**一，安装Git：**

**1.1Linux上安装命令：**

sudo apt-get install [**Git**](http://lib.csdn.net/base/git)

**1.2在Windows上安装Git：**

使用Windows版的msysgit，官方下载地址：http://msysgit.github.io/,[点击进入官网](http://msysgit.github.io/),如果官网无法正常下载我这里有当前的最新版，已经上传到CSDN上，下载地址为：http://download.csdn[**.NET**](http://lib.csdn.net/base/dotnet)/detail/huangyabin001/7564005,[点击进入下载](http://download.csdn.net/detail/huangyabin001/7564005)

1.3安装完成进行配置：

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

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

因为Git是分布式版本控制系统，每个机器都需要一个标识，也就是：你的名字和Email地址。

**二，创建版本库**

**2.1创建资源库所在的目录，命令：**

$ mkdir learngit

$ cd learngit

$ pwd

/Users/michael/learngit

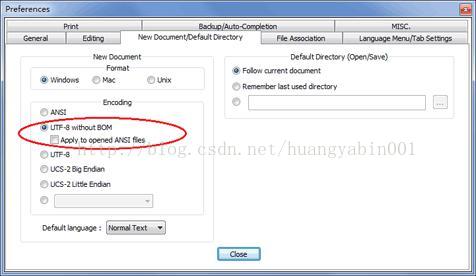
pwd命令用于显示当前目录完整路径，为了避免各种问题我们尽量避免路径命中出现中文字符。

**2.2通过git init命令把这个目录编程GIt可以管理的仓库：**

$ git init

Initialized empty Git repository in /Users/michael/learngit/.git/

！注意：版本控制系统只能跟踪文本文件的改动，例如txt文件、网页和所有的程序代码。版本控制器可以告诉你你的每次改动，但是图片、视频等二进制文件没办法跟踪，只知道文件大小的改变。在Windows下word格式也是二进制文件，因此我们如果要真正使用版本控制系统，就要以纯文本方式来编写文件，并且强烈建议使用标准的UTF-8编码。并且编辑文本文件我们推荐Notepad++，并要记得设置默认编码为UTF-8 without BOM.



**2.3添加文件到资源库**

**第一步：我们新建一个文本文件到我们的资源库learngit目录下。**

**第二步：试用git add命令告诉GIt，把文件添加到资源库**

$ git add test.txt

执行命令后，没有提示信息。

**第三步：用命令git commit告诉Git，把文件提交到仓库：**

$ git commit -m "wrote a test file"

说明：执行上述命令会打印提示信息如：

[master (root-commit) 3b15333] wrote a test file

 1 file changed, 1 insertion(+)

 create mode 100644 test.txt

git commit命令：-m后面输入的是本次提交的说明，可以输入任意有意义的内容，这样方便从历史记录中找到改动记录。

此外，我们可以同时添加很多文件，一起提交，例如：

$ git add file1.txt

$ git add file2.txt

$ git add file3.txt

$ git commit -m "add 3 files."

**三，版本回退**

**3.1修改文件**

原始文件中的内容为：this is my first  time to use Notepad++;

在原始文件中添加新的内容为：Add a new line。

**3.2使用git status命令查看状态：**

$ 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 d

        modified:   test.txt

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

执行git status命令，会打印提示消息。上述提示消息中告诉我们文件被修改，并没有提交。

**3.3使用git diff命令查看详细修改内容；**

$ git diff

diff --git a/test.txt b/test.txt

index d829b41..d6e3bba 100644

--- a/test.txt

+++ b/test.txt

@@ -1 +1,2 @@

-this is my first time to use Notepad

\ No newline at end of file

+this is my first time to use Notepad

+Add a new line。

\ No newline at end of file

**3.4使用git add test.txt 命令和git commit -m “add a new line.”提交修改；然后在文本中新添加内容“add a new line again”，以上述命令重新执行一边。  
3.5使用git log查看修改记录。**

