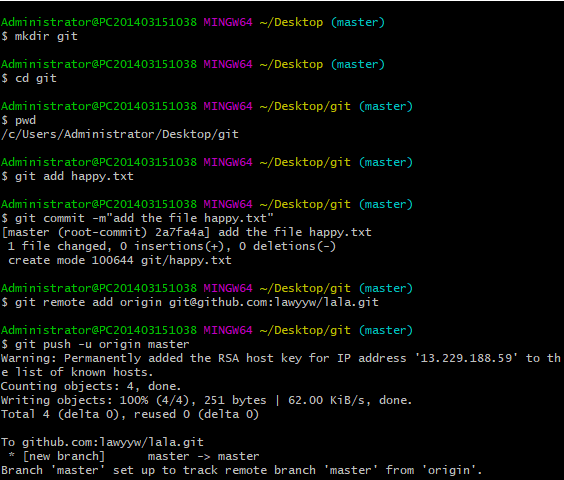
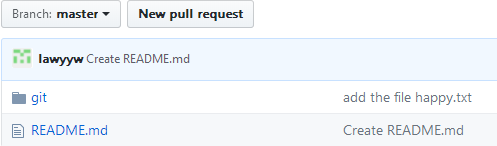
# 工作总结

### GitHub

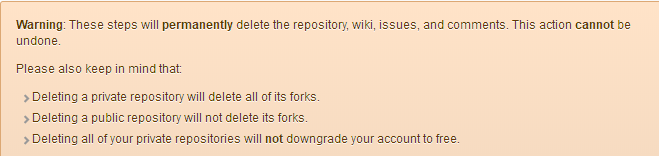
1.在本机与Github网站均重建了仓库，并进行了连接，终于操作成功！



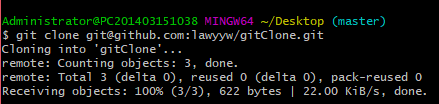


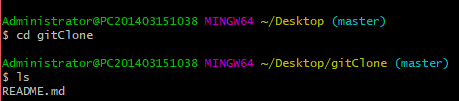
1. Clone

在进行克隆的时候，打算用第一个仓库git来进行操作，只不过它的命名与本机的git文件名重合，需要删除网站的repository或更改其名称

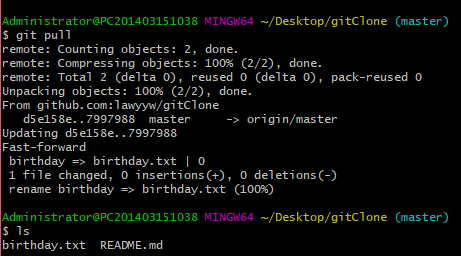


将其名称更改为gitClone，通过git clone命令已在桌面生成了gitClone文件夹，且含有README.md





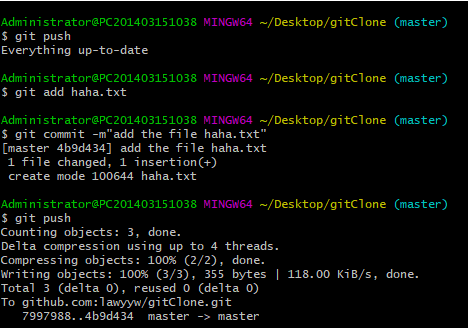
在网站中gitClone添加了birthday文本，通过git pull命令即可在本机中更新

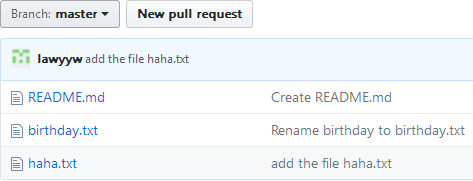


同理，在本机中增添新的文件，使用git push命令即可在网站中更新，只是一开始忘记对haha.txt进行git add和git commit命令，故显示

Everything up-to-date.

