本来是打算用md文件写的，但是就是好久没写了，懒得去弄练和熟悉，后面有时间再采用，现在就还是用word

# 0 之前写了点“关于Git和Github新知”，主要就是关于怎么进行一个版本的tag标记，迁移过来

[github创建tag](https://www.cnblogs.com/exmyth/p/5136904.html) --在github端好像要手动操作，不过很简单的（基本上链接的所有命令都很好）

查看tag 创建tag 删除tag 共享tag

[【Git】checkout 用法总结](https://www.jianshu.com/p/cad4d2ec4da5)—checkout的命令在处理分支，这也是我所未接触的

# 1 下面内容是工作实际上会用到的常用git命令的学习

主要场景是来自于这个链接：[JAVA实习生一般进去做什么呢？ - Java3y的回答 - 知乎](https://www.zhihu.com/question/397769391/answer/1258264668) 里面讲解了实习生最初进公司是什么场景

具体也不引用，直接看帖子上就有

* 在视频练的项目大多数都是由讲师在**本地**编码来讲解整一个项目的开发过程，**而我们去到公司做的第一件事是啥？把项目clone(checkout)到本地来看。**

**--这里说的clone不难理解，checkout应该是就clone之后需要切换分支例如到dev**

* 比如说，我们使用Git的时候，**要修改代码的时候会新建一个分支**，**改完了再合并到master分支上**

**--这里就是基础的分支操作**

* 那问题来了，我写完的代码怎么**调试**啊？我们可以这样干的：**将本地写好的代码push到测试环境**，然后本地**远程连接**测试环境，对其进行调试。

**--这个就有些难度，需要具体去公司听和问，可能不同公司不同**

* 在公司一般将错误的信息（或者有用的信息）写到log（**日志**）中。

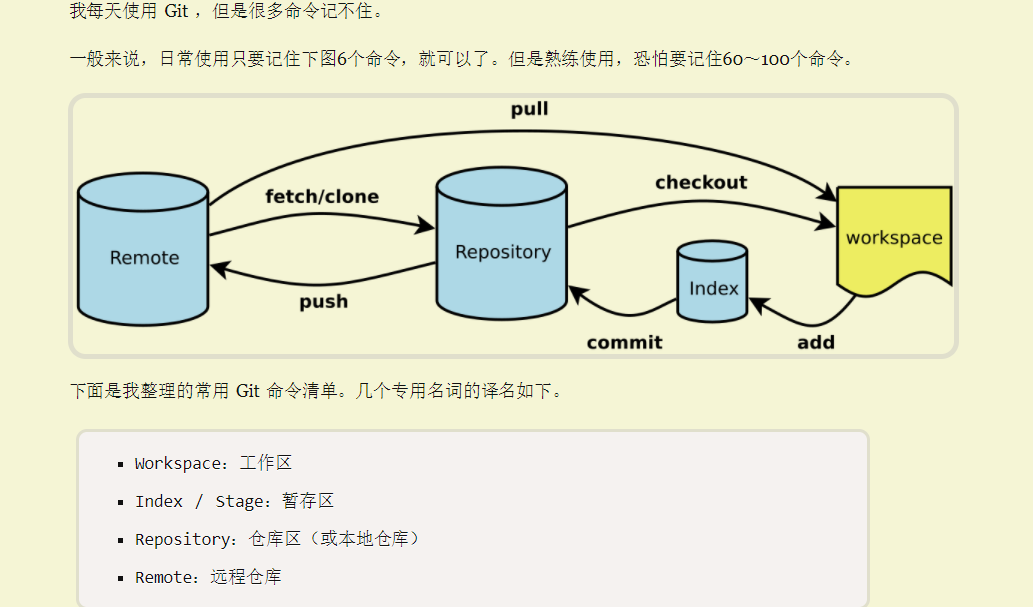
比如说：LOG.error("send java3y article failed, {}", e);

--日志其实也不难，熟悉一下怎么操作就好了，珍惜新手期，能尽量学多好学多好，那是你后面腾飞的优势

总结：

1. 版本控制工具
2. 远程连接Debug调试
3. 不再使用e.printStackTrace();，而是log.error()来替代
4. 各种权限都需要申请和审批