$ git log

commit 4e8b0d0aaa685a83fb96fe52997e5af1e9e541ce

Author: bill <huangyabin001@163.com>

Date:   Sat Jun 28 13:48:29 2014 +0800

    add a new line again

commit d4d025a1cffaa761a7b82f39551465f7610a82db

Author: bill <huangyabin001@163.com>

Date:   Sat Jun 28 13:47:43 2014 +0800

    add a new line

commit 3b15333fdbb147f183a9d3013eadfafc9b05b127

Author: bill <huangyabin001@163.com>

Date:   Sat Jun 28 13:18:15 2014 +0800

    wrote a test file

上述命令git log执行后会打印出具体日志信息。从上述信息中我们可以得到每次提交的记录，记录中包含提交的描述性信息例如“wrote a test file”，提交时间，提交人的具体信息等。而“commit 3b15333fdbb147f183a9d3013eadfafc9b05b127”则是我们每次提交的提交版本号，也称之提交的记录ID。

如果我们不需要提交人，提交时间等信息，我们也可以以一种更简洁的方式查看日志，只需要加上“--pretty=online”参数即可。

$ git log --pretty=oneline

4e8b0d0aaa685a83fb96fe52997e5af1e9e541ce add a new line again

d4d025a1cffaa761a7b82f39551465f7610a82db add a new line

3b15333fdbb147f183a9d3013eadfafc9b05b127 wrote a test file

**3.6使用git reset命令回退版本**

在工作当中，我们不可避免的使用回退版本，例如一个模块负责人提交了一部分代码，在项目负责人发编译发布版本前离开了工作岗位，项目负责人在编译发布版本的时候发现模块负责人工作失误造成项目无法编译通过，为了不耽误整个版本发布的工作，负责人不得不回退版本。

$ git reset --hard HEAD^

HEAD is now at d4d025a add a new line

上述命令中Head在Git中的概念是一个指向你正在工作中的本地分支的指针（可以把HEAD想象为当前分支的别名），其所对应的分支本质是个指向commit对象的可变指针。截止到目前的学习中，在我们若干差提交后，我们已经有了一个指向最后一次提交的master分支，它在每次提交的时候都会自动向前移动。

我们使用git log --pretty=oneline查看：

$ git log --pretty=oneline

d4d025a1cffaa761a7b82f39551465f7610a82db add a new line

3b15333fdbb147f183a9d3013eadfafc9b05b127 wrote a test file

这个时候我们发现记录中已经少了一条，并且打开文件也会发现，最后修改的内容已经不见了。

还拿上面的例子来说，如果项目管理员发现编译不能通过的原因不是模块负责人的误操作引起的，而是有其他原因引起的，并且当前需要发布的版本中需要模块负责人的改动，但是版本已经回退了，能否再回退到回退前的版本呢？

答案是肯定的，我们只需要知道我们需要回退到的那个版本号commit id即可（例如当前的命令窗口没有关闭，我们可以轻轻滑动滚轴就可以看到之前的版本号），或者知道前面一部分也可以。

$ git reset --hard 4e8b0d0

HEAD is now at 4e8b0d0 add a new line again

读者可以使用git log进行查看，是否已经回退成功。  
但是如果当前的命令窗口已经关闭了，我们无法在命令窗口中查看我们之前打印的版本号了怎么办？git也为我们提供了一个命令来记录我们每次执行的命令“git reflog”