如果有多个人协作开发，那么每个人各自从远程克隆一份就可以了。





3.在[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000" \t "https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)里，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。

一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点：



每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长。

当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



假如我们在dev上的工作完成了，就可以把dev合并到master上。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：



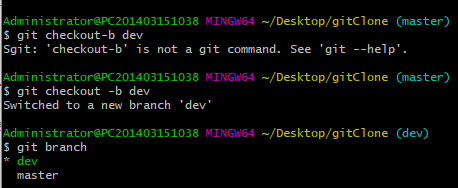
合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



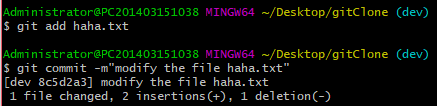
4.命令git checkout加上-b参数表示创建并切换，相当于以下两条命令：

创建分支：$ git branch

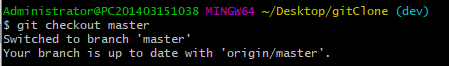
切换分支：$ git checkout

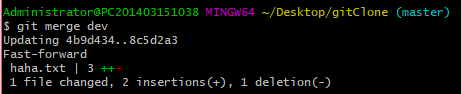


可以看到后缀也变成了(dev).

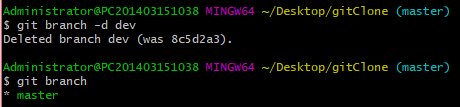


将分支重新切换至master后，再查看haha.txt，发现并没有修改痕迹，



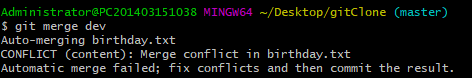


命令git merge用于合并指定分支到当前分支。合并后，再查看haha.txt的内容，就可以看到，和dev分支的最新提交是完全一样的，此时就可以删除分支dev（其中若未合并需要删除分支，需要命令：git branch -D dev）

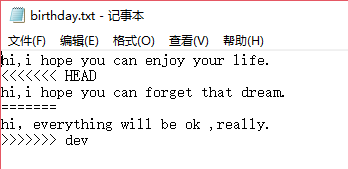


1. 冲突

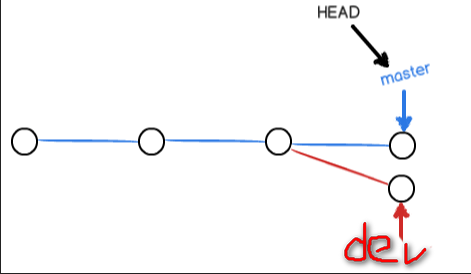
在新分支dev上修改了birthday.txt，并进行了提交命令；接着转向了master分支，仍修改了birthday.txt，进行了提交，此时进行git merge命令，反馈为：



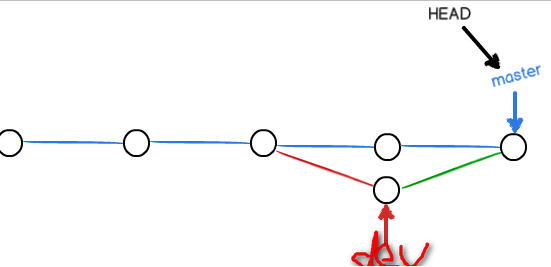
查看birthday.txt，显示：



此时分支情况为：



然后修改文件后再提交，分支变成了如下所示，最后可删除dev.



后来尝试了两次修改输入一样的内容也会出现冲突，在实际应用中应该不会出现两个人同时修改同一个文件，或者将一个文件放置在不同的分支里？

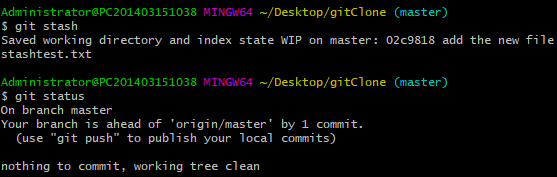
6.软件开发中经常会遇见bug，有了bug就需要修复。在Git中，每个bug都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

当接到一个修复bug的任务时，就会想创建一个分支来修复它，但是如果当前正在dev上进行的工作只进行到一半，还没有办法提交，但是必须在两个小时内修复该bug，怎么办？此时可以用命令git stash把当前工作现场“储藏”起来，等以后恢复现场后继续工作。