## 实际工作会用到的常用git命令（正文）



（引自[阮一峰](http://www.ruanyifeng.com/)[常用 Git 命令清单](https://www.ruanyifeng.com/blog/2015/12/git-cheat-sheet.html)）PS：repository也叫版本库

这个图是挺重要，并且上面的名词是要求记住的。它能让你基本上理解常用命令的意义

也不难，就是add之后是到一个叫做暂存区的地方去，commit之后就提交到本地仓库了

### 参考资料

[Git常用操作指南 - 郭耀华 - 博客园](https://www.cnblogs.com/guoyaohua/p/Git-tutorial.html) 又有实例又挺全的

[常用 Git 命令清单 - 阮一峰的网络日志](https://www.ruanyifeng.com/blog/2015/12/git-cheat-sheet.html) 大佬很权威，但是就是没有实例并且一些名字让人费解，并且太多只采取常用的下来即可（可以做类似手册的作用）

[高频使用的 Git 命令 - 云+社区 - 腾讯云](https://cloud.tencent.com/developer/article/1551458) 后面get的粗看好链接，就是那种突出重点的感觉

**次级参考：**

[我在实际工作中用的最多的 git 命令，全在这里了- 掘金](https://juejin.im/post/6844904174027538439) 有实例但感觉不太全又有点浅

[git在工作中的一些用处 - 个人文章 - SegmentFault 思否](https://segmentfault.com/a/1190000017232188) 有一个工作中小红小明展示git的场景

[GIT 命令“从初学到专业”完整进阶指南](https://gist.github.com/285571052/72fe4e85290d170b9de4634b6ad8c082) 界面好看且好像挺全的，不过没什么热度，参考用

[Git 分支简介、Git 和 GitHub 日常操作 – IBM Developer](https://developer.ibm.com/zh/technologies/devops/articles/os-cn-git-and-github-3/) 会展示github端的操作

### 所有之前md文件中的[基本命令](https://github.com/theoneofc/git-github-/blob/master/git%20%E5%92%8C%20github%E5%85%A5%E9%97%A8.md)

下面特别补充：

# 添加指定文件到暂存区

$ git add [file1] [file2] ...

# 添加指定目录到暂存区，包括子目录

$ git add [dir]

# 提交暂存区到仓库区

$ git commit -m [message]

# 提交暂存区的指定文件到仓库区

$ git commit [file1] [file2] ... -m [message]

# 提交时显示所有diff信息

$ git commit -v

~~git commit -a -m ‘提交信息’（ 不太建议使用，还是老老实实分两步好 ，不然出错很麻烦 ）~~

~~这个命令将上面两个步骤 add . 和 commit 合二为一（也就是添加所有变动并提交）~~

[git commit 详解](https://www.cnblogs.com/wangrenmeng/p/10430369.html) 里面有说进了vim模式原因和怎么弄，其实就是简单linux命令操作

### 日志查看/信息显示

#### git log

这个命令主要用于查看提交日志

# 输出概要日志,这条命令等同于

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

git log –oneline #等效于上一条长的

--阮一峰所说的某次commit是什么啊，commit还有commit名的吗？我怎么不知道，这有些冷僻了吧（选知）

--就是commit号，也就是版本号、commit id，可以通过**git log --pretty=oneline --abbrev-commit（等效于git log –online ）** 或者**git reflog** 查看（前者细后者略）

#### git status【高频使用】

可以用来查看仓库的状态，在开发中，使用最多的可能就是这个命令了，建议开发过程中没事就 git status 一下。

如果当你不知道你的git分支或者仓库的状态的时候，记得一定git status一下，不然，可能就会出问题哈。

### 分支管理【重头戏】branch 和 checkout

这个是重头戏哈，在实际的工作中，**分支创建开发新功能，切换分支简直能再多了**，如果操作不当，可能造成很大的麻烦，我在工作中就遇到过很多不必要的麻烦，而且解决起来特别难受。

#### 本地

##### 创建分支进行内容修改

首先，我们创建dev分支，然后切换到dev分支（一般直接切换过去比较多，因为建立分支就是为了修改）：

**$ 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：

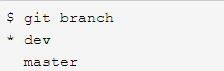
**$ git branch dev # 创建dev分支，但依然停留在当前分支**