$ git reflog

4e8b0d0 HEAD@{0}: reset: moving to 4e8b0d0

d4d025a HEAD@{1}: reset: moving to HEAD^

4e8b0d0 HEAD@{2}: commit: add a new line again

d4d025a HEAD@{3}: commit: add a new line

3b15333 HEAD@{4}: commit (initial): wrote a test file

上述提示信息中最前面的字符串即是我们需要的版本号。

        总结：我们在电脑中能够看到的目录，例如我们新疆an的文件夹learngit文件夹就是一个工作区，而隐藏目录.git是git的版本库，在这个版本中的index文件（stage）是一个很重要的文件，我们称之为暂存区，git为我们自动创建的第一个分之master，以及指向master的一个指针称作HEAD。我们之前添加文件的操作“git add”，实际上是把文件修改添加到暂存区。而提交操作“git commit”则是把暂存区的所有内容提交到当前分支。  因此我们如果要提交修改，在提交前我们应该执行git add命令，把修改的文件添加到暂存区。

**四，撤销修改 git checkout**

我们不能绝对的保证在日常的工作中不会出任何差错，如果我们在提交代码前发前发现有错误，但是我们没有执行了git add命令把修改的文件添加到了暂存区，那么能否撤销此次修改呢？

答案也是肯定的。

$ git checkout -- test.txt

执行上述命令，没有任何提示消息。  
  
而且我们也可以使用命令git reset HEAD file把暂存区的修改撤掉（unstage），重新放回工作区。

git reset HEAD test.txt

**五，删除文件**

首先我们新键一个文件，并去删除它

如果我们没用把它提交到版本库我们可以在文件管理器手动删除或者使用命令"rm 文件名"的方式进行删除，而如果我们要删除版本库中的文件我们可以使用"git rm 文件名"的方式来操作。

$ git rm test1.txt

rm 'test1.txt'

**六，远程仓库**

我们知道Git是分布式[**版本控制**](http://lib.csdn.net/base/git)系统，同一个Git仓库，可以分不到不同的机器上，怎么分布呢？最早，有一台机器又一个原始版本库，此后其他机器进行克隆原始的版本，而且每台机器的版本库版本是一样的，没有主次之分。而且我们可以在充当“服务器”的机器上进行克隆，也可在同一台机器上克隆多个版本库，只要不在同一个目录下就好。

**6.1从“服务器”仓库克隆，使用github进行git存储**

**第一步：注册GItHub帐号，GitHub官网地址：https://github.com/ ，**[**点击打开**](https://github.com/)**。**

**第二步：创建SSH Key。在用户主目录下，如果有.ssh目录，并且该目录下有id\_rsa和id\_rsa.pub这两个文件，（跳过下一步操作），如果没有在windows打开Git Bash（linux 下打开Shell），创建SSH Key：**

$ ssh-keygen -t rsa -C "huangyabin001@163.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/STAR/.ssh/id\_rsa):

Created directory '/c/Users/STAR/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/STAR/.ssh/id\_rsa.

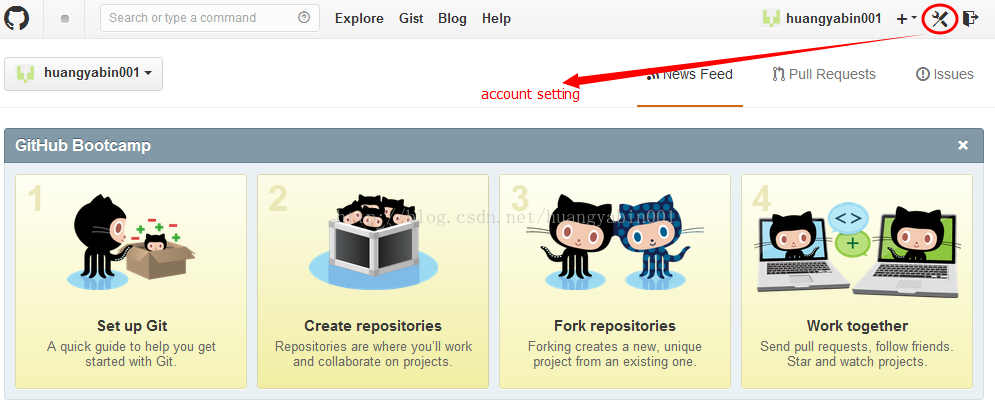
Your public key has been saved in /c/Users/STAR/.ssh/id\_rsa.pub.