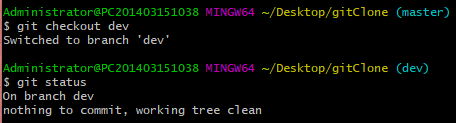
①在文件夹gitClone中新增了文件stashtest.txt，需要先进行add/commit命令，再操作git stash，否则显示结果：



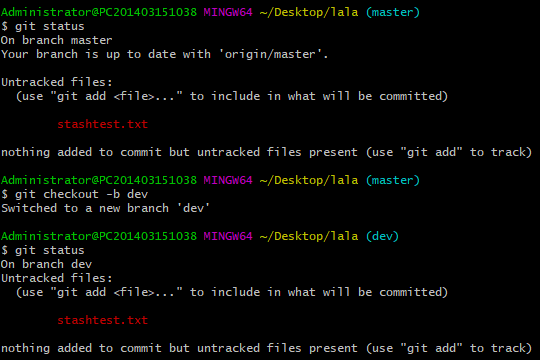
②在分支master中，修改了stashtest.txt，不必进行add/commit，然后使用命令git stash：



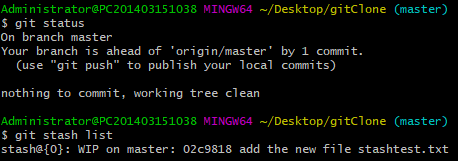
接下来创建了一个新分支dev，使用git status命令显示当前分支干净。



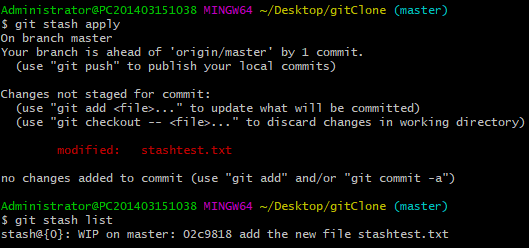
其中下图为未使用git stash命令显示的结果：



③切换至master，仍显示当前分支干净，需要其他命令来进行恢复。



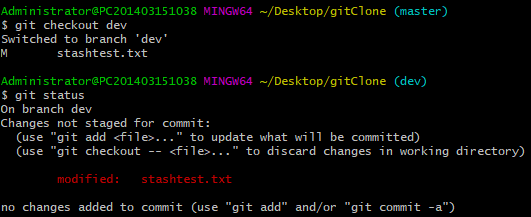
一是用命令git stash apply恢复，但是恢复后需要用命令git stash drop来删除stash的内容；二是用命令git stash pop，恢复的同时把stash内容删除；此时再用git stash list查看，就看不到任何stash内容了。



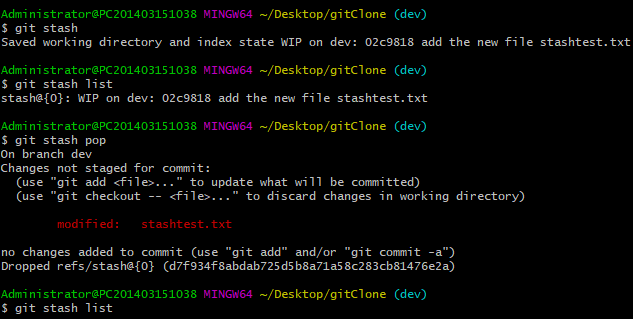




但再接着转向分支dev，仍会有未提交的新文件：



接着使用git stash命令来验证git stash pop：

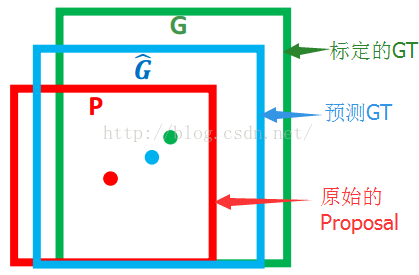


可以多次使用命令git stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令git stash apply stash@{0}

### BBOX:Bounding-Box

1. Bounding-box regression（边框回归）

对于窗口一般使用（x,y,w,h）四维向量来表示，分别表示窗口的中心点坐标和宽高。对于下图，红色的P表示原始的Proposal，绿色的框G表示目标的Ground Truth，我们的目标是寻找一种关系使得原始的窗口P经过映射得到一个跟真实窗口G更接近的回归窗口。