**$ git checkout dev # 切换到dev分支**

##### 查看分支

# 列出所有本地分支

$ git branch





# 列出所有远程分支

$ git branch -r

# 列出所有本地分支和远程分支

$ git branch -a

##### 返回master并且合并分支

当dev分支的工作完成，我们就可以切换回master分支：

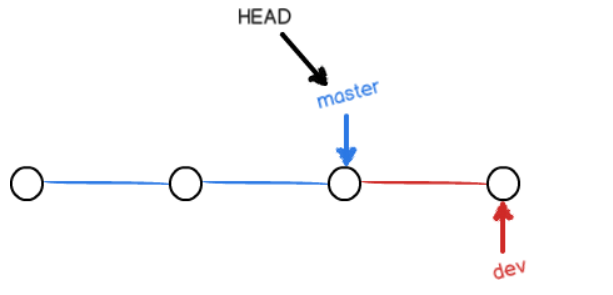
$ git checkout master

注意：必须得先add和commit之后再切换分支，否则修改会被同步到其他分支

现在，我们把dev分支的工作成果合并到master分支上：

$ git merge dev

切换回master分支后，我们查看文件内容时会发现所有修改的内容都不见了！因为那个提交是在dev分支上，而master分支此刻的提交点并没有变（示意图如下）：



git merge命令用于合并指定分支到当前分支。合并后我们就可以看到文件内容和dev分支的最新提交是完全一样的。

##### 合并完成后，就可以放心地删除dev分支了：（在master下删，在dev下删不成功也很奇怪哈哈哈）

$ git branch -d dev

删除后，查看branch，就只剩下master分支了：

$ git branch

\* master

##### [解决冲突](https://www.cnblogs.com/guoyaohua/p/Git-tutorial.html#%E5%88%86%E6%94%AF%E7%AE%A1%E7%90%86%E7%AD%96%E7%95%A5)

可以戳上面的链接看，只是里面说的冲突是有意设计的，其实是可以规避的，所以冲突大多是因为无意不小心形成的吗？而且它的处理也看着有些懵

**后面有碰到冲突情况再搜索学习和解决把**

##### 因为创建、合并和删除分支非常快，所以Git鼓励你使用分支完成某个任务，合并后再删掉分支，这和直接在master分支上工作效果是一样的，但过程更安全。

##### 分支策略



#### 远程

##### 推送分支（将分支上传也是这个命令）

**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：

$ git push origin master

**如果要推送其他分支，比如dev，就改成：**（将分支上传也是这个命令，例如这里就可以把本地有而远程没有的分支上传）

**$ git push origin dev**

但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* 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，就是在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！

##### 指定本地dev分支与远程origin/dev分支的链接

git push --set-upstream origin dev 或者

git branch --set-upstream-to=origin/dev dev

The current branch dev has no upstream branch.

**To push the current branch and set the remote as upstream**, use

git push --set-upstream origin dev

If you wish to set tracking information for this branch you can do so with: git branch --set-upstream-to=origin/<branch> dev

##### GitHub--创建新的分支（内容已补充到上面去了，这里就是参考链接）

[GitHub--创建新的分支(转) - 前端邢晋宇 - 博客园](https://www.cnblogs.com/autoXingJY/p/9004724.html)

[GitHub创建分支两种方式\_今天又是充满希望的一天-CSDN博客](https://blog.csdn.net/HcJsJqJSSM/article/details/84558229)

### [标签 tag](https://www.cnblogs.com/guoyaohua/p/Git-tutorial.html#%E6%A0%87%E7%AD%BE%E7%AE%A1%E7%90%86) （进入链接看详细内容）

发布一个版本时，我们通常先在版本库中打一个标签（tag），这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以，标签也是版本库的一个快照。