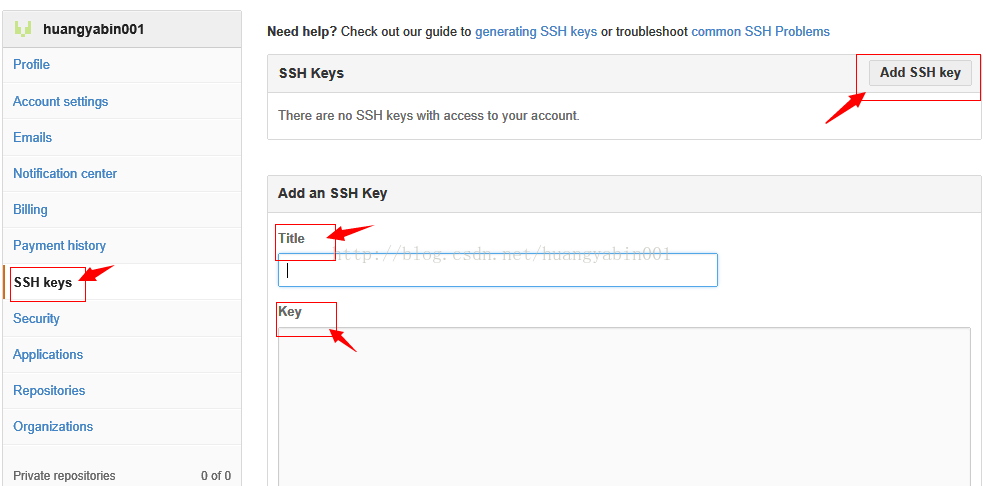
The key fingerprint is:

1e:49:1e:a4:fa:38:65:0e:4c:41:20:df:67:a2:0c:bf huangyabin001@163.com

回车，使用默认设置就好，无需设置密码。如果顺利在用户主目录下会看到id\_rsa和id\_rsa.pub两个文件，其中id\_rsa是私钥，不可泄露出去，而id\_rsa.pub是公钥，可以公开。

