Day04

一、复习

二、git的使用

1.git简介

1.1案例引入

> 如果你用Word写过毕业论文，那你一定有这样的经历：
>
> 想删除一个段落，又怕将来想恢复找不回来怎么办？有办法，先把当前文件“另存为……”一个新的Word文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的Word文档会很多：
>
> 过了一周，你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了，只好一个一个文件去找，真麻烦。
>
> 看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。
>
> 更要命的是，有些部分需要你的财务同事帮助填写，于是你把文件Copy到U盘里给她（也可能通过Email发送一份给她），然后，你继续修改Word文件。一天后，同事再把Word文件传给你，此时，你必须想想，发给她之后到你收到她的文件期间，你作了哪些改动，得把你的改动和她的部分合并，真困难。
>
> 于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便?
>
> 这个软件用起来就应该像这个样子，能记录每次文件的改动：

版本	文件名	用户	说明	日期
1	service.doc	张三	删除了软件服务条款5	7/12 10:38
2	service.doc	张三	增加了License人数限制	7/12 18:09
3	service.doc	李四	财务部门调整了合同金额	7/13 9:51
4	service.doc	张三	延长了免费升级周期	7/14 15:17

> 这样，你就结束了手动管理多个“版本”的史前时代，进入到版本控制的20世纪。
>
> Git是目前世界上最先进的分布式版本控制系统（没有之一）。
>
> Git有什么特点？简单来说就是：高端大气上档次！

1.2git的由来

> 很多人都知道，Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。

Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？

事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！

你也许会想，为什么Linus不把Linux代码放到版本控制系统里呢？不是有CVS、SVN这些免费的版本控制系统吗？因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。

不过，到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。

安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了（监控工作做得不错！），于是BitMover公司怒了，要收回Linux社区的免费使用权。

Linus可以向BitMover公司道个歉，保证以后严格管教弟兄们，嗯，这是不可能的。实际情况是这样的：

Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！牛是怎么定义的呢？大家可以体会一下。

Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。

历史就是这么偶然，如果不是当年BitMover公司威胁Linux社区，可能现在我们就没有免费而超级好用的Git了

1.3集中式和分布式

SVN都是集中式的版本控制系统，而Git是分布式版本控制系统，集中式和分布式版本控制系统有什么区别呢？

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。例如：中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆【缺点：必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟，这还不得把人给憋死啊】

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。如果是一个团队合作工作，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

二者之间的区别：和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为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。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法干活了。

实际情况：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，也可能今天你的同事病了，他的电脑压根没有开机。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。

> 当然，Git的优势不单是不必联网这么简单，后面我们还会看到Git极其强大的分支管理，把SVN等远远抛在了后面。

2.安装git

> git：查看是否已经安装
>
> sudo apt-get install git ： 安装

3.创建版本库

3.1什么是版本库

> 版本库又被称为仓库，【repository】,初期可以理解为一个目录，这个目录里面管理的文件都可以被称为被git管理起来的，每个文件的修改，删除等的操作git都能进行跟踪

3.2创建版本库

> git init：将一个普通目录变成版本库

```
演示命令：
ijeff@Rock:~$ cd Desktop/
ijeff@Rock:~/Desktop$ mkdir python1901
ijeff@Rock:~/Desktop$ cd python1901/
ijeff@Rock:~/Desktop/python1901$ pwd
/home/ijeff/Desktop/python1901
ijeff@Rock:~/Desktop/python1901$ git init
已初始化空的 Git 仓库于 /home/ijeff/Desktop/python1901/.git/
#就创建一个git仓库【版本库】，

ijeff@Rock:~/Desktop/python1901$ ls
ijeff@Rock:~/Desktop/python1901$ ls -a
.  ..  .git
#.git是一个目录，就是用来跟踪管理版本库

#注意：也不一定目录是空的，但是为了安全起见，最好以一个空的目录作为版本库
```

