分支：`git checkout -b <name>` 或者 `git switch -c <name>`

合并某分支到当前分支：`git merge <name>`

删除分支：`git branch -d <name>`



- 创建 dev 分支，然后切换到 dev 分支:

```
git checkout -b dev
Switched to a new branch 'dev'
```

- 在 dev 分支上正常提交，比如对 readme.txt 做个修改，然后提交

```
git add readme.txt
git commit -m 'branch test'
```

- 现在 dev 分支工作完成，可以切换回 master 分支：

```
git checkout master
```

- 把 dev 分支的工作成果合并到 master 分支上：

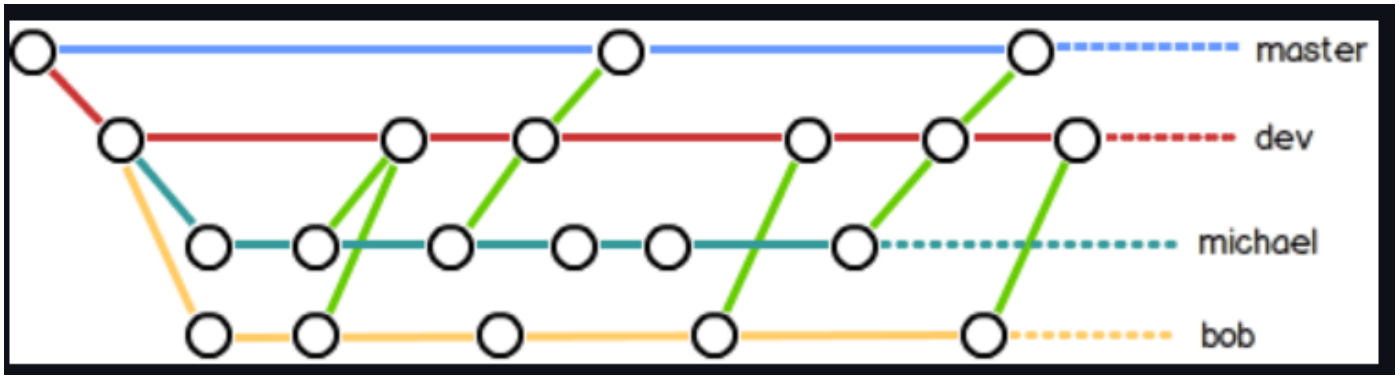
```
git merge dev
```

- 合并后，就可以放心删除 dev 分支了：

```
git branch -d dev
//-d delete
```

分支策略

master-仅用来发布新版本，平时不在上面干活； dev-干活分支； 团队合作分支
belike:



BUG 分支-git stash - 通过一个新的临时分支来修复，修复后，合并分支，然后删除临时分支

git stash-

cherry-pick - 复制一个特定的提交到当前分支

在master分支上修复了bug后，我们要想一想，dev分支是早期从master分支分出来的，所以，这个bug其实在当前dev分支上也存在。

那怎么在dev分支上修复同样的bug？重复操作一次，提交不就行了？

有木有更简单的方法？

有！

同样的bug，要在dev上修复，我们只需要把`4c805e2 fix bug 101`这个提交所做的修改“复制”到dev分支。注意：我们只想复制`4c805e2 fix bug 101`这个提交所做的修改，并不是把整个master分支merge过来。

为了方便操作，Git专门提供了一个`cherry-pick`命令，让我们能复制一个特定的提交到当前分支：

```
$ git branch
* dev
  master
$ git cherry-pick 4c805e2
[master 1d4b803] fix bug 101
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Git自动给dev分支做了一次提交，注意这次提交的commit是`1d4b803`，它并不同于master的`4c805e2`，因为这两个commit只是改动相同，但确实是两个不同的commit。用`git cherry-pick`，我们就不需要在dev分支上手动再把修bug的过程重复一遍。

有些聪明的童鞋会想了，既然可以在master分支上修复bug后，在dev分支上可以“重放”这个修复过程，那么直接在dev分支上修复bug，然后在master分支上“重放”行不行？当然可以，不过你仍然需要`git stash`命令保存现场，才能从dev分支切换到master分支。

多人协作

因此，多人协作的工作模式通常是这样：

1. 首先，可以试图用`git push origin <branch-name>`推送自己的修改；
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用`git pull`试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用`git push origin <branch-name>`推送就能成功！

如果`git pull`提示`no tracking information`，则说明本地分支和远程分支的链接关系没有创建，用命令`git branch --set-upstream-to <branch-name> origin/<branch-name>`。

这就是多人协作的工作模式，一旦熟悉了，就非常简单。

rebase