**第三步：登陆GitHub，打开Account settings，SSH Keys页面，并点击Add SH Key，添加SSH Key，Title可以自由定义，Key文本框中就是id\_rsa.pub文件的内容，直接复制即可。**

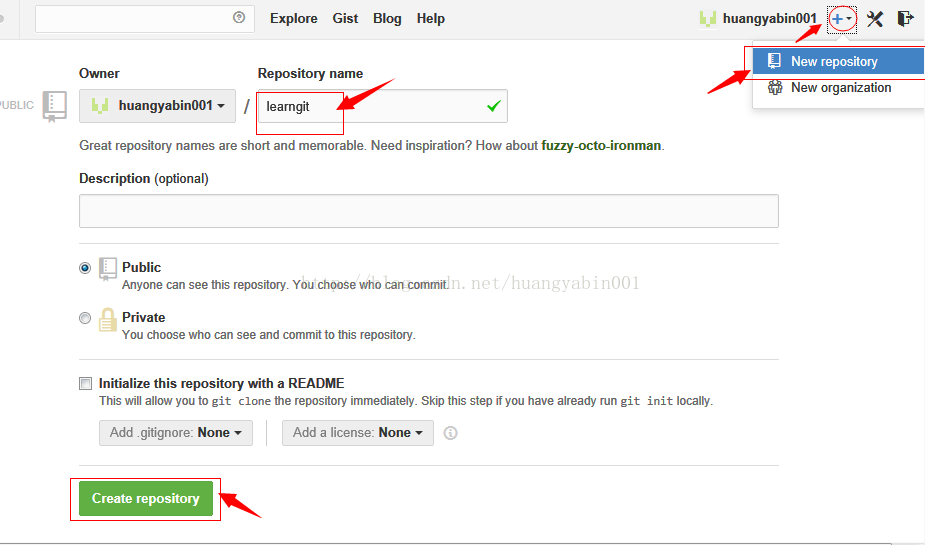




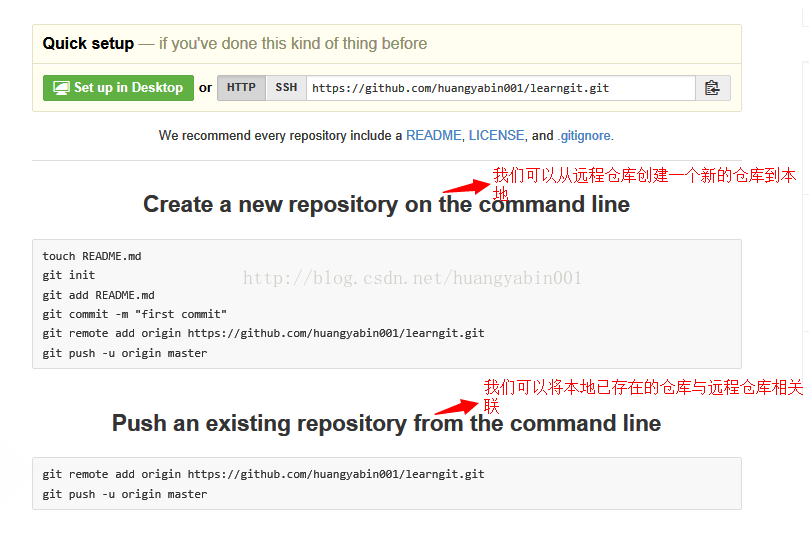
如果点击Add  SSH Key无反应，也就是没有弹出Title和Key的编辑框可能是浏览器的问题，换一个浏览器试试，360浏览器6.3的版本（其他版本没试，整的我开始还以为为是被墙掉了，后来验证一下才发现是浏览器的问题）就会出现这样的失误。

这里还需要解释一下，GitHub需要SSH Key的原因是为了确认确实是由你来提交的，而不是他人。

**第四步：在GitHub上点击Create a new repo按钮，创建一个新的仓库。**



创建成功后的界面：



**第五步，将本地资源库推送到远程仓库中。**

由于我们本地已经存在了一个仓库，我们可以根据上面的提示将本地的仓库推送到远程仓库中去。（请注意用户名正确，你自己的用户名）

$ git remote add origin https://github.com/huangyabin001/learngit.git

执行上述命令，没有任何信息提示；

接着执行

$ git push -u origin master

Username for 'https://github.com': huangyabin001

Password for 'https://huangyabin001@github.com':

Counting objects: 14, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (9/9), done.

Writing objects: 100% (14/14), 1.15 KiB | 0 bytes/s, done.

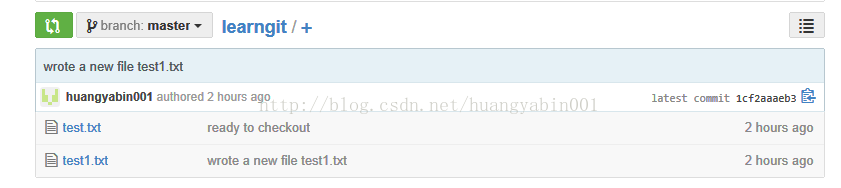
Total 14 (delta 2), reused 0 (delta 0)

To https://github.com/huangyabin001/learngit.git

 \* [new branch]      master -> master

Branch master set up to track remote branch master from origin.