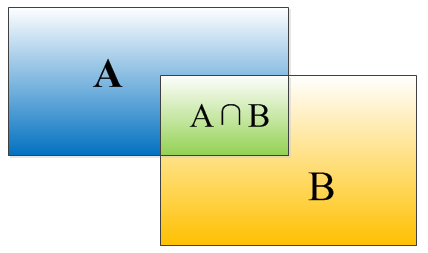


1. 物体检测需要定位出物体的bounding box，就像下面的图片一样，我们不仅要定位出车辆的bounding box，我们还要识别出bounding box 里面的物体就是车辆。对于bounding box的定位精度，有一个很重要的概念，因为我们算法不可能百分百跟人工标注的数据完全匹配，因此就存在一个定位精度评价公式：IOU。



IOU定义了两个bounding box的重叠度，如下图所示，矩形框A、B的一个重合度IOU计算公式为：IOU=(A∩B)/(A∪B)

就是矩形框A、B的重叠面积占A、B并集的面积比例:IOU=SI/(SA+SB-SI)



1. 定位一个车辆，最后算法找出了一堆的方框，我们需要判别哪些矩形框是没用的。

非极大值抑制：先假设有6个矩形框，根据分类器类别分类概率做排序，从小到大分别属于车辆的概率分别为A、B、C、D、E、F。

(1)从最大概率矩形框F开始，分别判断A~E与F的重叠度IOU是否大于某个设定的阈值;

(2)假设B、D与F的重叠度超过阈值，那么就扔掉B、D；并标记第一个矩形框F，是我们保留下来的。

(3)从剩下的矩形框A、C、E中，选择概率最大的E，然后判断E与A、C的重叠度，重叠度大于一定的阈值，那么就扔掉；并标记E是我们保留下来的第二个矩形框。

就这样一直重复，找到所有被保留下来的矩形框。

4.卷积神经网络(Convolutional Neural Networks, CNN)

神经认知机将一个视觉模式分解成许多子模式（特征），然后进入分层递阶式相连的特征平面进行处理，它试图将视觉系统模型化，使其能够在即使物体有位移或轻微变形的时候，也能完成识别。

视觉皮层的细胞存在一个复杂的构造。这些细胞对视觉输入空间的子区域非常敏感，我们称之为感受野，以这种方式平铺覆盖到整个视野区域。这些细胞可以分为两种基本类型，简单细胞和复杂细胞。简单细胞最大程度响应来自感受野范围内的边缘刺激模式。复杂细胞有更大的接受域，它对来自确切位置的刺激具有局部不变性。

CNN是一类特别设计用来处理二维数据的多层神经网络。CNN被认为是第一个真正成功的采用多层层次结构网络的具有鲁棒性的深度学习方法。CNN通过挖掘数据中的空间上的相关性，来减少网络中的可训练参数的数量，达到改进前向传播网络的反向传播算法效率，因为CNN需要非常少的数据预处理工作，所以也被认为是一种深度学习的方法。

5.selective search

目标识别（objec recognition）是指明一幅输入图像中包含哪类目标。其输入为一幅图像，输出是该图像中的目标属于哪个类别（class probability）。而目标检测（object detection）除了要告诉输入图像中包含了哪类目标外，还要框出该目标的具体位置（bounding boxes）。

在目标检测时，为了定位到目标的具体位置，通常会把图像分成许多子块（sub-regions / patches），然后把子块作为输入，送到目标识别的模型中。分子块的最直接方法叫滑动窗口法（sliding window approach）。滑动窗口的方法就是按照子块的大小在整幅图像上穷举所有子图像块。这种方法产生的数据量很大。和滑动窗口法相对的是另外一类基于区域（region proposal）的方法。selective search就是其中之一。

1. SVM

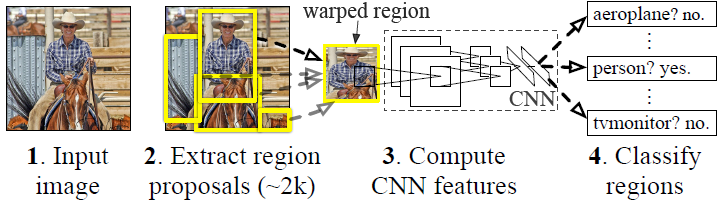
SVM(Support Vector Machine)指的是[支持向量机](https://baike.so.com/doc/6186041-6399292.html" \t "https://baike.so.com/doc/_blank)，是常见的一种判别方法。在机器学习领域，是一个有监督的学习模型，通常用来进行模式识别、分类以及回归分析。

SVM方法是通过一个非线性映射p，把[样本空间](https://baike.so.com/doc/6833412-7050620.html" \t "https://baike.so.com/doc/_blank)映射到一个[高维](https://baike.so.com/doc/1657186-1751734.html" \t "https://baike.so.com/doc/_blank)乃至无穷维的特征空间中(Hilbert空间)，使得在原来的样本空间中非线性可分的问题转化为在特征空间中的线性可分的问题，简单地说，就是升维和线性化升维，把样本向高维空间做映射，一般情况下这会增加计算的复杂性，甚至会引起“维数灾难”，因而人们鲜有问津。但是作为分类、回归等问题来说，很可能在低维样本空间无法线性处理的样本集，在高维特征空间中却可以通过一个线性超平面实现线性划分(或回归)。

一般的升维都会带来计算的复杂化，SVM方法巧妙地解决了这个难题：应用[核函数](https://baike.so.com/doc/3445751-3626034.html" \t "https://baike.so.com/doc/_blank)的展开定理，就不需要知道非线性映射的显式表达式。由于是在高维特征空间中建立线性学习机，所以与线性模型相比，不但几乎不增加计算的复杂性，而且在某种程度上避免了“维数灾难”。

### 《【目标检测】R-CNN算法详解》

1.R-CNN ( Regions with Convolutional Neural Network Features ) 可以说是利用深度学习进行目标检测的开山之作，简单来说，RCNN使用以下四步实现目标检测：   
a. 在图像中确定约1000-2000个候选框   
b. 对于每个候选框内图像块，使用深度网络提取特征   
c. 对候选框中提取出的特征，使用分类器判别是否属于一个特定类   
d. 对于属于某一特征的候选框，用回归器进一步调整其位置



2.使用Selective Search方法从一张图像生成约2000-3000个候选区域。

基本思路如下：   
①使用一种过分割手段，将图像分割成小区域；  
②查看现有小区域，合并可能性最高的两个区域。重复直到整张图像合并成一个 区域位置；  
③输出所有曾经存在过的区域，所谓候选区域；

其中候选区域生成和后续步骤相对独立，实际可以使用任意算法进行。

1. 合并规则

优先合并以下四种区域：   
①颜色（颜色直方图）相近的   
②纹理（梯度直方图）相近的   
③合并后总面积小的   
④合并后，总面积在其BBOX中所占比例大的

在合并中，应保证合并操作的尺度较为均匀，并且保证合并后的形状规则。

上述规则只涉及区域的颜色直方图、纹理直方图、面积和位置。合并后的区域特征可以直接由子区域特征计算而来，速度较快。

1. 为尽可能不遗漏候选区域，上述操作在多个颜色空间中同时进行。在一个颜色空间中，使用上述规则的不同组合进行合并。所有颜色空间与所有规则的全部结果，在去除重复后，都作为候选区域输出。

### 《基于R-CNN的物体检测》

物体检测和图片分类的区别：图片分类不需要定位，而物体检测需要定位出物体的位置，也就是相当于把物体的bbox检测出来，还有一点物体检测是要把所有图片中的物体都识别定位出来。

有监督预训练(Supervised pre-training)，又称迁移学习。比如你已经有一大堆标注好的人脸年龄分类的图片数据，训练了一个CNN，用于人脸的年龄识别。然后当你遇到新的项目任务是：人脸性别识别，那么这个时候你可以利用已经训练好的年龄识别CNN模型，去掉最后一层，然后其它的网络层参数就直接复制过来，继续进行训练。这就是所谓的迁移学习，说的简单一点就是把一个任务训练好的参数，拿到另外一个任务，作为神经网络的初始参数值,这样相比于你直接采用随机初始化的方法，精度可以有很大的提高。