#### 对分支的某当前commit打tag

首先，切换到需要打标签的分支上：

$ git branch

\* dev

master

$ git checkout master

敲命令git tag <name>就可以打一个新标签：

$ git tag v1.0

可以用命令git tag查看所有标签：

$ git tag

v1.0

#### 对历史commit打tag

默认标签是打在最新提交的commit上的。有时候，如果忘了打标签，比如，现在已经是周五了，但应该在周一打的标签没有打，怎么办？

方法是找到历史提交的commit id，然后打上就可以了：

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

比方说要对add merge这次提交打标签，它对应的commit id是f52c633，敲入命令：

$ git tag v0.9 f52c633

再用命令git tag查看标签：

$ git tag

v0.9

v1.0

#### 操作标签

注意，标签不是按时间顺序列出，而是按字母排序的。可以用git show <tagname>查看标签信息：

$ git show v0.9

还可以创建带有说明的标签，用-a指定标签名，-m指定说明文字：

$ git tag -a v0.1 -m "version 0.1 released" 1094adb

如果标签打错了，也可以删除：

$ git tag -d v0.1

因为创建的标签都只存储在本地，不会自动推送到远程。所以，打错的标签可以在本地安全删除。

#### 推送到远端

如果要推送某个标签到远程，使用命令git push origin <tagname>：

$ git push origin v1.0

或者，一次性推送全部尚未推送到远程的本地标签：

$ git push origin --tags

**删除已推送到远程的标签**

如果标签已经推送到远程，要删除远程标签就麻烦一点，先从本地删除：

$ git tag -d v0.9

然后，从远程删除。删除命令也是push，但是格式如下：

$ git push origin :refs/tags/v0.9

要看看是否真的从远程库删除了标签，可以登陆GitHub查看。

### 版本回退（部分内容在7. 其他操作中）

reset命令格式：

git reset [--hard|soft|mixed|merge|keep] [commit|HEAD]

1. git reset --hard HEAD^

HEAD^表示上一个版本，HEAD^^表示上上一个版本，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。

2. git reset --hard 版本号

--怎么找版本号

a. git log 可以

b. git reflog 找寻所有历史版本号，满足您的所有需求

Q：回退可以针对单一文件吗？还是对于整个版本

~~--不能，只能进行版本回退~~

~~所以这也是回退版本相对来说用的比较少的地方，这个是要大动手脚啊，不是微调~~

**【找到了，是可以的，之前是没找到】**

* 反悔单个文件的版本，这很可（但还是不要依赖，就是和版本回退一样特殊情况才用）

# 恢复**某个commit的指定文件**到暂存区和工作区

$ git checkout [commit] [file]

HEAD 说明：

* HEAD 表示当前版本
* HEAD^ 上一个版本
* HEAD^^ 上上一个版本
* 以此类推...

可以使用 ～数字表示

* HEAD~0 表示当前版本
* HEAD~1 上一个版本
* HEAD~2 上上一个版本
* 以此类推...

### 其他操作

#### 撤销修改—反悔工作区

**撤销修改特定文件**

-- 撤销修改 git checkout -- <file> （注意有空格）

git checkout [commit] [file] （命令等价）

当我们因为一些原因想要丢弃工作区某些文件修改时，可以使用“git checkout -- <file>”命令，该命令仅会恢复工作区文件状态，不会对版本库有任何改动。

命令git checkout -- file1.txt意思就是，把file1.txt文件在工作区的修改全部撤销，这里有两种情况：

* 一种是file1.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file1.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。

**也就是还没提交就去暂存区恢复，已经提交了暂存区就没东西了那当然是去版本库repository恢复**

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

-- 只回退add中的指定文件，但工作区不受影响，也就是纯反悔add，**可以和反悔工作区指定文件配合**

# **重置**暂存区的指定文件，与上一次commit保持一致，**但工作区不变**

$ git reset [file]

（其实是跟"git reset HEAD <file>...**”是等价的，就是省略了**HEAD）

那同样应该是可以

$ git reset [commit] [file]

**撤销修改所有文件**

# 恢复暂存区的所有文件到工作区

$ git checkout .