3.3把文件添加到版本库

> 注意：所有的版本控制系统都是跟踪的是文件的改动
>
> git add filename :将文件添加到缓存区
>
> git commit -m "日志" ： 提交文件到版本库【仓库】

```
演示命令：
angRock@Rock:~/Desktop/python1901$ touch text.txt
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git commit -m "create a new file and init"

*** 请告诉我你是谁。          #不知道主人是谁，则需要配置用户名和邮箱
                            #注意：用户名和邮箱来自github上的注册

运行
```

```
  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

来设置您账号的缺省身份标识。
如果仅在本仓库设置身份标识，则省略 --global 参数。
fatal: 无法自动探测邮件地址（得到 'ijeff@Rock.(none)'）
ijeff@Rock:~/Desktop/python1901$ git config --global user.email "11111@163.com"      #配置github上的邮箱
ijeff@Rock:~/Desktop/python1901$ git config --global user.name "ijeff-git"                #配置github上的用户名
ijeff@Rock:~/Desktop/python1901$ git commit -m "create a new file and init"
#-m后面输入的是本次提交的说明【就是所谓的日志】，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
[master （根提交） d4b0bde] create a new file and init
 1 file changed, 1 insertion(+)
 create mode 100644 text.txt
#git commit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）；3 insertions：插入了一行内容

#再次修改文件，则重复git add和git commit命令
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git commit -m "add hello"
[master 1f12c8b] add hello
 1 file changed, 1 insertion(+)
ijeff@Rock:~/Desktop/python1901$
```

问题：为什么Git添加文件需要add，commit一共两步呢？因为commit可以一次提交很多文件，所以你可以多次add不同的文件，如下：

```
演示命令：
ijeff@Rock:~/Desktop/python1901$ touch text1.txt
ijeff@Rock:~/Desktop/python1901$ touch text2.txt
ijeff@Rock:~/Desktop/python1901$ git add text1.txt
ijeff@Rock:~/Desktop/python1901$ git add text2.txt
ijeff@Rock:~/Desktop/python1901$ git commit -m "create twofiles"
[master ce63cb0] create twofiles
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 text1.txt
 create mode 100644 text2.txt

  #总结：add一次添加一个文件，commit可以一次提交多个文件
```

4.时光穿梭机【覆水可收】

git status:查看仓库当前的状态

git diff filename:查看仓库具体的改动

```
演示命令：
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git add text.txt
```

```
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
要提交的变更：
  （使用 "git reset HEAD <文件>..." 以取消暂存）

    修改：      text.txt

ijeff@Rock:~/Desktop/python1901$ git commit -m "add 111"
[master 3af9e8e] add 111
 1 file changed, 1 insertion(+)
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
无文件要提交，干净的工作区
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git diff text.txt
diff --git a/text.txt b/text.txt
index 4892bab..cbb039d 100644
--- a/text.txt
+++ b/text.txt
@@ -1,3 +1,3 @@
 this is a text
 hello
-1111
+11112222
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git commit -m "222"
[master 55804ad] 222
 1 file changed, 1 insertion(+), 1 deletion(-)
ijeff@Rock:~/Desktop/python1901$ git diff text.txt
ijeff@Rock:~/Desktop/python1901$
```

4.1版本回退

工作原理：每当修改一个文件，并且使用commit提交之后，其实就相当于保存了一个快照

需求：回退到上一个版本

补充：要回退版本，首先需要知道当前处于哪个版本，在git中，用HEAD表示当前版本，上一个版本是HEAD^,上上个版本是HEAD^^,如果向上找100个版本，则表示为HEAD~100

git reset --hard 版本号【commit id】

```
演示命令：
ijeff@Rock:~/Desktop/python1901$ git log
commit 55804ad1c9bca1d98d366e28e50ecbfac2c82597 (HEAD -> master)
Author: ijeff-git <123456@163.com>
Date:   Fri Jun 29 10:31:55 2018 +0800

    222

commit 3af9e8eeb2d9d5dfb04e52968c255d0dd5e3e8ee
Author: ijeff-git <123456@163.com>
Date:   Fri Jun 29 10:28:47 2018 +0800
```

```
    add 111

commit ce63cb0594622577d144372d8c48dcabee631973
Author: ijeff-git <123456@163.com>
Date:   Fri Jun 29 10:05:22 2018 +0800

    create twofiles

commit 1f12c8b33abea0c7c0ae195aaa7ef18894681137
Author: ijeff-git <123456@163.com>
Date:   Fri Jun 29 10:03:26 2018 +0800

    add hello

commit d4b0bde029497a68dcacb026fd299b90c0604116
Author: ijeff-git <123456@163.com>
Date:   Fri Jun 29 10:00:33 2018 +0800

    create a new file and init

ijeff@Rock:~/Desktop/python1901$ git log --pretty=oneline
55804ad1c9bca1d98d366e28e50ecbfac2c82597 (HEAD -> master) 222
3af9e8eeb2d9d5dfb04e52968c255d0dd5e3e8ee add 111
ce63cb0594622577d144372d8c48dcabee631973 create twofiles
1f12c8b33abea0c7c0ae195aaa7ef18894681137 add hello
d4b0bde029497a68dcacb026fd299b90c0604116 create a new file and init
ijeff@Rock:~/Desktop/python1901$ git reset --hard HEAD^
HEAD 现在位于 3af9e8e add 111
ijeff@Rock:~/Desktop/python1901$ cat text.txt
this is a text
hello
1111
ijeff@Rock:~/Desktop/python1901$ git log --pretty=oneline
3af9e8eeb2d9d5dfb04e52968c255d0dd5e3e8ee (HEAD -> master) add 111
ce63cb0594622577d144372d8c48dcabee631973 create twofiles
1f12c8b33abea0c7c0ae195aaa7ef18894681137 add hello
d4b0bde029497a68dcacb026fd299b90c0604116 create a new file and init
ijeff@Rock:~/Desktop/python1901$ git reset --hard 55804ad1
HEAD 现在位于 55804ad 222
ijeff@Rock:~/Desktop/python1901$ cat text.txt
this is a text
hello
11112222
ijeff@Rock:~/Desktop/python1901$ git reset --hard HEAD^
HEAD 现在位于 3af9e8e add 111
ijeff@Rock:~/Desktop/python1901$ git reflog
3af9e8e (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
55804ad HEAD@{1}: reset: moving to 55804ad1
3af9e8e (HEAD -> master) HEAD@{2}: reset: moving to HEAD^
55804ad HEAD@{3}: commit: 222
3af9e8e (HEAD -> master) HEAD@{4}: commit: add 111
ce63cb0 HEAD@{5}: commit: create twofiles

1f12c8b HEAD@{6}: commit: add hello
```

```
d4b0bde HEAD@{7}: commit (initial): create a new file and init
ijeff@Rock:~/Desktop/python1901$ git reset --hard 55804ad
HEAD 现在位于 55804ad 222
```

总结：

- HEAD指向的是当前版本，所以，git在历史的版本之间来回切换，使用git reset --hard commit id
- 切换版本前，可以使用git log查看提交历史记录，以便于确定回到哪个历史版本
- 要重返未来，用git reflog查看历史执行过的git操作，从上往下寻找第一个commit的操作，则是未来的最新的版本

4.2工作区和暂存区

工作区：working Driectory，就是你电脑中的能看到的目录【python1901】

版本库：工作区中有一个隐藏的目录.git,该目录就是git中的版本库

版本库中存放了很多的数据，其中包括暂存区【缓存区，stage或者index】，还有git为我们自动创建的第一个分支master【主分支】，以及指向master的一个指针HEAD

往git版本库中添加文件，分为两步：

a.git add,实际是将文件添加到暂存区中

b.git commit .实际是将暂存区的文件提交到当前分支【主分支】

```
演示命令：
#a.修改text.txt文件
ijeff@Rock:~/Desktop/python1901$ vim text.txt
#b.在工作区中新增一个check.txt文本文件
ijeff@Rock:~/Desktop/python1901$ touch check.txt
#c.在check.txt文件中新增内容
ijeff@Rock:~/Desktop/python1901$ vim check.txt
#d.使用git status查看一下当前的状态
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
尚未暂存以备提交的变更：
  (使用 "git add <文件>..." 更新要提交的内容)
  (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

    修改：      text.txt

未跟踪的文件：
  (使用 "git add <文件>..." 以包含要提交的内容)

    check.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
#Git非常清楚地告诉我们，text.txt被修改了，而check.txt还从来没有被添加过，所以它的状态是未跟踪

#e.将两个文件都添加到暂存区
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git add check.txt
#f.再次查看
ijeff@Rock:~/Desktop/python1901$ git status
```

```
位于分支 master
要提交的变更:
   (使用 "git reset HEAD <文件>..." 以取消暂存)

     新文件:    check.txt
     修改:      text.txt
#g.将两个文件提交到分支
ijeff@Rock:~/Desktop/python1901$ git commit -m "modify textxt and create a new file named
check"
[master 9885393] modify textxt and create a new file named check
 2 files changed, 2 insertions(+)
 create mode 100644 check.txt
#h.再次查看
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
无文件要提交，干净的工作区
ijeff@ijeff-virtual

#一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是干净的
```

4.3管理修改

> 注意：git跟踪管理的是修改，并非文件

```
演示命令:
3af9e8e (HEAD -> master) HEAD@{2}: reset: moving to HEAD^
55804ad HEAD@{3}: commit: 222
3af9e8e (HEAD -> master) HEAD@{4}: commit: add 111
ce63cb0 HEAD@{5}: commit: create twofiles
1f12c8b HEAD@{6}: commit: add hello
d4b0bde HEAD@{7}: commit (initial): create a new file and init
ijeff@Rock:~/Desktop/python1901$ git reset --hard 55804ad
HEAD 现在位于 55804ad 222
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ touch check.txt
ijeff@Rock:~/Desktop/python1901$ vim check.txt
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
尚未暂存以备提交的变更:
   (使用 "git add <文件>..." 更新要提交的内容)
   (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

     修改:      text.txt

未跟踪的文件:
   (使用 "git add <文件>..." 以包含要提交的内容)

     check.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git add check.txt

ijeff@Rock:~/Desktop/python1901$ git status
```

```
位于分支 master
要提交的变更:
   (使用 "git reset HEAD <文件>..." 以取消暂存)

     新文件:    check.txt
     修改:      text.txt

ijeff@Rock:~/Desktop/python1901$ git commit -m "modify textxt and create a new file named
check"
[master 9885393] modify textxt and create a new file named check
 2 files changed, 2 insertions(+)
 create mode 100644 check.txt
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
无文件要提交，干净的工作区
ijeff@Rock:~/Desktop/python1901$ cat text.txt
this is a text
hello
11112222
3333
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
要提交的变更:
   (使用 "git reset HEAD <文件>..." 以取消暂存)

     修改:      text.txt

ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git commit -m "add 44 & 55"
[master ad2c692] add 44 & 55
 1 file changed, 1 insertion(+)
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
尚未暂存以备提交的变更:
   (使用 "git add <文件>..." 更新要提交的内容)
   (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

     修改:      text.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
ijeff@Rock:~/Desktop/python1901$ git diff HEAD -- text.txt
diff --git a/text.txt b/text.txt
index e1fd18b..4ae8e5f 100644
--- a/text.txt
+++ b/text.txt
@@ -3,3 +3,4 @@ hello
 11112222
 3333
 44444
+55555

ijeff@Rock:~/Desktop/python1901$ git add text.txt
```

```
ijeff@Rock:~/Desktop/python1901$ git commit -m "add 55"
[master e233f10] add 55
 1 file changed, 1 insertion(+)
ijeff@Rock:~/Desktop/python1901$ git diff HEAD -- text.txt
ijeff@Rock:~/Desktop/python1901$
```

4.4撤销修改

a.修改了文件内容，但是还没有添加到暂存区

```
演示命令：
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
尚未暂存以备提交的变更：
   (使用 "git add <文件>..." 更新要提交的内容)
   (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

     修改：      text.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
ijeff@Rock:~/Desktop/python1901$ cat text.txt
this is a text
hello
11112222
3333
44444
55555
stupid boss
ijeff@Rock:~/Desktop/python1901$ git checkout -- text.txt                    #回到最近一次
git commit或git add时的状态
ijeff@Rock:~/Desktop/python1901$ cat text.txt
this is a text
hello
11112222
3333
44444
55555
```

b.不但修改了内容，还添加到了暂存区，但是还没有提交

```
演示命令：
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
要提交的变更：
   (使用 "git reset HEAD <文件>..." 以取消暂存)

     修改：      text.txt

ijeff@Rock:~/Desktop/python1901$ git reset HEAD text.txt
重置后取消暂存的变更：          #把暂存区的修改撤销掉
M    text.txt
```

```
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
尚未暂存以备提交的变更:
   (使用 "git add <文件>..." 更新要提交的内容)
   (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

     修改:       text.txt

修改尚未加入提交 (使用 "git add" 和/或 "git commit -a")
ijeff@Rock:~/Desktop/python1901$ git checkout -- text.txt      #丢弃工作区的修改
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
无文件要提交，干净的工作区
ijeff@Rock:~/Desktop/python1901$ cat text.txt
this is a text
hello
11112222
3333
44444
55555
```

c.直接将修改的内容提交到了版本库

实质：版本回退

```
演示命令:
ijeff@Rock:~/Desktop/python1901$ vim text.txt
ijeff@Rock:~/Desktop/python1901$ git add text.txt
ijeff@Rock:~/Desktop/python1901$ git commit -m "fehj"
[master 8ac0ac4] fehj
 1 file changed, 1 insertion(+)
ijeff@Rock:~/Desktop/python1901$ git reset --hard HEAD^
HEAD 现在位于 e233f10 add 55
ijeff@Rock:~/Desktop/python1901$ cat text.txt
this is a text
hello
11112222
3333
44444
55555
```

4.5删除文件

在git中，删除文件也是一个修改操作

```
演示命令:
ijeff@Rock:~/Desktop/python1901$ touch newfile.txt
ijeff@Rock:~/Desktop/python1901$ git add newfile.txt
ijeff@Rock:~/Desktop/python1901$ git commit -m "create new file"
[master f86d2bd] create new file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newfile.txt

ijeff@Rock:~/Desktop/python1901$ rm newfile.txt
```

```
ijeff@Rock:~/Desktop/python1901$ git tsatus
git: 'tsatus' 不是一个 git 命令。参见 'git --help'。

最相似的命令是
    status
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
尚未暂存以备提交的变更:
  (使用 "git add/rm <文件>..." 更新要提交的内容)
  (使用 "git checkout -- <文件>..." 丢弃工作区的改动)

    删除:       newfile.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
ijeff@Rock:~/Desktop/python1901$ git rm newfile.txt
rm 'newfile.txt'
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
要提交的变更:
  (使用 "git reset HEAD <文件>..." 以取消暂存)

    删除:       newfile.txt

ijeff@Rock:~/Desktop/python1901$ git commit -m "delete newfile"
[master d41fc15] delete newfile
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 newfile.txt
ijeff@Rock:~/Desktop/python1901$ git status
位于分支 master
无文件要提交，干净的工作区
```

5.远程仓库

5.1建立远程仓库的准备工作

步骤:

a.创建github账号

b.生成ssh key【秘钥，建立本地和网络之间的连接】

命令：ssh-keygen -t rsa -C "github的注册邮箱"

c.添加到github

d.检测是否添加成功

命令:ssh -T git@github.com

```
演示命令:
ijeff@Rock:~/Desktop/python1901$ ssh-keygen  -t   rsa  -C    "123456@163.com"      #生成ssh key
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ijeff/.ssh/id_rsa):

Created directory '/home/ijeff/.ssh'.
```

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ijeff/.ssh/id_rsa.
Your public key has been saved in /home/ijeff/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:o/KqM3kWKnQBZL8xGGyxhO6n82YJpN6YQfTcZMbp/3E 123456@163.com
The key's randomart image is:
+---[RSA 2048]----+
|+*. . .          |
|+==  *           |
|+o++*            |
| + ++o           |
|=  .. . S        |
|.= o.  o o E     |
|o Xoo.. . o      |
|.*==oo   .       |
| .*B...          |
+----[SHA256]-----+
ijeff@Rock:~/Desktop/python1901$ cd ../..
ijeff@Rock:~$ cd .ssh/
ijeff@Rock:~/.ssh$ ls
id_rsa  id_rsa.pub
ijeff@Rock:~/.ssh$ cat id_rsa.pub      #查看生成的公钥
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCwiwfMpduAI7BP7ItaoZ8iyqWwN4io6ErpF9wNWLHXHrKSgR5A/tcVW5T90/q
7v/HDgMQ8uBcF+WLBKamYVG31mj43yu2io/FRmEoSHKbRW0Q5aBtlu9bnvUsiJVI4F8eI9O9/b1iiAaQBta5swEkUUq
J6e2ZpKxeye5HU5Hgm36zEjT4EIflMr4ox/+WQmVYLf9lA7IMvYpmRd3cqtoGoxCdcmUu3T/hi9anYhlrsUxda46w+X
C2ckZ1K0ZE3lrQOs0HQX9f4HQrKZupar8+GYst3dBQZz+V1YP7QmfMlSB0sM2s32SurSSO4ae2A3Oho8TPvuhJLp6Ou
fr+McK4j 123456@163.com
```

5.2.将本地仓库和远程仓库联系起来

演示命令：