在输入的过程中会要求你输入你在GitHub上的用户名和密码。根据提示进行操作即可。远程克隆到本地就不再赘述了，按照上面的命令进行执行即可。



刷新GitHub我们就可以看到我们push的仓库中的内容了。

**一、分支**

**1.1分支的概念。**

    对于的分支的理解，我们可以用模块化这个词来解释；在日常工作中，一个项目的开发模式往往是模块化，团队协作式的开发。这样我们项目的进度可以称得上多核并发式的开发了。这种模块化的开发要求我们尽可能的高内聚低耦合以免造成一只胳膊没了整个人都废了的局面。因此在所有的版本控制器对代码进行管理的时候都引入了分支这个概念。那么分支是什么呢？

    分支是相对于主干来说的，或者是相对于主分支来说的，它是用来将特性开发绝缘开来的。我们在创建仓库的时候，系统会默认创建master分支，也就是我们默认的主干分支。当我们开发一个项目的时候，在框架搭建完成后，需要开发一个一个模块的功能的时候，我们往往会创建一个一个的分支来进行分别开发，没跟人都在自己的一亩三分地里劳作，相互没有影响，当某一项功能或者模块开发完成通过测试的时候在整合到主框架上，这样做的好处是，可以避免单个模块的开发工作的缺陷造成整个框架系统无法编译通过，无法正常运转。如果你是一个Android开发人员，我们可以用主线程（UI）线程和子线程来理解，比较类似的是当我们创建一个Activity的时候，系统会默认创建一个主线程，也就是我们的UI线程，如果我们在需要访问网络获取数据的时候（耗时操作），我们一般的做法就是我们会重新开启一个子线程进行远程数据的获取与解析，当我们完成数据读取操作后在对UI线程进行更新，以免耗时操作造成UI线程的阻塞（ANR）。

**1.2创建分支与分支合并**

    我们每次执行提交的时候，git都会把它们串成一条时间线，这条时间线就是一个分支。经过前一阶段的学习，我们知道，在我们的仓库中只有一个主分支，也就是只有一条时间线，随着我们每一次的提交，master分支的时间想也就越长。当我们需要开发一个新的功能的时候，我们可以新建一个分支的时候（dev）来进行该功能模块的开发工作，Git就会同时新建了一个指针（dev），把HEAD指向dev分支，表示当前分支在dev上。那么对工作区的修改和提交就是在dev分支上了，我们每一次提交后，dev指针就会往前移动一次，不会影响到master分支。当我们在dev分支上的开发工作完成以后，通过测试验证后，再把dev分支与master进行合并。那么如何合并呢？我们可以让master分支指向dev分支（dev分支又指向当前分支下的最后一次提交）的当前提交，也就是把这个dev分支作为master主干分支的一次修改来进行提交，这样就完成了合并。合并完成后我们甚至可以删除被合并的dev分支。

    下面我们来进行实际的命令操作：

**1.2.1 创建分支**

$ git branch dev

！注意：执行上述命令，没有任何提示！

**1.2.2切换分支**

$ git checkout dev

Switched to branch 'dev'

！注意：

**1.2.3创建并切换分支命令**

使用git checkout命令加上-b参数表示创建并切换分支，相当于上述两条命令；

$ git checkout -b dev

**1.2.4查看当前分支**

$ git branch

\* dev

  master

绿色部分为当前分所在的分支。

STAR@STAR-PC ~/learngit (dev)

$ git add test.txt

STAR@STAR-PC ~/learngit (dev)

$ git commit -m "create new branch"

[dev cee7bfc] create new branch

 1 file changed, 2 insertions(+), 1 deletion(-)

STAR@STAR-PC ~/learngit (dev)  
$ git commit -m "create new branch"  
[dev cee7bfc] create new branch  
 1 file changed, 2 insertions(+), 1 deletion(-)

上述代码在dev分之下提交修改。

STAR@STAR-PC ~/learngit (dev)

$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 1 commit.

