git知识整理

参考链接：https://www.liaoxuefeng.com/wiki/896043488029600/898732792973664

1. SVN与Git的区别？

1. SVN是集中式版本控制系统，Git是分布式版本控制系统；
2. GIT把内容按元数据方式存储，而SVN是按文件：因为git目录是处于个人机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。
3. GIT分支和SVN的分支不同： svn会发生分支遗漏的情况，而git可以同一个工作目录下快速的在几个分支间切换，很容易发现未被合并的分支，简单而快捷的合并这些文件。
4. GIT没有一个全局的版本号，而SVN有（Git缺点）；
5. GIT的内容完整性要优于SVN:GIT的内容存储使用的是SHA-1哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

2. 集中式和分布式的区别

集中式版本控制系统：版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。集中式版本控制系统最大的毛病就是必须联网才能工作。
分布式版本控制系统：分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

3. 创建版本库

第一步创建一个空目录：

```
$ mkdir learngit
$ cd learngit
$ pwd
/Users/michael/learngit
```

- pwd（print working directory）打印当前的工作目录；
  【注】windows系统注意目录名（父目录）不包含中文以防出现奇怪问题；

第二步，通过`git init`命令吧这个目录变成Git可以管理的仓库（即**初始化Git仓库**）：

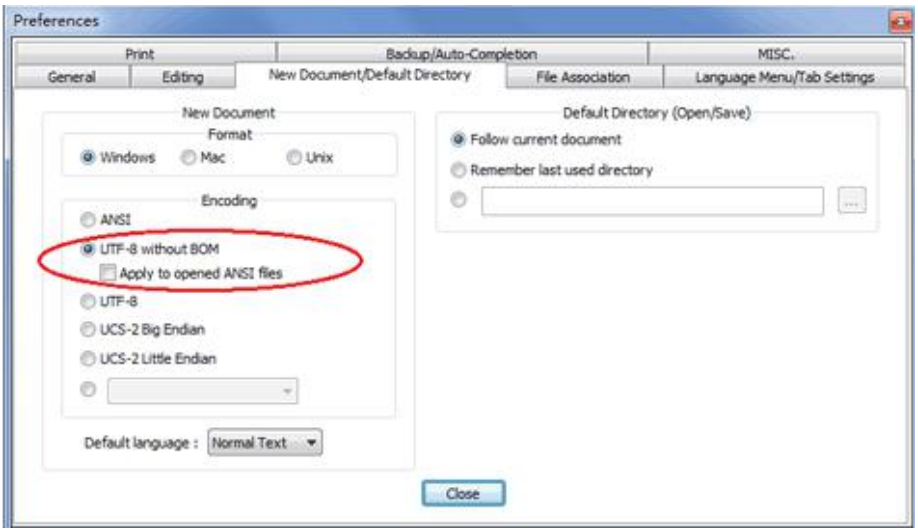
```
$ git init
Initialized empty Git repository in
/Users/michael/learngit/.git/
```

瞬间Git就把仓库建好了，而且告诉我们是一个空的仓库（empty Git repository），细心的读者可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

如果你没有看到`.git`目录，那是因为这个目录默认是隐藏的，用`ls -ah`命令就可以看见。

使用windows特别注意：

千万不要使用Windows自带的记事本编辑任何文本文件， 原因是Microsoft开发记事本的团队使用了一个非常弱智的行为来保存UTF-8编码的文件，他们自作聪明地在每个文件开头添加了0xefbbbf（十六进制）的字符，你会遇到很多不可思议的问题，比如，网页第一行可能会显示一个“?”，明明正确的程序一编译就报语法错误，等等，都是由记事本的弱智行为带来的。建议你下载Notepad++代替记事本，不但功能强大，而且免费！记得把Notepad++的默认编码设置为UTF-8 without BOM即可：



第三步 添加文件到Git仓库，分两步：

1. 使用命令`git add <file>`，注意，可以多次使用添加多个文件；
    - 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]`
    - 添加当前目录的所有文件到暂存区
      `$ git add .`
      添加每个变化前，都会要求确认,对于同一个文件的多处变化，可以实现分次提
      `$ git add -p`
2. 使用命令`git commit -m <message>`完成；
    - 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]`

- 提交工作区自上次commit之后的变化，直接到仓库区:
  `$ git commit -a`
- 提交时显示所有diff信息:
  `$ git commit -v`
- 使用一次新的commit，替代上一次提交,如果代码没有任何新变化，则用来改写上一次commit的提交信息:
  `$ git commit --amend -m [message]`
- 重做上一次commit，并包括指定文件的新变化:
  `$ git commit --amend [file1] [file2] ...`