也是让文件（所有）回到**最近一次**git commit或git add时的状态

#### 撤销修改--如何反悔你的add和commit

快捷命令：

-**-撤销本次add**

~~$ git reset HEAD^ （错的）~~

$ git reset HEAD

**--撤销本次commit**

**git reset --hard HEAD^**

* 就是指定commit的版本回退了，就是**默认的--mixed模式**而已

# 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变

$ git reset [commit]

同样重置当前分支的指针为上一个版本，同时重置暂存区，但工作区不变命令：**【重要，撤销本次add】**

$ git reset HEAD^

补充知识：

**--mixed** 为默认，可以不用带该参数，用于重置暂存区的文件与上一次的提交(commit)保持一致，工作区文件内容保持不变

**--soft** 参数用于回退到某个版本：（但不会删除版本后的commit\_id吧。例如回退到第二次提交时通过git log 还能看到第三次的版本号）

**--hard** 参数撤销工作区中所有未提交的修改内容，将暂存区与工作区都回到上一次版本，并删除之前的所有信息提交：(与soft不同就是会删除版本后的commit\_id吧，需要用git reflog才能查到版本后的id)

资料：[重置命令](https://www.cnblogs.com/guoyaohua/p/Git-tutorial.html#%E9%87%8D%E7%BD%AE%E5%91%BD%E4%BB%A4)

[git reset --hard 操作后的数据恢复 - Chris-dc - 博客园](https://www.cnblogs.com/dongcanliang/p/11162235.html)

[git reset 命令 | 菜鸟教程](https://www.runoob.com/git/git-reset.html)

~~这就是版本回退了，但是一般要加HEAD的吧，毕竟指针，暂时不确定是相同命令还是不同（不重要，不用就好了，用有HEAD的）~~

~~# 重置暂存区与工作区，与上一次commit保持一致~~

~~$ git reset --hard~~

**建议改正命令为：git reset --hard HEAD^**

~~【错误认知】因为如果要撤销本次commit其实就是回到上上一次commit去，相信你懂我意思，所以命令为：~~

**~~git reset --hard HEAD^^ 【撤销本次commit】~~**

**撤销本次commit，因为已经commit了，所以HEAD指针就指向了本次commit上去的版本（HEAD版本），所以撤销本次commit也就是让HEAD指针指向上一个版本（HEAD^版本），所以撤销本次commit的命令为：**

**git reset --hard HEAD^ 【撤销本次commit】**

**那么问题来了git reset --hard HEAD 又是什么呢？这个好像咋一看会觉得是空命令一样，但是实际上不是的，所以HEAD的指向没有发生变化，但是它是会刷新暂存区和工作区的，所以是把暂存区和工作区变成最近一次commit的命令**

# 重置暂存区与工作区，与最近一次commit保持一致

**git reset --hard HEAD**

就是指定commit的版本回退了

# 重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

$ git reset --hard [commit]

HEAD 说明：

* HEAD 表示当前版本
* HEAD^ 上一个版本
* HEAD^^ 上上一个版本
* 以此类推...

可以使用 ～数字表示

* HEAD~0 表示当前版本
* HEAD~1 上一个版本
* HEAD~2 上上一个版本
* 以此类推...

**HEAD指针**

有关HEAD指针，HEAD指针其实是存在在版本库中的，默认指向最新的一个版本，这个最新的版本名字就是HEAD，而它的上一个就是HEAD^，以此类推

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从指向回退的版本，然后顺便刷新工作区文件。

**三个重置参数HEAD的行为：**

 –hard：**重设“暂存区”和“工作区”**，**从<commit>以来在工作区中的任何改变都被丢弃，并把HEAD指向<commit>**。**（彻底回退到某个版本，本地的源码也会变为上一个版本的内容。）**

 –soft：**“工作区”中的内容不作任何改变，HEAD指向<commit>，自从<commit>以来的所有改变都会回退到“暂存区”中，显示在git status的*“Changes to be committed”*中。（回退到某个版本，只回退了commit的信息。如果还要提交，直接commit即可。）**

 –mixed：**仅重设“暂存区”，并把HEAD指向<commit>，但是不重设“工作区”，本地文件修改不受影响。**这个模式是默认模式，即当不显示告知git reset模式时，会使用mixed模式。这个模式的效果是，工作区中文件的修改都会被保留，不会丢弃，但是也不会被标记成“*Changes to be committed*”，但是会提示文件未被更新。**（回退到某个版本，只保留源码，回退commit和index信息**

soft比较费解，并且也没什么常用命令是它的，暂时忽略。

head和mixed（默认）的不同就是前者会reset暂存区和工作区，后者只reset暂存区，不动工作区——所以如果你希望保留工作区就用mixed不希望就用hard