  (use "git push" to publish your local commits)

上述命令是我们在dev分之下条修改后进行的分支切换操作，此时我们查看工作区，则在dev分支下的修改并不存在。此时若想在master看到我们在dev分支下的修改，则需要合并分支。

$ git merge dev

Updating 94bf25d..cee7bfc

Fast-forward

 test.txt | 3 ++-

 1 file changed, 2 insertions(+), 1 deletion(-)

上述命令的执行完成了整合，此时查看工作区的文件发现，主分支下的文件已经能够看到之前在dev分支下的修改了。

注意！上述命令执行后打印的提示信息中的“Fast-forward”，告诉我们此次合并为“快进模式”也就是直接让master指针指向dev的当前提交。

**1.2.5删除分支**

由于之前的内容我们提到过，在分支合并后我们可以删除已经合并的分支，因此我们来删除dev分支，并查看所有分支。

STAR@STAR-PC ~/learngit (master)

$  git branch -d dev

Deleted branch dev (was cee7bfc).

STAR@STAR-PC ~/learngit (master)

$ git branch

\* master

提示信息告诉我们dev分支已删除，剩余分支为master，绿色表示当前分支为master

**1.3解决分支合并冲突**

你看到这个题目的时候或许心里会有疑惑？上述的操作不是很顺利吗？怎么会有冲突呢？

我们回头看看上述分支合并的操作就会发现，我们在合并分支的时候，新建分支dev有改动，而master分支没有提交任何修改，但是如果我们在合并分支的时候，master和dev分支均提交了修改呢？这样一来合并分支会还会一帆风顺吗？我们带着这个问题来进行接下来的操作。

我们首先创建一个分支dev，然后分别在两个分支上提交修改。

STAR@STAR-PC ~/learngit (master)

$ git checkout -b dev2

Switched to a new branch 'dev2'

STAR@STAR-PC ~/learngit (dev2)

$ git add test.txt

STAR@STAR-PC ~/learngit (dev2)

$ git commit -m "create a new brance dev2"

[dev2 046661c] create a new brance dev2

 1 file changed, 2 insertions(+), 1 deletion(-)

STAR@STAR-PC ~/learngit (dev2)

$ git switch master

git: 'switch' is not a git command. See 'git --help'.

STAR@STAR-PC ~/learngit (dev2)

$ git checkout master

Switched to branch 'master'

Your branch is ahead of 'origin/master' by 2 commits.

  (use "git push" to publish your local commits)

STAR@STAR-PC ~/learngit (master)

$ git add test.txt

STAR@STAR-PC ~/learngit (master)

$ git commit -m "add a new line for master"

[master 835e78c] add a new line for master

 1 file changed, 2 insertions(+), 1 deletion(-)

合并分支：

$ git merge dev2

Auto-merging test.txt

CONFLICT (content): Merge conflict in test.txt

Automatic merge failed; fix conflicts and then commit the result.

根据上述提示信息我们发现，test.txt文件发生冲突，合并失败。我们可以根据git status来查看冲突文件。

$ git status

On branch master

Your branch is ahead of 'origin/master' by 3 commits.

  (use "git push" to publish your local commits)

You have unmerged paths.

  (fix conflicts and run "git commit")

Unmerged paths:

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

        both modified:   test.txt

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

或许我们也可以直接查看test.txt的内容：

<<<<<<< HEAD

add a new line for master。

=======

create a new branch dev2.

>>>>>>> dev2

git用<<<<<<,======,>>>>>>标记出不同分支的内容，我们可以对文件进行修改如下；

add a new line for master。

create a new branch dev2.

并在master进行提交：

STAR@STAR-PC ~/learngit (master|MERGING)

$ git add test.txt

STAR@STAR-PC ~/learngit (master|MERGING)

$ git commit -m "fixed"

[master 51e165e] fixed

提示信息告诉我们问题已经解决。接着我们就可以删除dev2分支了。

$ git branch -d dev2

Deleted branch dev2 (was 046661c).

