安装完git

$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"

git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

成功后

第一步：创建一个目录。

第二步：通过git init命令把这个目录变成Git可以管理的仓库。

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

创建一个readme.txt到learngit目录

第一步：用命令git add告诉git，把文件提交到仓库。

$git add readme.txt

第二步：用命令git commit告诉git，把文件提交到仓库。

$git commit -m "wrote a readme file" (-m后面输入的是本次提交的说明，可以使任意内容，最好有意义。)

小结

现在总结一下今天学的两点内容：

初始化一个Git仓库，使用git init命令。

添加文件到Git仓库，分两步：

使用命令git add <file>，注意，可反复多次使用，添加多个文件；

使用命令git commit -m <message>，完成。

https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013743858312764dca7ad6d0754f76aa562e3789478044000

--2019年4月17

更改readme.txt的内容后

使用命令git status查看仓库当前状态，如果发现没有提交，被修改了。

使用命令gir diff readme.txt查看修改的内容。

然后gir add readme.txt，再查看一下仓库状态。会提示将要被提交的修改包括readme.txt。

然后就可以安心的提交了，git commit -m “add distributed”

再查看下仓库状态。

小结

要随时掌握工作区的状态，使用git status命令。

如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

练习以上动作。至今已经有三个版本。

使用git log查看最近提交的日志，

如果感觉多，可以使用：git log--pretty=oneline

准备回退

在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

$git reset --hard HEAD^

使用git log后发现最后一次提交的没有了。

没关系，可以指定回到未来的某个版本：$git reset --hard commit\_id

$git reset --hard 2bd6e

Git提供了一个命令git reflog用来记录你的每一次命令

小结

现在总结一下：

HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。

穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

工作区和暂存区

工作区：learngit文件夹就是一个工作区。

版本库：learngit文件夹中的一个隐藏目录.git就是版本库。

暂存区：版本库中最重要的stage(或者叫index)，还有为我们自动创建的第一个分支master，以及指向master的一个指针HEAD。

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。

管理修改： 每次修改完，都要把为文件add到暂存区中，最后一起commit。

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。

针对第二种情况：使用命令git reset HEAD <file> 就可以清掉提交到暂存区的文件。

命令中 -- 很重要，没有-- 就变成了切换到另一个分支的命令。

PS：在使用git status命令行后就会出现下一步可以使用的下一步指令(包括提交或者回退)。

小结

又到了小结时间。

场景1：当你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。

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。

场景3：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。

删除文件：

小提示：先手动删除文件，然后使用git rm <file>和git add<file>效果是一样的。

小结

命令git rm用于删除一个文件。如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失最近一次提交后你修改的内容。

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

远程仓库

申请GitHub账号，这样就可以免费使用一个远程的github仓库和你的仓库之间通过ssh加密，需要设置。设置详情参考：

https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001374385852170d9c7adf13c30429b9660d0eb689dd43a000

第一步使用命令：

$ssh-keygen -t rsa -C "youremail@example.com"

创建SSH Key。

第2步：登录GitHub，打开Account settings”，“SSH Keys”页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：

点“Add Key”，你就应该看到已经添加的Key。

这个仓库是免费托管的，可读不可写，不要放敏感信息。也可以缴费使用加密不可读不可写。

小结

要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；

分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

从远程库克隆

创建一个新的git库：gitskills

然后使用命令：git clone git@github.com:Ezio521/gitskills.git

git支持多协议的，默认是git://使用ssh，也可以使用https等其他协议。(HTTPS速度慢而且每次推送都必须使用命令)

小结

要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

分支管理

可以创建自己的分支，然后在自己的分支上修改，不影响其他人工作。

SVN也有自己的分支管理，只是很慢。

创建与合并分支

创建新的分支，然后切换到Dev分支：

$git checkout -b dev

git checkout 命令加上 -b 参数表示创建并切换，相当于以下两条命令：

$git branch dev

$git checkout dev

然后用git branch命令查看当前分支：

分支前带\*，表示为当前分支

此时对文件作出修改然后提交：$git add 文件名 $git commit -m "备注"

分支工作完成后切换到master分支：$git checkout master

此时修改的内容并没有在master中体现，修改内容还在dev分支中，

需要把dev分支合并到master分支上:

$git merge dev

git merge命令用于合并指定分支到当前分支。

此时可以删除分支dev

$git branch -d dev

小结

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>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

解决冲突

新建的分区与自有的master分区做出了不同的改变。合并的时候就会出现冲突，

冲突只能自己在master主线分支下修改之后再提交就可以了（必须要重新修改）

查看冲突日志：

git log --graph --pretty=oneline --abbrev-commit

小结

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。

用git log --graph命令可以看到分支合并图。

分支管理策略

Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

实战--no-ff方式的git merge

创建分支---修改文件---add-commit---切换到master分支

使用：git merge --no-ff -m "merge with no-ff" dev ---表示禁用Fast forward

因为本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去。

查看分支历史：git log --graph --pretty=oneline --abbrev-commit

以上分支管理策略就是不使用Fast forward模式

分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



小结

Git分支十分强大，在团队开发中应该充分应用。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

**Bug分支**

假如你在dev分支上工作一半的时候，还不能提交。紧急情况下如要修复bug，此时可以使用stash功能，把当前的工作状态隐藏起来，等以后恢复现场后继续工作：

$git stash

先确定是哪个分支需要修复bug，假定是master，就在master上创建分支：

$git checkout master

$git checkout –b issue-101

修复bug，然后提交。

修复完成后，切换到master分支，完成合并，然后删除issue-101分支

$ git merge --no-ff -m "merged bug fix 101" issue-101

完成修复后，返回dev分区工作，但是此时：git status，发现工作区是干净的。

使用：$git stash list 来查看

Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

另一种方式是用：

$git stash pop

此时再使用 $ git stash list 就看不到内容了

可以多次使用stash，恢复的时候，先用git stash list 查看，然后恢复指定的stash，用命令：

$git stash apply stash@{0}

小结

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场。

**Feature分支**

在新的分支上开发一个功能

$git checkout –b feature-vulcan

开发完成后

$git add vulcan.py

提交，切换到dev，准备合并

但是，，，

此时这个功能不需要上线了，需要删掉,删掉整个分支

$git branch –d feature-vulcan

但是还会提示feature-vulcan分支还没有被合并，如果删除，将丢失掉修改，如果要强行删除，需要使用大写的-D参数

$git branch –D feature-vulcan

小结

开发一个新feature，最好新建一个分支；

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

**多人协作**

查看远程库的信息，用：

$git remote

或者，使用

$git remote –v

$ git remote -v

origin git@github.com:Ezio521/learngit.git (fetch)

origin git@github.com:Ezio521/learngit.git (push)

上面显示了可以抓取和推送的origin的地址。如果没有推送权限，就看不到push的地址。

### 推送分支

推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上：

可以是：

$git push origin master

如果要推送其它分支

$git push origin dev

并不是所有的分支都远程推送的，那么那些分支需要推送呢？

。master分支是主分支，因此要时刻与远程同步；

。dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

。bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

。feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，在git中分支完全可以自己藏着玩。

### 抓取分支

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。

### 小结

* 查看远程库信息，使用git remote -v；
* 本地新建的分支如果不推送到远程，对其他人就是不可见的；
* 从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；
* 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；
* 建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；
* 从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。