```
chenxushu@chenxushu:~$ cd Desktop/
chenxushu@chenxushu:~/Desktop$ cd python1901/
chenxushu@chenxushu:~/Desktop/python1901$ git remote add origin1
git@github.com:chenxushu1025/python1901.git   #建立连接
chenxushu@chenxushu:~/Desktop/python1901$ git push -u origin1 master
对象计数中: 22, 完成.       #推送master分支并与远程master产生关联
Delta compression using up to 2 threads.
压缩对象中: 100% (13/13), 完成.
写入对象中: 100% (22/22), 1.79 KiB | 918.00 KiB/s, 完成.
Total 22 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To github.com:chenxushu1025/python1901.git
 * [new branch]      master -> master
分支 'master' 设置为跟踪来自 'origin1' 的远程分支 'master'。
chenxushu@chenxushu:~/Desktop/python1901$ ls
text1.txt  text2.txt  text.txt
chenxushu@chenxushu:~/Desktop/python1901$ vim text1.txt
chenxushu@chenxushu:~/Desktop/python1901$ git add text1.txt

chenxushu@chenxushu:~/Desktop/python1901$ git commit -m "modify text1.txt"
```

```
[master 69f0343] modify text1.txt
 1 file changed, 1 insertion(+)
chenxushu@chenxushu:~/Desktop/python1901$ git push origin1 master
对象计数中: 3, 完成.
Delta compression using up to 2 threads.
压缩对象中: 100% (2/2), 完成.
写入对象中: 100% (3/3), 308 bytes | 308.00 KiB/s, 完成.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:chenxushu1025/python180
```

总结:

- 要关联一个远程仓库，使用命令 git remote add origin git@github.com:username/repoName.git
- 关联成功之后，使用命令git push -u origin master第一次推送master分支的内容到远程仓库
- 以后，每次本地提交之后，只需要使用命令git push origin master推送最新的修改【本地修改----》add到暂存区---》commit到本地仓库----》push到远程仓库】

5.3从远程仓库克隆

git clone git@github.com:username/repoName.git

```
演示命令:
chenxushu@chenxushu:~/Desktop/python1901$ cd ..
chenxushu@chenxushu:~/Desktop$ git clone git@github.com:chenxushu1025/clonegithub.git   #从远程仓库克隆
正克隆到 'clonegithub'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
接收对象中: 100% (3/3), 完成.
chenxushu@chenxushu:~/Desktop$ cd clonegithub/
chenxushu@chenxushu:~/Desktop/clonegithub$ ls
README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ vim README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git add README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git commit -m "modefy"[master 4d3b04a] modefy
 1 file changed, 2 insertions(+), 1 deletion(-)
chenxushu@chenxushu:~/Desktop/clonegithub$ git push origin master
Warning: Permanently added the RSA host key for IP address '52.74.223.119' to the list of known hosts.
对象计数中: 3, 完成.
写入对象中: 100% (3/3), 258 bytes | 258.00 KiB/s, 完成.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:chenxushu1025/clonegithub.git
   24ee8e4..4d3b04a  master -> master

#克隆到本地之后，就可以任意修改本地工作区中的文件，修改完成之后，首先将修改add到暂存区，然后将暂存区中的修改提交到本地仓库，最后将本地仓库中的修改推送到远程仓库
```

6.分支管理

6.1.创建和合并分支

