$git config --global user.name “your name”

$git config --global user.email ”email@example.com” //注意git config命令的—global 参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置

Git简介：

$mkdir learngit //创建空目录learngit

$cd learngit //进入目录learngit

$cd ../.. //回退两级

$pwd //显示当前目录

$git init //把当前目录变成git仓库

$ls -ah //查看当前目录下隐藏文件

$git add readme.txt //将readme.txt文件添加到暂存库

$git commit -m “wrote a readme file”//将暂存库中的文件提交到仓库中（"内的是提交的说明”）

时光机穿梭：

$git status //显示仓库当前的状态

$git diff readme.txt //查看修改的difference

$git log //显示从近到远的提交日志

$git log – –pretty=oneline//在一行显示提交日志（版本号 说明）

$git reset –-hard HEAD^（）//将当前文件退回上个版本（HEAD^^为上上个版本 HEAD~100为上100个版本）

$git reset –-hard 1094a//将当前文件退回1094a版本

$cat readme.txt //查看文件内容

$git reflog //显示每一次命令

$git diff HEAD –- readme.txt//查看版本库和工作区里面最新版本的区别

$git checkout –- readme.txt//让readme.txt回到最近一次git commit或git add时的状态

//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”。

$git reset HEAD readme.txt//把暂存区的修改回退到工作区

$rm test.txt //删除目录中文件

$git rm test.txt //删除版本库中文件

远程仓库：

$git remote add origin [git@github.com:hhhcchhh/learngit.git//](mailto:git@github.com:hhhcchhh/learngit.git//)本地关联hhhcchhh的远程库origin

$git push -u origin master//将本地库内容推送到远程库origin上, 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。

$git push origin master//把本地master分支的最新修改推送至GitHub远程库origin

$git clone [git@github.com:hhhcchhh/gitskills.git//](mailto:git@github.com:hhhcchhh/gitskills.git//)将远程库gitskills克隆至本地

//实际上，Git支持多种协议，默认的git:// 使用ssh，但也可以使用https等其他协议。

在其他电脑上使用需要把SSH KEY添加到 Github

$ git remote -v //查看远程库信息

$ git remote rm <name>根据库名字删除 //此处的“删除”其实是解除了本地和远程的绑 定关系，并不是物理上删除了远程库。远程库 本身并没有任何改动。要真正删除远程库，需 要登录到GitHub，在后台页面找到删除按钮 再删除。）

分支管理：

$git branch //列出所有分支，当前分支前标\*

%git checkout master //切换回master 分支

$ git checkout -b dev //创建dev分支并切换=$git branch dev + $git checkout dev

$git merge dev //合并指定分支到当前分支（快速合并）

$git branch -d dev //删除dev分支

$git switch -c dev //创建并切换到新的dev分支（与第一个一样）

$git switch master //直接切换到已有的master分支

（解决冲突：

$git status //查看冲突的文件

$git log --graph (--pretty=oneline --abbrev-commit) //可以看到分支合并图

$ git merge --no-ff -m "merge with no-ff" dev//合并指定分支到当前分支--no-ff禁用fast forward（快速合并，会丢掉分支信息）

（分支策略：

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



（Bug分支

$git stash //储存当前工作，以后可以恢复现场

$git stash list //查看当前有的工作现场

$git stash apply //不删除stash 内容恢复

$git stash pop //删除stash 内容恢复

$ git stash apply stash@{0} //恢复指定的stash

$git cherry-pick 4c805e2 //复制特定的修改到当前分支

$git branch -D <name> //强行删除一个没合并的分支

（多人协作

$ git remote //查看远程库的信息

-v //显示更详细信息（可以抓取和推送的地址要 权限）

当你的小伙伴从远程库clone时，默认情况下，你的小伙伴只能看到本地的master分支。

$ git checkout -b dev origin/dev //将远程dev分支创建到本地

$ git branch --set-upstream-to=origin/dev dev//指定本地dev分支与远程origin/dev分支 的链接

$git pull //将最新的提交从origin/dev抓下来

多人协作的工作模式通常是这样：

首先，可以试图用git push origin <branch-name>推送自己的修改；

如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

如果合并有冲突，则解决冲突，并在本地提交；

没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

Rebase：

$git rebase //把分叉的提交历史“整理”成一条直线

$ git tag v1.0 //在一个分支上打一个新标签

$git tag //查看所有标签

$ git tag v0.9 f52c633 //对历史提交打标签

$ git show v0.9 //查看标签信息

$ git tag -a v0.1 -m "version 0.1 released" 1094adb//用-a指定标签名，-m指定说明文字

注意：标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。

$ git tag -d v0.1 //删除标签

$ git push origin v1.0 //推送标签到远程

$ git push origin –tags //推送全部尚未推送到远程的本地标签

$ git push origin :refs/tags/<tagname> //删除远程标签、

使用Gitee：

git remote add origin [git@gitee.com:liaoxuefeng/learngit.git](mailto:git@gitee.com:liaoxuefeng/learngit.git) //