！注意：我们之前的分至合并操作都是快速模式下执行的，但是在这种模式下删除分支后，会丢失分支信息。因此我们在合并分支的时候也可以采用no-ff方式，如下，有兴趣的朋友可以自己进行测试。

$ git nerge --no-ff -m "merge with no-ff" dev

**1.4分支的隐藏与恢复**

如果我们在项目的开发过程中，需要暂时搁置当前分支的开发并在其他分之下进行操作，我们可以使用git stash对当前分支进行隐藏。

$ git checkout -b dev3

Switched to a new branch 'dev3'

STAR@STAR-PC ~/learngit (dev3)

$ git add test.txt

STAR@STAR-PC ~/learngit (dev3)

$ git commit -m "use stash"

[dev3 d358fab] use stash

 1 file changed, 2 insertions(+), 1 deletion(-)

查看状态，并执行git stash命令

$ git status

On branch dev3

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:   test.txt

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

STAR@STAR-PC ~/learngit (dev3)

$ git stash

Saved working directory and index state WIP on dev3: d358fab use stash

HEAD is now at d358fab use stash

切换回主分支进行提交修改操作。

$ git status

On branch master

Your branch is ahead of 'origin/master' by 6 commits.

  (use "git push" to publish your local commits)

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:   test.txt

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

STAR@STAR-PC ~/learngit (master)

$ git add test.txt

STAR@STAR-PC ~/learngit (master)

$ git commit -m "hello"

[master 404a601] hello

 1 file changed, 2 insertions(+), 1 deletion(-)

切换会dev3分支

STAR@STAR-PC ~/learngit (master)

$ git checkout dev3

Switched to branch 'dev3'

切换回分支之后，此时我们需要恢复现场。

首先查看隐藏的现场。

STAR@STAR-PC ~/learngit (dev3)

$ git stash list

stash@{0}: WIP on dev3: d358fab use stash

    恢复现场的方式有两种一种是使用git stash apply [stash@{0}]，但是恢复后stash内容并不会删除，我们需要手动执行git stash drop 来删除。

    第二种执行git stash pop，恢复的同时删除stash的内容。

$ git stash pop

On branch dev3

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:   test.txt

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

Dropped refs/stash@{0} (6696a348f1e160fa3f234dff50eaad0d59e4d264)

dev3分支下完成修改后，执行合并分支的操作。

！注意：如果我们开发完成一个分支，准备切换到主分支进行合并的时候，却发现该分支下的修改已经不需要了，这个时候我们如果要删除该分支我们需要执行git branch -D [分支名]

**1.5推送本地分支到远程分支**

    一般来说我们都会在本地分支上进行修改和提交，然后与主干分支进行合并，再删除无用分支，因此我们向远程分支进行推送的时候只需要推送主干分支即可。

$ git push -u origin master

Username for 'https://github.com': huangyabin001

Password for 'https://huangyabin001@github.com':

Counting objects: 31, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (19/19), done.

Writing objects: 100% (29/29), 2.23 KiB | 0 bytes/s, done.

Total 29 (delta 9), reused 0 (delta 0)

To https://github.com/huangyabin001/learngit.git

   1cf2aaa..7b69267  master -> master

Branch master set up to track remote branch master from origin.

   注意如果输入git push origin master则会出现一下问题：  
  
$ git push origin master  
fatal: unable to access 'https://github.com/huangyabin001/learngit.git/': Empty

reply from server

**二、自定义Git**

**2.1客户端配置**

**2.1.1 core.editor**

Git默认会调用你的环境变量editor定义的值作为文本编辑器，如果没有定义的话，会调用vi来创建和编辑，我们可以使用core.editor改变默认编辑器。

$ git config --global core.editor emacs

**2.1.2 help.autocorrect**

该配置只在Git1.6.1及以上版本有效，如果你错打了一条命令，会显示：

$git com tig:'com' is not a git-command.See 'git --help'. Did you mean this?commit

**2.2Git的着色**

Git能够为输出到你终端的内容着色，以便你可以凭直观的界面进行快速的分析。

Git会按照你的需要自动为大部分的输出加上颜色

$git config --global color.ui true