```
演示命令:
```

```
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout -b dev
切换到一个新分支 'dev'
chenxushu@chenxushu:~/Desktop/clonegithub$ git branch
* dev
  master
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout master
切换到分支 'master'
您的分支与上游分支 'origin/master' 一致。
chenxushu@chenxushu:~/Desktop/clonegithub$ git branch
  dev
* master
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout dev
切换到分支 'dev'
chenxushu@chenxushu:~/Desktop/clonegithub$ vim README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git add README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git commit -m "111"
[dev 122ae1f] 111
 1 file changed, 1 insertion(+)
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout master
切换到分支 'master'
您的分支与上游分支 'origin/master' 一致。
chenxushu@chenxushu:~/Desktop/clonegithub$ cat README.md
# clonegithub
hello
chenxushu@chenxushu:~/Desktop/clonegithub$ git merge dev
更新 4d3b04a..122ae1f
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
chenxushu@chenxushu:~/Desktop/clonegithub$ cat README.md
# clonegithub
hello
1111
chenxushu@chenxushu:~/Desktop/clonegithub$ git branch -d dev
已删除分支 dev（曾为 122ae1f）。
```

总结：

- git鼓励大量使用分支
- 查看分支：git branch
- 创建分支：git checkout -b name 【-b创建分支并且同时切换到子分支下】
- 切换分支：git checkout name
- 合并某个子分支到当前分支：git merge name
- 删除分支：git branch -d name

6.2解决冲突

```
演示命令：
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout -b feature1
切换到一个新分支 'feature1'
chenxushu@chenxushu:~/Desktop/clonegithub$ vim README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git add README.md

chenxushu@chenxushu:~/Desktop/clonegithub$ git commit -m "add hello and hi"
```

```
[feature1 0296b41] add hello and hi
 1 file changed, 1 insertion(+)
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout master
切换到分支 'master'
您的分支领先 'origin/master' 共 1 个提交。
  (使用 "git push" 来发布您的本地提交)
chenxushu@chenxushu:~/Desktop/clonegithub$ vim README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git add Re
fatal: 路径规格 'Re' 未匹配任何文件
chenxushu@chenxushu:~/Desktop/clonegithub$ git add README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git commit -m "add hello & hi"
[master 594d625] add hello & hi
 1 file changed, 1 insertion(+)
chenxushu@chenxushu:~/Desktop/clonegithub$ git merge feature1
自动合并 README.md
冲突（内容）：合并冲突于 README.md
自动合并失败，修正冲突然后提交修正的结果。
chenxushu@chenxushu:~/Desktop/clonegithub$ git status
位于分支 master
您的分支领先 'origin/master' 共 2 个提交。
  (使用 "git push" 来发布您的本地提交)

您有尚未合并的路径。
  (解决冲突并运行 "git commit")
  (使用 "git merge --abort" 终止合并)

未合并的路径：
  (使用 "git add <文件>..." 标记解决方案)

    双方修改：    README.md

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
chenxushu@chenxushu:~/Desktop/clonegithub$ vim README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git add README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git commit -m "fix conflict"
[master 5b37524] fix conflict
chenxushu@chenxushu:~/Desktop/clonegithub$ git log  --graph#合并后，我们用git log看看分支历史
```

总结：

- 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

6.3分支合并策略

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

演示命令：

```
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout -b dev

切换到一个新分支 'dev'
```

```
chenxushu@chenxushu:~/Desktop/clonegithub$ vim README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git add README.md
chenxushu@chenxushu:~/Desktop/clonegithub$ git commit -m "222"
[dev 8337c54] 222
 1 file changed, 1 insertion(+)
chenxushu@chenxushu:~/Desktop/clonegithub$ git checkout master
切换到分支 'master'
您的分支领先 'origin/master' 共 4 个提交。
  （使用 "git push" 来发布您的本地提交）
chenxushu@chenxushu:~/Desktop/clonegithub$ git merge --no-ff -m "222-1" dev
#准备合并dev分支，请注意--no-ff参数，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。
Merge made by the 'recursive' strategy.
 README.md | 1 +
 1 file changed, 1 insertion(+)
chenxushu@chenxushu:~/Desktop/clonegithub$ git log --graph
```

总结：

- master分支应该是非常稳定的，仅仅使用master分支发布版本，平时不在上面干活
- 比如：每个人都在dev的分支上干活，每个人都有自己的分支，时不时的向dev上合并代码就可以了
- 比如：发布1.0版本，再把dev上的代码合并到master上面
- 合并分支时，加上--no-ff参数表示使用普通模式进行合并，合并之后可以查看历史 记录，而Fast-Forword快速模式没有历史记录