* 反悔单个文件的版本，这很可（但还是不要依赖，就是和版本回退一样特殊情况才用）

# 恢复**某个commit的指定文件**到暂存区和工作区

$ git checkout [commit] [file]

#### 删除文件

一般情况下，你通常直接在文件管理器中把没用的文件删了

这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，git status命令会立刻告诉你哪些文件被删除了：

$ git status

现在你有两个选择，一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit：

$ git rm test.txt

$ git commit -m "remove test.txt"

（或者后面git add . 再commit、push也是一样的，只是或许我们不会用到git add .）

【撤销修改】另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

$ git checkout -- test.txt

#### 如何取消一个目录的git初始化

手动删掉.git文件夹即可或者命令**rm -rf. git**删除

[如何取消一个目录的git初始化 - SegmentFault 思否](https://segmentfault.com/q/1010000006717152)

[git取消跟踪文件目录\_sun2009\_的博客-CSDN博客](https://blog.csdn.net/sun2009_/article/details/70198580)

[如何完全删除一个Git库,初始化一个新的？\_git\_酷徒编程知识库](https://kb.kutu66.com/git/post_17910)

### [总结](https://www.cnblogs.com/guoyaohua/p/Git-tutorial.html#%E6%80%BB%E7%BB%93)

直接点开链接看

### 之前的基本命令（git 和 github入门）

#### 一、Git的简单使用

（参考链接：<https://blog.csdn.net/javaandroid730/article/details/53522872> ）

git使用最好的参考资料是Pro Git book。<https://git-scm.com/book/zh/v2>

***第一步 下载Git for Windows***

1. 在官网点击Download，下载对应的exe文件，注意你的操作系统是32位还是64位。
2. 双击安装，中间不用做任何改动，一直下一步就行。如果你想修改安装位置，请放在纯英文路径下。

***第二步 创建一个本地hello-world仓库***

1. 安装成功，你现在就可以使用git命令行工具了。在你想要下载代码的路径，点击鼠标右键，选择Git Bash here。注意，你的代码路径也应是纯英文的。（打开命令行（cmd）或者在想要创建repository的地方右键鼠标并点击 Git Bash Here 打开窗口）
2. 在命令行输入 mkdir hello-word，创建一个新文件夹。你可以使用ls命令来查看当前目录下有哪些文件和文件夹。
3. 输入cd hello-world进入新文件夹，注意在输入命令时，你可以用Tab键来自动补全。（也就是hello-world才是仓库）
4. 输入git init初始化Git仓库。此时用ls -a查看当前目录，可以看到多了一个.git/的文件夹，此文件夹保存了版本控制的所有相关信息。

注意，在此处使用的所有命令，如果你是在Linux环境下开发，用法都是完全一样的。所以对于完全没有Linux使用经验的学员，这也是一个开始接触Linux工作方式的好方法。 接下来，让我们创建一份简单的说明文件，并提交到版本库中。

1. 输入echo "This is a simple practise" > readme.txt，创建一个readme.txt文件。（ps:> + 文件名 可创建文件）
2. 输入git status查看当前版本库状态，在Untracked files(未跟踪文件)下，会出现红色的readme.txt，代表此文件还未被Git所管理。
3. 使用git add readme.txt，将该文件加入缓冲区，如果你确定所有的修改都需要提交，可以使用git add .来加入所有修改。现在用git status查看，将看到文件名变为绿色。（add .多一点，因为不用去看文件名）
4. 使用git commit -m "This is my first commit via Git!"来提交修改，-m后面所带的参数是本次提交信息，一般用来记录本次提交的主要意图。
5. 提交成功后，可以用git log查看历史提交记录。每个记录都会有提交id，作者和提交日期。
6. 你可以用git branch查看当前有哪些分支，当然，因为我们没有创建任何分支，目前只会有一个master分支。
7. 使用git checkout -b feature创建一个名为feature的分支，再用git branch查看一下。 以上是最最基本的Git操作，大家可以在此hello-world项目中随意尝试各种其他Git命令，最好的参考资料是Pro Git book。<https://git-scm.com/book/zh/v2>

***第三步：版本回退***

1. git reset --hard HEAD^

HEAD^表示上一个版本，HEAD^^表示上上一个版本，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。

1. git reset --hard 版本号

怎么找版本号

a. git log 可以

b. git reflog 找寻所有历史版本号，满足您的所有需求

***git小结***

--git init

--git add .