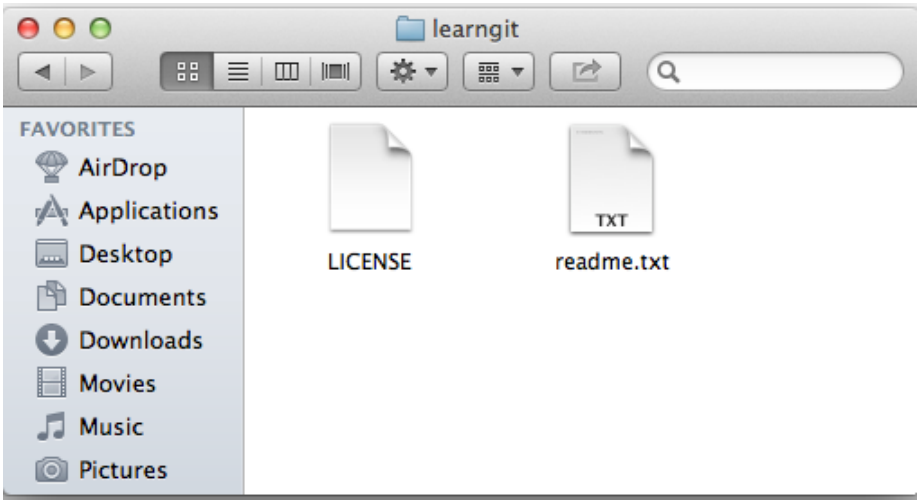
4. 版本回退

1. 查看信息：
   - 显示当前分支的版本历史:
     `$ git log`
   - 显示简短的log：
     `$ git log --pretty=oneline`
   - 显示commit历史，以及每次commit发生变更的文件:
     `$ git log --stat`
   - 搜索提交历史，根据关键词
     `$ git log -S [keyword]`
   - 显示某个commit之后的所有变动，每个commit占据一行：
     `git log [tag] HEAD --pretty=format:%s`
     ...
   - 记录你的每一次命令：
     `$ git reflog`
2. 版本回退：
   - 回退到上一个版本
     `$ git reset --hard HEAD^`
   - 回退到上上个版本
     `$ git reset --hard HEAD^^`
   - 回退到前n个版本
     `$ git reset --hard HEAD~n`
   - 回退到某个版本
     `$ git reset --hard 1094a`(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知道id开头未‘1094a’，然后顺便把工作区的文件更新了。)

5. 工作区和暂存区

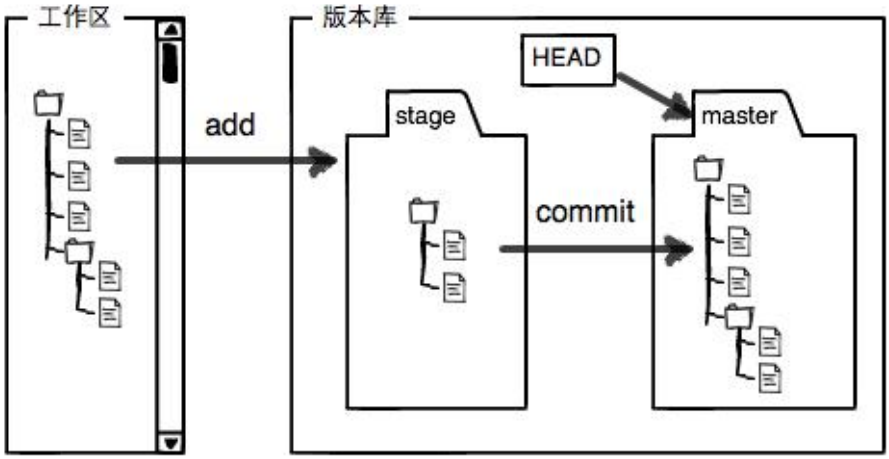
工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的learngit文件夹就是一个工作区：



版本库（Repository）

工作区有一个隐藏目录.git，这个不算工作区，而是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为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



6.管理修改

`$git commit`只负责把暂存区的修改提交,提交后，用`$git diff HEAD -- readme.txt`命令可以查看工作区和版本库里面最新版本的区别;
【总结】： 不用`git add`到暂存区，那就不会加入到`commit`中；

