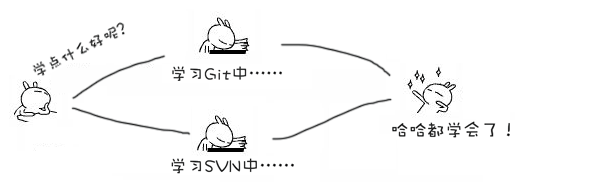
**惠普软件班实训课件**

**主讲：汤燕**

* 课程回顾GIT工具
* Git工具分支管理

# 分支管理

## 什么是分支



## 分支的好处

分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了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了。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。

现在有了分支，就不用怕了。你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，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

## Git分支的优势

其他版本控制系统如SVN等都有分支管理，但是用过之后你会发现，这些版本控制系统创建和切换分支比蜗牛还慢，简直让人无法忍受，结果分支功能成了摆设，大家都不去用。

但Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。

## 分支原理

每次提交，Git都把它们串成一条时间线，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。



HEAD指向当前分支。

创建一个新分支dev后,HEAD指向新分支。



在新的分支上修改代码。



假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！

合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



## 分支常用命令

Git鼓励大量使用分支

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>或者git switch <name>

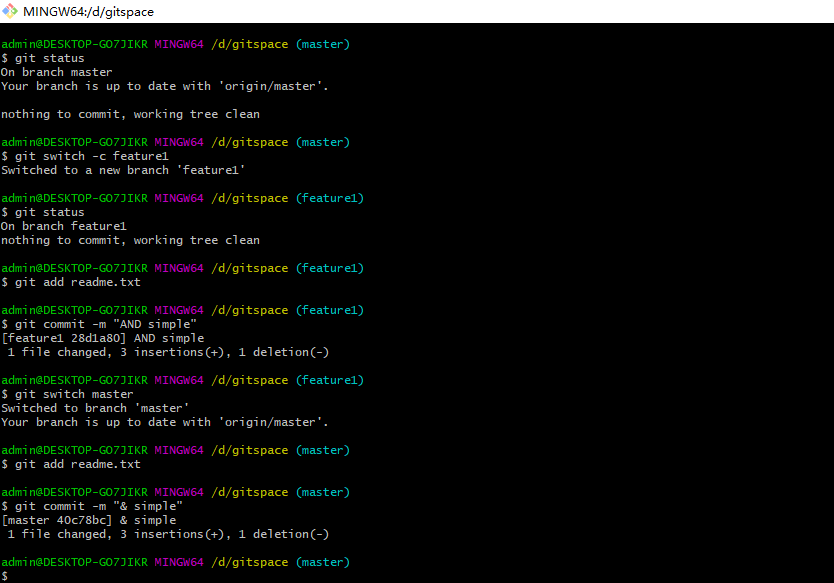
创建+切换分支：git checkout -b <name>或者git switch -c <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

## 手动解决冲突

人生不如意之事十之八九，合并分支往往也不是一帆风顺的。



创建新分支： git switch -c feature1

在分支添加新修改： git add readme.txt

在分支提交修改：git commit -m "AND simple"

切换到主分支：git switch master

在主分支添加修改：git add readme.txt

提交主分支修改：git commit -m "& simple"

当我们创建多个分支，每个分支都有修改，并且都修改的是同一个位置。

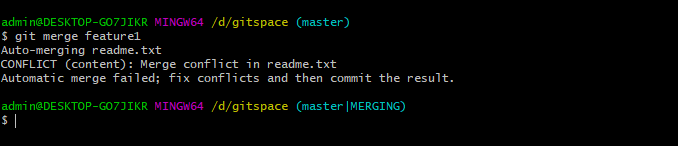
合并的时候就是这样：

现在，master分支和feature1分支各自都分别有新的提交，变成了这样：

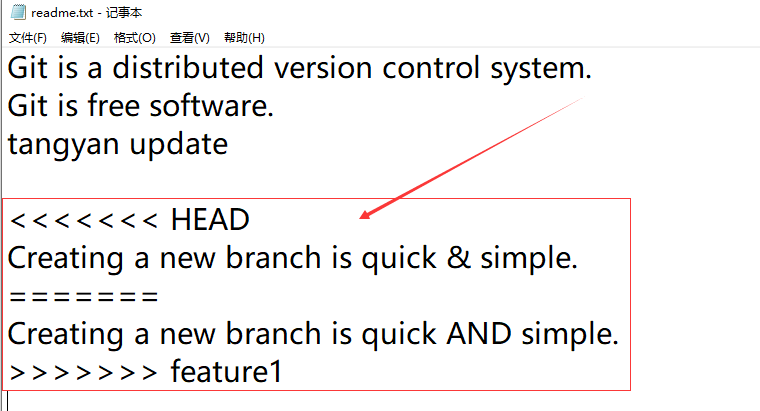


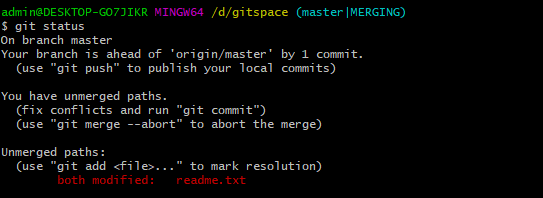
这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们试试看：