--git commit -m "注释"

--git status

--git log

--git diff

--git reset --hard +

注意：学会Git的唯一方式是在实际使用中学习，切记不要尝试先记住一大堆理论知识或者Git命令。

Git基础可以看下面两个链接，特别是上面没有的版本回退，还有SSH的创建

<https://www.jianshu.com/p/296d22275cdd>

<https://www.cnblogs.com/schaepher/p/5561193.html>

git和GitHub的就我做自己的补充，因为两位大佬说的都不全符合我的想法

##### 二、 Github与Git的关联（综合了两个链接的方法）

0 前期工作

a. 到Github注册账号。

b. 本地配置用户名和邮箱（如果已经设置好，跳过该步）：

git config --global user.name "你的用户名"

git config --global user.email "你的邮箱"

c. 生成ssh key

运行 ssh-keygen -t rsa -C "你的邮箱" ，它会有三次等待你输入，直接回车即可。

将生成的ssh key复制到剪贴板，执行 clip < ~/.ssh/id\_rsa.pub （或者到上图提示的路径里去打开文件并复制）：

d. 打开Github，进入Settings--点击左边的 SSH and GPG keys--点new SSH key--将ssh key 粘贴到右边的Key里面，Title随便命名即可

--点击下面的---Add SSH key 就添加成功了---测试一下吧，执行 ssh -T git@github.com

***最简单的方法（先远程建仓-克隆本地建立关联-初始化提交-普通提交）***

1. 建GitHub仓库
2. 本地clone（建立关联了） git clone + github克隆链接 这样就不用git remote add origin 你复制的地址
3. 直接本地git有修改即add、commit后，提交远程github 你的修改 git push -u origin master（初次）
4. 然后后面的修改提交就都只用git push

***第二种关联方式（本地建与远程同名仓ps 本地和远程谁先建都可-用命令建立关联-初始化提交-普通提交）***

1. 本地建仓或已有仓（建仓就是文件夹git init）
2. 远程建同名仓
3. git remote add origin 你复制的地址->建立关联 如果你在创建 repository 的时候，加入了 README.md 或者 LICENSE (就是远程仓比本地仓高版本)，那么 github 会拒绝你的 push 。 你需要先执行 git pull origin master。（pull命令 从远程仓库同步，在本地版本低于远程仓库版本的时候，获取远程仓库的commit）
4. 直接本地git有修改即add、commit后，提交远程github 你的修改 git push -u origin master（初次）
5. 然后后面的修改提交就都只用git push

**大佬总结**

1. 设置用户名 config --global user.name "你的用户名"
2. 设置邮箱 config --global user.email "你的邮箱"
3. 生成ssh key ssh-keygen -t rsa -C "你的邮箱" 这条命令前面不用加git 4.clone 地址 从网络上某个地址拷贝仓库(repository)到本地
4. 添加远程仓库 remote add origin 你复制的地址 设置origin
5. 上传并指定默认 push -u origin master 指定origin为默认主机，以后push默认上传到origin上
6. 提交到远程仓库 push 将当前分支增加的commit提交到远程仓库
7. 从远程仓库同步 pull 在本地版本低于远程仓库版本的时候，获取远程仓库的commit

### 其他的就差不多了，后面有需要就点开参考链接查阅或者网上搜索

[找了两个视频，可能会有那种实例性的，不能还没看过，需要时看](https://www.one-tab.com/page/BdzprvY2T_eNfU1nRk8lxQhttps:/www.one-tab.com/page/BdzprvY2T_eNfU1nRk8lxQ)【选】

你觉得我的打字速度和打字姿势怎么样，其实我觉得或许在后面我需要练习一下在键盘上打字，其实我觉得就是习惯问题，其他的差不多

# 后面到了贡献开源的时候再去学习一下Github端的常见操作，例如提交问题什么的