Git——创建版本库：

1、创建learngit文件夹

$ mkdir learngit

2、切换到learngit文件夹下

$ cd learngit

3、显示当前目录

$ pwd

4、把当前目录变成版本仓库

$ git init

5、把文件添加到版本库（前提应该已经有文件）

vi readme.txt

6、把文件添加到仓库，实际上就是把文件修改添加到暂存区

$ git add readme.txt

7、把文件提交到仓库，实际上就是把暂存区的所有内容提交到当前分支

$ git commit -m "wrote a readme file"

回退与未来：

1、了解仓库当前的状态

$ git status

2、具体修改了的内容

$ git diff readme.txt

3、查看提交历史

$ git log

4、简洁查看提交历史

$ git log --pretty=oneline

回退与未来——版本回退：

1、回退到上一个版本

$ git reset --hard HEAD^

2、回退到上一个版本

$ git reset --hard HEAD^^

3、回退到上一百个版本

$ git reset --hard HEAD~100

备注：之前的版本已经看不到了，只要上面的命令行窗口还没有被关掉，可以顺着往上找，指定回到未来的某个版本，版本号不用写全，前几位就可以。

4、查看提交历史（查看回退版本号）

$ git log

5、查看命令历史（查看未来版本号）

$ git reflog

回退与未来——工作区和暂存区：

工作区（Working Directory）就是在电脑里能看到的目录

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。

HEAD指向的版本就是当前版本

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。

查看工作区和版本库里面最新版本的区别：

$ git diff HEAD -- readme.txt

回退与未来——撤销修改：

$ git checkout -- readme.txt

一种是readme.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。

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

如果已经$ git add到暂存区

用命令$ git reset HEAD <file>可以把暂存区的修改撤销掉（unstage），重新放回工作区

再丢弃工作区的修改

$ git checkout -- readme.txt

Git知道删除了文件，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

$ rm test.txt

此时有两个选择：

一种情况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"

另一种情况是删错了，因为版本库里还有，所以可以把误删的文件恢复到最新版本：

$ git checkout -- test.txt

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”。

远程仓库：

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Shell（Windows下打开Git Bash），

创建SSH Key：

$ ssh-keygen -t rsa -C "984837776@qq.com"

2、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

远程仓库——添加远程库：

把本地仓库的内容推送到GitHub仓库：

关联：

$ git remote add origin git@github.com:CheatCode666/learngit.git

推送：

$ git push -u origin master

备注：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。

远程仓库——从远程库克隆：

$ git clone git@github.com:CheatCode666/gitskills.git

分支管理——创建与合并分支：

1、创建并切换到dev分支：

$ git checkout -b dev

相当于：

$ git branch dev

$ git checkout dev

2、查看分支：

$ git branch

3、在dev分支上进行文件操作后提交

$ git add readme.txt

$ git commit -m "branch test"

4、切换回master分支：

$ git checkout master

5、此时看不到在dev分支中的修改操作

把dev分支的工作成果合并到当前（master）分支上：

$ git merge dev

此时可以看到在dev分支中的修改操作

6、合并完成后可以删除dev分支：

$ git branch -d dev

7、查看分支：

$ git branch

分支管理——解决冲突：

1、准备新的feature1分支：

$ git checkout -b feature1

2、在feature1分支上进行文件操作后提交

$ git add readme.txt

$ git commit -m "AND simple"

3、切换到master分支：

$ git checkout master

4、在master分支上进行文件操作后提交

$ git add readme.txt

$ git commit -m "& simple"

5、此时合并两个分支会冲突，因为之间相隔1个提交

$ git merge feature1

$ git status

6、必须手动解决冲突后再提交：

$ git add readme.txt

$ git commit -m "conflict fixed"

7、用带参数的git log查看分支的合并情况：

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

8、最后，删除feature1分支：

$ git branch -d feature1

分支管理——分支管理策略（--no-ff）：

1、创建并切换dev分支：

$ git checkout -b dev

2、在dev分支上进行文件操作后提交

$ git add readme.txt

$ git commit -m "add merge"

3、切换回master：

$ git checkout master

4、合并dev分支，--no-ff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：

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

5、合并后，查看分支历史：

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

分支管理——Bug分支：

1、（情景）当前分支工作尚未完成

$ git status

2、隐藏当前工作现场：

$ git stash

3、干净的工作区：

$ git stash

4、切换需要修复bug的分支（假定是master分支）：

$ git checkout master

5、从master创建临时分支issue-101：

$ git checkout -b issue-101

5、在issue-101分支上进行修复操作后提交

$ git add readme.txt

$ git commit -m "fix bug"