7. 撤销修改

- 还没有add到暂存区时`$git checkout -- file`可以丢弃工作区的修改(撤销工作区修改)：
  `$git checkout -- reademe.txt`
- 加入到暂存区未commit（撤销暂存区的修改,回到工作区）
  `$git reset HEAD <file>`
  `$git reset HEAD readme.txt`
  用HEAD表示到最新的版本；
  用`$git status`查看，暂存区时干净的，工作区有修改；
- 已经commit了，只能参考4版本回退的方法了；

- 撤销merging状态状态，`git reset --hard HEAD`；
- 撤销(dev|REBASE 1/5)状态， `git rebase --abort` 来取消目前的进程；

---

8. 删除文件

- 在文件管理器中把没用的文件删了，或者用rm命令删了：
  `$ rm test.txt`
  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
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be
committed)
  (use "git checkout -- <file>..." to discard changes in
working directory)

    deleted:    test.txt

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

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
rm 'test.txt'

$ git commit -m "remove test.txt"
[master d46f35e] remove test.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt
```

现在，文件就从版本库中被删除了。
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： (撤销操作)
`$ git checkout -- test.txt`
`$git checkout`其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

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。

9. 远程仓库

- 关联远程仓库:
  `$ git remote add origin git@github.com:michaelliao/learngit.git`
- 本地库的所有内容推送到远程库上:
  `$ git push -u origin master`
  由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令，以后直接`git push`进行推送；

10. 从远程库克隆

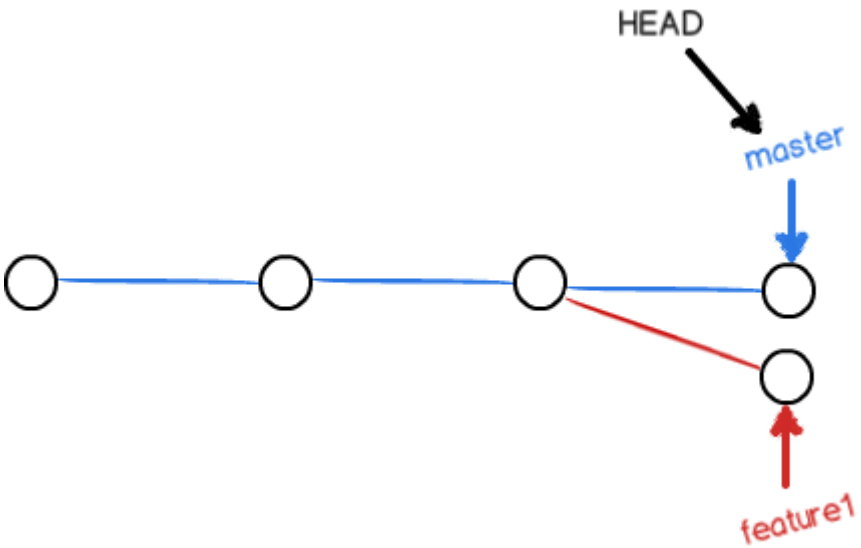
```
$ git clone git@github.com:Grekit-Sun/gitskills.git
```

11. 创建合并

- 创建dev分支，然后切换到dev分支：`$git switch -c dev`;
- 查看当前分支：`$git branch`;
- 切回master分支：`$git switch master`
- 将dev分支工作合并到master:`$git merge dev`
- 合并完成后可以删除分支：`$git branch -d dev`

12. 解决冲突

`master`分支和`feature1`分支各自都分别有新的提交，变成了这样：



这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们试试看：

```
$ git merge feature1
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the
result.
```

Git告诉我们，`readme.txt`文件存在冲突，必须手动解决冲突后再提交。`git status`也可以告诉我们冲突的文件：

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   readme.txt

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

我们可以直接查看readme.txt的内容：

```
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.
<<<<<<< HEAD
Creating a new branch is quick & simple.
=======
Creating a new branch is quick AND simple.
>>>>>>> feature1
```

Git用`<<<<<<<`，`=======`，`>>>>>>>`标记出不同分支的内容，我们修改好后保存；

再提交：

```
$ git add readme.txt
$ git commit -m "conflict fixed"
[master cf810e4] conflict fixed
```

最后，删除`feature1`分支：

```
$ git branch -d feature1
Deleted branch feature1 (was 14096d0).
```

分支管理策略

通常，合并分支时，如果可能，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。
下面我们实战一下--no-ff方式的git merge：

- 创建并切换dev分支：