使用命令 git merge feature1 把feature1合并到主分支。

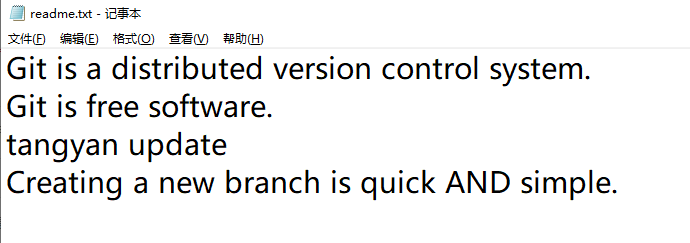


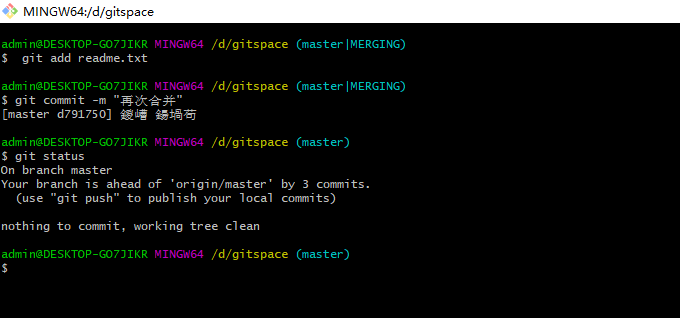
合并失败了。我们可以直接查看readme.txt的内容：





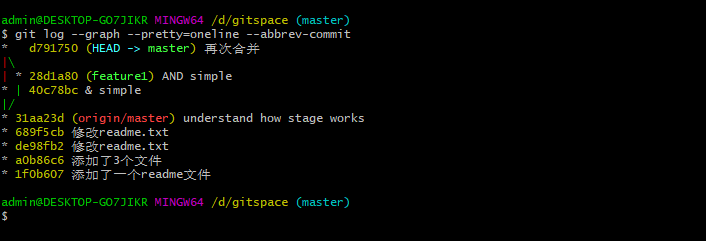
Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们手动修改如下后保存：





用带参数的git log也可以看到分支的合并情况：

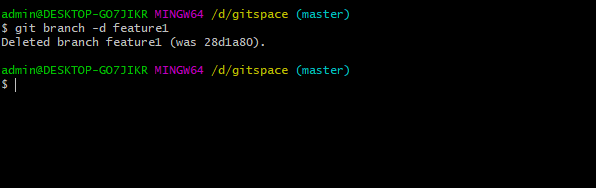
git log --graph --pretty=oneline --abbrev-commit





最后，删除feature1分支：

git branch -d feature1



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。

## 分支策略（了解）

通常，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。

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：



Git分支十分强大，在团队开发中应该充分应用。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

## Bug分支

修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场；

在master分支上修复的bug，想要合并到当前dev分支，可以用git cherry-pick <commit>命令，把bug提交的修改“复制”到当前分支，避免重复劳动。

## Feature分支

开发一个新feature，最好新建一个分支；

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

## 多人协作

要查看远程库的信息，用git remote。

用git remote -v显示更详细的信息。

## 推送分支

推送分支，就是把该分支上的所有本地提交推送到远程库。推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上：

git push origin master

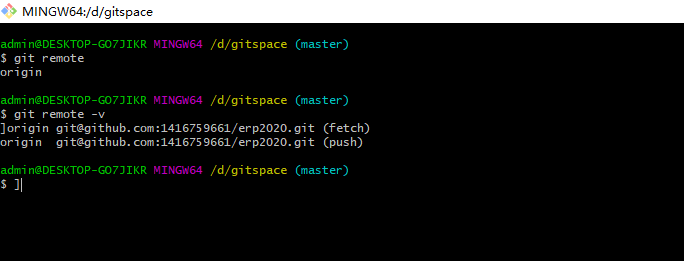
如果要推送其他分支，比如dev，就改成：

git push origin dev

但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，就是在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！



# Git实验操作步骤总结

第一步：在github.com上面注册账号。建立一个远程仓库 canvas.

第二步：下载git工具，并安装git工具，所有步骤均默认就可以。git下载地址：https://git-scm.com/downloads

第三步：在D盘根目录下建立一个项目文件夹canvas,把项目里需要上传的文件全部放到此文件夹下面。

第四步：在canvas文件夹上面点击右键，执行git bash here ,执行命令git init , canvas文件夹里面就会产生一个.git的文件夹，他就是本地仓库。

第五步：在打开的dos窗口里面执行

git config --global user.name "648744453"

git config --global user.email "648744453@qq.com"

作用就是配置当前是哪个用户在操作git.

第六步：执行 ssh-keygen -t rsa -f ~/.ssh/id\_rsa -C "648744453@qq.com" 会在c盘的用户文件夹里面生产一个.ssh的文件夹，找不到可以搜索。执行这个命令后会产生一个公钥和一个私钥。把公钥复制到github.com这个网站上面对应的项目里面。

第七步：在.ssh目录下面建一个没有后缀的config配置文件,里面的内容是：

Host github.com

HostName github.com

User git

PreferredAuthentications publickey

IdentityFile /C/Users/admin/.ssh/id\_rsa

第八步：

执行 eval $(ssh-agent)，

然后再执行ssh-add /C/Users/admin/.ssh/id\_rsa

第九步： 执行 git add . ,表示把当前所有的文件添加到暂存区

执行 git commit -m "提交备注" 把暂存区的代码提交到本地仓库

第十步： git remote add origin git@github.com:648744453/canvas.git

ssh -T github.com 测试本地仓库和远程仓库网络是否联通。

git push -u origin master 把本地仓库里面的文件推送到远程仓库。

git remote add origin git@github.com:1416759661/timeline.git

git fetch --all

git reset --hard origin/master

git pull

ssh-add id\_rsa文件的路径，把私钥添加到ssh列表中

如果执行ssh-add ...命令提示如下错误：

Could not open a connection to your authentication agent.

那么请执行eval $(ssh-agent)命令后再重试，如果还不行，请再执行ssh-agent bash命令后重试。

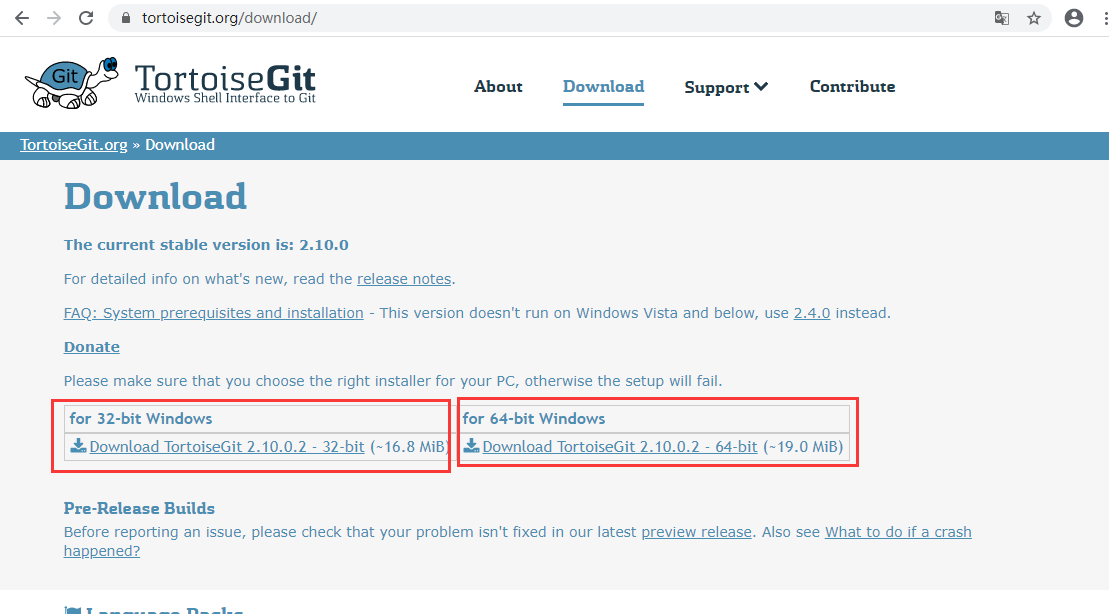
git remote add origin 路径

//git remote rm origin 删除

ssh -T github.com

git push -u origin master

# Git客户端可视化工具



## 下载地址

<https://tortoisegit.org/download/>

## 语言包安装