6、切换到master分支，并完成合并，最后删除issue-101分支：

$ git checkout master

$ git merge --no-ff -m "merged bug fix 101" issue-101

$ git branch -d issue-101

7、回到dev分支继续完成隐藏的工作：

$ git checkout dev

8、查看隐藏的工作现场：

$ git stash list

9、两种方式恢复：

① 恢复：git stash apply stash@{0}

删除：git stash drop stash@{0}

② 恢复并删除：git stash pop

10、继续先前分支工作

分支管理——Feature分支：

1、创建feature-vulcan分支开发新功能：

$ git checkout -b feature-vulcan

2、 $ git add a.txt

3、 $ git status

4、 $ git commit -m "add feature vulcan"

5、切换dev分支，准备合并：

$ git checkout dev

6、因各种原因，销毁新功能（没有被合并过的分支）：

$ git branch -d feature-vulcan

error: The branch 'feature-vulcan' is not fully merged.

If you are sure you want to delete it, run 'git branch -D feature-vulcan'.

7、强制删除

$ git branch -D feature-vulcan

分支管理——多人协作：

查看远程库的信息：$ git remote

查看远程库详细的信息：$ git remote -v

远程仓库的默认名称是：origin

一、推送分支

1、把master分支上的所有本地提交推送到远程库（主分支）

$ git push origin master

2、把dev分支上的所有本地提交推送到远程库（开发分支）

$ git push origin dev

3、其他分支是否远程同步需要视情况而定

二、抓取分支

1、在本地电脑创建文件夹进行克隆：

$ cd /d/Git/

$ cd /d/Git/test/

$ git clone git@github.com:CheatCode666/learngit.git

$ cd /d/Git/test/learngit

2、该目录只能查看本地的master分支

$ git branch

3、test要在dev分支上开发，就必须创建远程origin的dev分支到本地，于是创建本地dev分支：

$ git checkout -b dev origin/dev

4、test在开发并push到远程：

$ git add a.txt

$ git commit -m "add a.txt"

$ git push origin dev

5、test已经向origin/dev分支推送了他的提交，碰巧你也对同样的文件作了修改，并试图推送（推送失败）：

$ cd /d/Git/learngit

$ git add a.txt

$ git commit -m "add a.txt"

$ git push origin dev

6、因为test的最新提交和你试图推送的提交有冲突，先用git pull把最新的提交从origin/dev抓下来，然后，在本地合并，解决冲突，再推送（失败）：

$ git pull

提示no tracking information

7、原因是本地dev分支与远程origin/dev分支的链接关系没有创建：

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

8、 $ git pull

9、必须手动解决冲突后再提交：

$ git add a.txt

$ git commit -m "fix a conflict"

$ git push origin dev

分支管理——Rebase：

1、先对test.txt这个文件做两次提交

2、用git log查看：

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

(HEAD -> master)和(origin/master)标识出当前分支的HEAD和远程origin的位置分别是582d922 add author和d1be385 init hello，本地分支比远程分支快两个提交。

3、尝试推送本地分支（失败）：

$ git status

4、有人先于我们推送了远程分支。按照经验，先pull一下：

$ git pull

5、做出合并动作后提交并看看状态

$ git status

加上刚才合并的提交，现在我们本地分支比远程分支超前3个提交。

6、用git log查看：

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

提交历史分叉了，即提交历史不是一条干净的直线。

7、此时可以输入命令git rebase：

$ git rebase

8、用git log查看：

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

原本分叉的提交现在变成一条直线了。

Git把我们本地的提交“挪动”了位置，放到了f005ed4 (origin/master) set exit=1之后，这样，整个提交历史就成了一条直线。rebase操作前后，最终的提交内容是一致的，但是，我们本地的commit修改内容已经变化了，它们的修改不再基于d1be385 init hello，而是基于f005ed4 (origin/master) set exit=1，但最后的提交7e61ed4内容是一致的。

9、把本地分支push到远程后用git log查看：

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

远程分支的提交历史也是一条直线。

标签管理——创建标签：

1、切换到需要打标签的分支上：

$ git branch

$ git checkout master

2、打一个新标签：

$ git tag v1.0

3、查看所有标签：

$ git tag

4、可以找到历史提交的commit id，然后打上标签：

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

$ git tag v0.9 xxxxxxx

5、查看所有标签：

$ git tag

6、查看标签信息：

$ git show v0.9

7、创建带有说明的标签：

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

8、查看标签信息：

$ git show v0.9

标签管理——操作标签：

1、删除标签：

$ git tag -d v0.1

2、推送某个标签到远程：

$ git push origin v1.0

创建的标签都只存储在本地，不会自动推送到远程

3、一次性推送全部尚未推送到远程的本地标签：

$ git push origin --tags

4、删除远程标签：

先从本地删除：git tag -d v0.9

再从远程删除：$ git push origin :refs/tags/v0.9

自定义Git:

自定义Git——忽略特殊文件:

自定义Git——配置别名:

自定义Git——搭建Git服务器: