返回目录

目录

返回目录

返回目录

一、 阅读说明

*绿色斜体*代表个人的思考理解，*黄色斜体*代表阅读理解过程中的疑问，红色正体代表关键重要信息，下划线代表次关键重要信息

二、 集中式与分布式

集中式版本控制系统相比，每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。

在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。（*那“中央服务器”换掉咋办？*）

三、 创建版本库

1. 创建空目录：

   Windows 下路径不要包含中文

2. 通过 git init 命令把这个目录变成 Git 可以管理的仓库

四、 将文件添加到版本库

所有的版本控制系统，其实只能跟踪文本文件的改动，而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从 100KB 改成了 120KB，但到底改了啥，版本控制系统不知道。Microsoft 的 Word 格式是二进制格式

1. 文件放置目录下

   文件放置 Git 仓库目录或子目录下

2. 文件添加到仓库

```
$ git add readme.txt
```

文件提交到仓库

```
$ git commit -m "wrote a readme file"
[master (root-commit) eaadf4e] wrote a readme file
 1 file changed, 2 insertions(+)
 create mode 100644 readme.txt
```

返回目录

git commit 命令，-m 后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。命令执行成功后会告诉你，1 file changed：1 个文件被改动（我们新添加的 readme.txt 文件）；2 insertions：插入了两行内容（readme.txt 有两行内容）

commit 可以一次提交很多文件，所以你可以多次 add 不同的文件，比如：

```
$ git add file1.txt
$ git add file2.txt file3.txt
$ git commit -m "add 3 files."
```

五、 仓库状态及文件变化

成功地添加并提交了一个 readme.txt 文件，然后继续修改 readme.txt 文件，运行 git status 命令看看结果：

```
$ 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")
```

上面的命令输出告诉我们，readme.txt 被修改过了，但还没有准备提交的修改，git status 用于查看是否有文件修改，是否有修改已经添加(add)，是否有需求未提交。git diff 这个命令用于查看具体修改了什么内容。

六、 版本回退

文件修改到一定程度的时候，就可以“保存一个快照”，这个快照在 Git 中被称为 commit。git log 命令显示从最近到最远的提交日志以及当前版本状态：

```
$ git log
commit 1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master)
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Fri May 18 21:06:15 2018 +0800

    append GPL
```

返回目录

```
7.
8. commit e475afc93c209a690c39c13a46716e8fa000c366
9. Author: Michael Liao <askxuefeng@gmail.com>
10.   Date:   Fri May 18 21:03:36 2018 +0800
11.
12.       add distributed
```

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。现在，我们要把当前版本 append GPL 回退到上一个版本 add distributed，使用 git reset 命令：

```
1. $ git reset --hard HEAD^
2. HEAD is now at e475afc add distributed
```

此时文件恢复至上一版本，而最新版本(append GPL)的信息通过 git log 无法找到，此时可以通过 git reflog 命令找回 commit id ，在通过 reset 命令恢复即可：

```
1. $ git reflog
2. e475afc HEAD@{1}: reset: moving to HEAD^
3. 1094adb (HEAD -> master) HEAD@{2}: commit: append GPL
4. e475afc HEAD@{3}: commit: add distributed
5. eaadf4e HEAD@{4}: commit (initial): wrote a readme file
6. $ git reset --hard 1094a #  写 commit id 前几位即可
7. HEAD is now at 83b0afe append GPL
```

七、 工作目录和暂存区

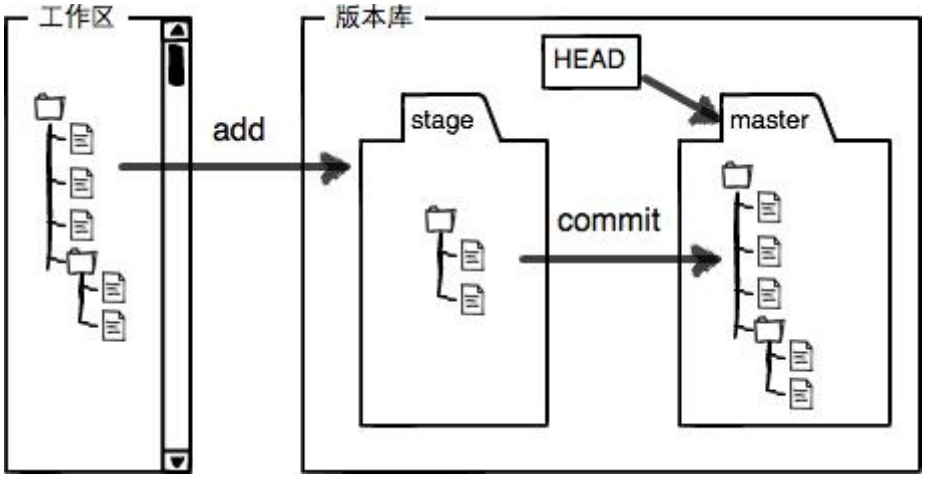
1. 工作目录

电脑中看到的目录

2. 版本库（Repository）

工作目录有一个隐藏目录.git，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 stag（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。

返回目录



Git 版本库里添加的时候，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。

因为我们创建 Git 版本库时，Git 自动为我们创建了唯一一个 master 分支，所以现在 git commit 就是往 master 分支上提交更改。可以简单理解为需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

八、 撤销修改

1. 撤销暂存区的修改

如果修改通过 git add 添加到了暂存区，但是尚未 git commit，通过命令 git reset HEAD <file>可以把暂存区的修改撤销掉(unstage)，重新放回工作区(git reset 命令既可以回退版本，也可以把暂存区的修改回退到工作区)

```
1. $ git reset HEAD readme.txt
2. Unstaged changes after reset:
3. M    readme.txt
```

2. 撤销工作区的修改

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：1.一种是 readme.txt 自修改后还没有被放到暂存区，撤销修改就回到和版本库一模一样的状态；2.一种是 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。

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

3. 小结

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。

返回目录

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

九、 删除文件

1. 文件管理器中将不需要的文件删除
2. 从版本库删除

使用命令 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
```

3. 文件误删，从版本库中恢复

文件管理器中误删文件，使用 git checkout 命令从版本库中恢复，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”(*又是 git checkout 命令，所以究竟 git checkout 是使用暂存区还是版本库都内容替代工作目录？？*)

```
$ git checkout -- test.txt
```

十、 远程仓库——与 GitHub 连接前设置

由于你的本地 Git 仓库和 GitHub 仓库之间的传输是通过 SSH 加密的，所以需要一点设置

1. 创建 SSH Key

在用户主目录下，看看有没有.ssh 目录，如果有，再看看这个目录下有没有 id_rsa 和 id_rsa.pub 这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开 Shel（Windows 下打开 Git Bash），创建 SSH Key:

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

id_rsa 和 id_rsa.pub 两个文件就是 SSH Key 的秘钥对，id_rsa 是私钥，不能泄露出去，id_rsa.pub 是公钥

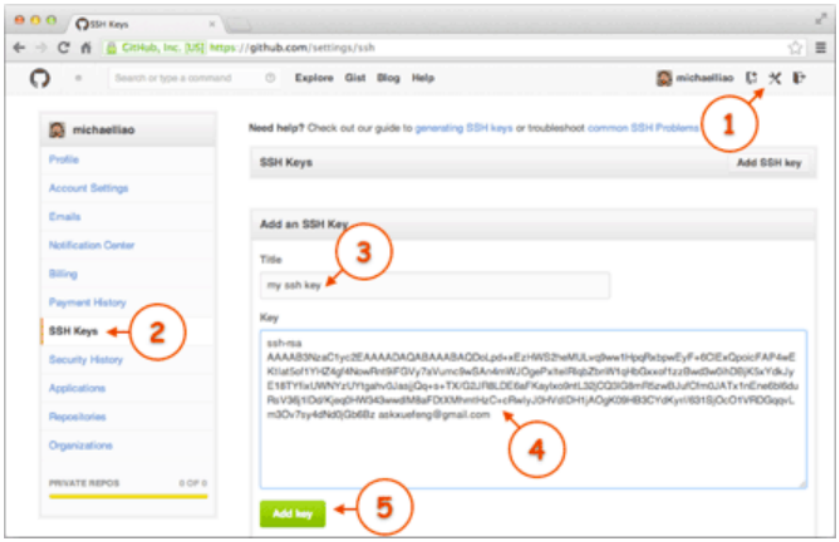
2. Account settings 添加 key

GitHub 允许添加多个 Key。假定你有若干电脑，只要把每台电脑的 Key 都添加到 GitHub，就可以在每台电脑上往 GitHub 推送了。

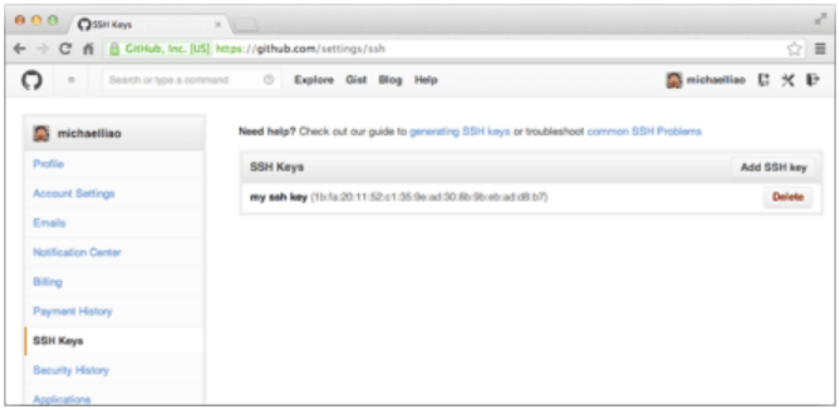
返回目录

第2步：登陆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：



十一、 添加远程仓库

目标：本地创建了一个 Git 仓库后，又想在 GitHub 创建一个 Git 仓库，并且让这两个仓库进行远程同步，这样 GitHub 上的仓库既可以作为备份，又可以让其他人通过该仓库来协作

1. Github 建立新仓库
2. 关联远程库

GitHub 新建的仓库是空的，此时可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后把本地仓库的内容推送到 GitHub 仓库。关联命令如下：

```
1. #  命令格式：git remote add origin git@server-name:path/repo-name.git
2. $ git remote add origin git@github.com:username/learngit.git
```

添加后，远程库的名字就是 origin，这是 Git 默认的叫法，可更改

3. 本地库的所有内容推送到远程库

返回目录

本地库的内容推送到远程，用 git push 命令，实际上是把当前分支 master 推送到远程。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。

```
1. $ git push -u origin master
2. Counting objects: 20, done.
3. Delta compression using up to 4 threads.
4. Compressing objects: 100% (15/15), done.
5. Writing objects: 100% (20/20), 1.64 KiB | 560.00 KiB/s, done.
6. Total 20 (delta 5), reused 0 (delta 0)
7. remote: Resolving deltas: 100% (5/5), done.
8. To github.com:michaelliao/learngit.git
9.  * [new branch]      master -> master
10.   Branch 'master' set up to track remote branch 'master' from
   'origin'.
```

以后推送的简化命令为:

```
1. $ git push origin master
```

十二、 从远程仓库克隆

目标：本地没有仓库，先在 GitHub 创建远程仓库，然后从远程库克隆

1. Github 建立新仓库
2. 使用命令 git clone 克隆一个本地库

```
1. $ git clone git@github.com:michaelliao/gitskills.git
2. Cloning into 'gitskills'...
3. remote: Counting objects: 3, done.
4. remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 3
5. Receiving objects: 100% (3/3), done.
```

十三、 分支管理

假设准备开发一个新功能，但是需要两周才能完成，第一周完成 50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

返回目录

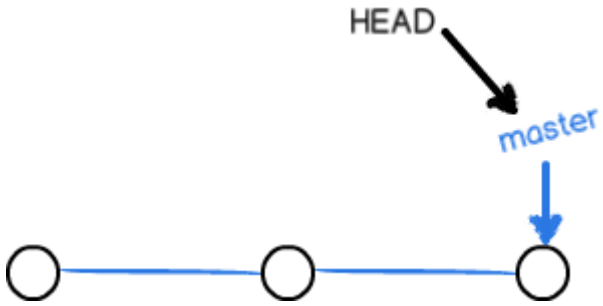
创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性将分支合并到原来的分支上，这样既安全，又不影响别人工作。

十四、 创建与合并分支

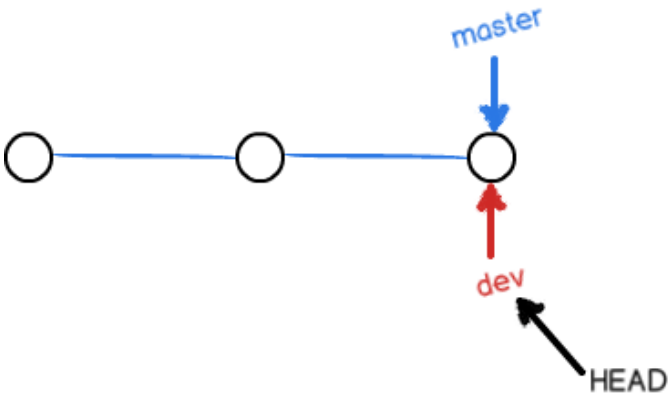
1. 原理

Git 把每次提交串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在 Git 里这个分支叫主分支，即 master 分支。HEAD 严格来说不是指向提交，而是指向 master，master 才是指向提交的，所以 HEAD 指向的就是当前分支。

一开始的时候，master 分支是一条线，Git 用 master 指向最新的提交，再用 HEAD 指向 master，就能确定当前分支，以及当前分支的提交点，每次提交，master 分支都会向前移动一步

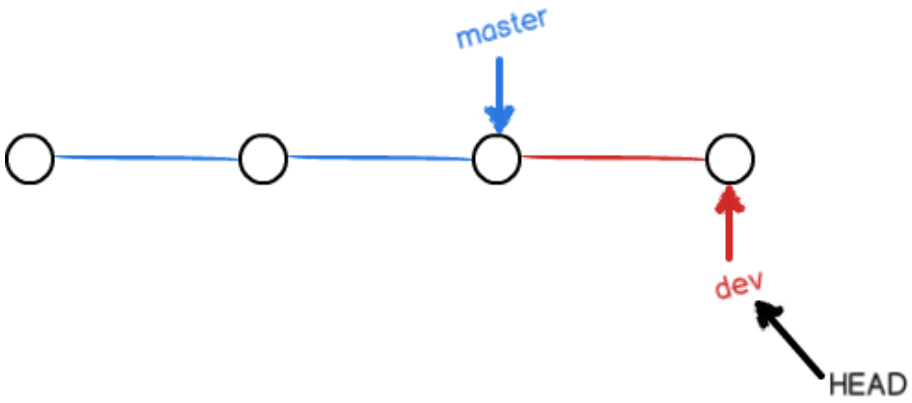


创建新的分支，例如 dev 时，Git 新建了一个指针叫 dev，指向 master 相同的提交，再把 HEAD 指向 dev，就表示当前分支在 dev 上:

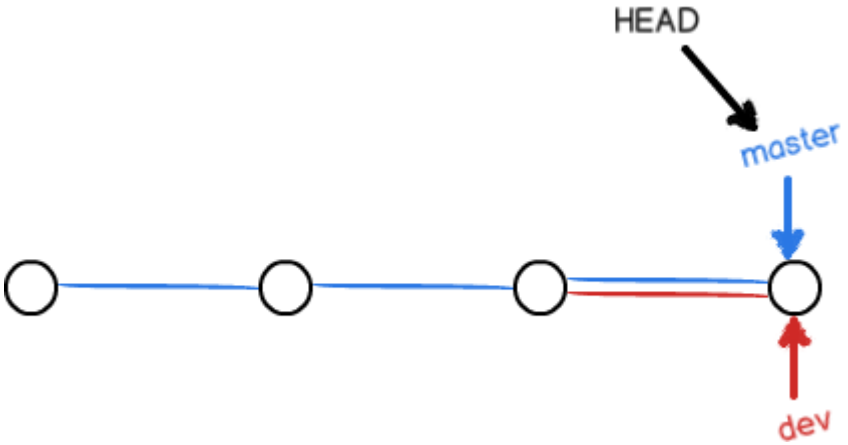


从现在开始，对工作区的修改和提交就是针对 dev 分支了，比如新提交一次后，dev 指针往前移动一步，而 master 指针不变:

返回目录



假如在 dev 上的工作完成了，就可以把 dev 合并到 master 上。直接把 master 指向 dev 的当前提交，就完成了合并。



2. 操作命令

创建分支:

```
1. $ git branch dev
```

切换分支:

```
1. $ git checkout dev
2. Switched to branch 'dev'
```

以上命令可用下述命令替代(创建并切换分支):

```
1. $ git checkout -b dev
2. Switched to a new branch 'dev'
```

用 git branch 命令查看当前分支，git branch 命令会列出所有分支，当前分支前面会标一个*号。

合并分支:

```
1. $ git merge dev
2. Updating d46f35e..b17d20e
3. Fast-forward
```

返回目录

```
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
```

删除分支:

```
$ git branch -d dev
Deleted branch dev (was b17d20e).
```

十五、 解决分支冲突

- %% %号本身