```
$ git switch -c dev
Switched to a new branch 'dev'
```

- 现在，我们切换回master：

```
$ git checkout master
Switched to branch 'master'
```

- 准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward：

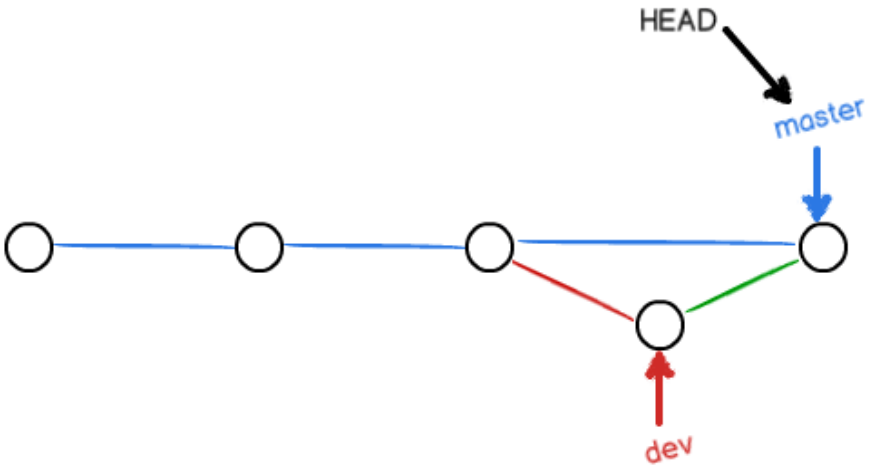
```
$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'recursive' strategy.
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
```

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。

- 合并后，我们用git log看看分支历史：

```
$ git log --graph --pretty=oneline --abbrev-commit
*   e1e9c68 (HEAD -> master) merge with no-ff
|\
| * f52c633 (dev) add merge
|/
*   cf810e4 conflict fixed
...
```

可以看到，不使用Fast forward模式，merge后就像这样：



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。

举一反三

git rebase 和merge

`git rebase`使用:

1. `git rebase master`:分支（自己开发的分支），master待合进的分支;
2. `git merge feature(分支)`

使用 `rebase` 和 `merge` 的基本原则：

- 下游分支更新上游分支内容的时候使用 rebase;
- 上游分支合并下游分支内容的时候使用 merge;
- 更新当前分支的内容时一定要使用 --rebase 参数;
  例如现有上游分支 master，基于 master 分支拉出来一个开发分支 dev，在 dev 上开发了一段时间后要把 master 分支提交的新内容更新到 dev 分支，此时切换到 dev 分支，使用 git rebase master。

等 dev 分支开发完成了之后，要合并到上游分支 master 上的时候，切换到 master 分支，使用 `git merge dev`;

git中merge，rebase，cherry-pick，patch的联系与区别

这些操作都是为了把一个分支上的工作加到另一个分支上。

1. merge
   把另一个分支合并到当前分支上。

2. rebase
   把当前分支的提交在另一分支上重演。（如果可以成功重演，本分支将会消失）
3. cherry-pick
   把本分支或者其他分支的某一次或某几次提交，在当前分支上重演。
4. patch
   把一次或几次提交，做成补丁文件（可以远程发送给其他人，这是与cherry-pick最大的不同）。这个补丁文件可以被应用到其它分支上。

git rebasse自己理解

- `git rebase <主干>`：就是当前的分支合并入主干版本，解决冲突依次`git add <>`,`git rebase --continue`；之后切换到主干版本用`git merge <分支>`;

---

git rm与git rm --cached 的区别

git rm 是删除暂存区或分支上的文件, 同时也删除工作区中这个文件。
git rm --cached是删除暂存区或分支上的文件,但本地还保留这个文件， 是不希望这个文件被版本控制。

---

git查看本机秘钥

输入命令`$ ssh-keygen -t rsa -C "1024809664@qq.com"`

Git学习心得

> 由于之前没有用过git，一开始学习的时候想要把每条指令都学习理解，都看了一遍之后，发现很多都记不住，然后我就在这边自己练习，模拟真实项目开发环境，练习中遇到各种问题，如连接不上仓库，是因为没有添加秘钥等，通过磊哥的指导，以及自己的再次深入学习，之后对常用的命令得以熟悉，可能后面工作中应用到git时还会遇到些小问题，有了这些天的学习基础，我有信心一定可以解决的；