# Linux下安装git

1. $ git
2. $ sudo
3. $ git config --global user.name “your name”
4. $ git config –glogal user.email “your email”
5. 因为git是分布式系统，所有每个机器都必须自报家门：名字和邮箱。如果有人冒充也是可以查的。
6. $ git config命令的 –global参数，表示这台机器的所有仓库都会用这个配置。当然，也可以对某个仓库指定不同的用户和email。

# 创建版本库

1. $ mkdir learngit
2. $ cd learngit
3. $ init
4. 只需要选择一个合适的地方创建一个空目录。然后初始化为一个仓库。
5. 仓库下的.git文件目录用来管理追踪版本库。
6. 所有的版本控制系统只能跟踪文本文件的修改（具体修改内容，比如第五行多了Linux），比如txt、网页、所有的程序代码等。而图片、视频这些二进制文件虽然可以跟踪版本变化，但是无法跟踪具体什么内容变化了。
7. 微软的word是二进制编码的；千万不要用windows自带的记事本编辑任何文件，原因是它保存utf8的策略“不合潮流”，会发生莫名其妙的错误。

# 添加文件到git仓库

1. $ git add file1.txt
2. $ git commit –m “本次操作备注”
3. $ git status ：查看仓库当前的状态。（哪些文件被修改、为add、commit等）
4. $ git diff readme.txt：自己和自己对比，查看具体修改的内容。（---表示修改前，+++修改后）
5. $ git diff HEAD – readme.txt：文件和本分支上的内容对比。

# 版本回退

1. $ git log：查看本地仓库的修改记录。
2. $ git log --pretty=oneline:将版本的一条历史修改记录放在同一行。
3. $ git reset –hard HEAD^：回退一个版本。
4. $ git reset –hard 12345（commit id） :当回退到过去后，又想回退未来的12345版本。
5. $ git reflog:查看你的每一次命令以及相应的commit id。

# 工作区和暂存区

1. 暂存区是git和其它版本控制系统如SVN不一样的地方。
2. 工作区：就是电脑的目录，.git所在文件夹，但是不包括.git本身，它是版本库。
3. 版本库：git的版本库是.git。版本库存了很多东西，其中最重要的是成为stage（或叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及只想master的一个指针叫HEAD。
4. $ git add file是把文件放到暂存区，commit是提交更改，把暂存区的所有内容提交到当前分支。（本质是stage的头指针指向master的末尾指针）

# 管理修改

1. Git比其它版本控制优秀的地方在于它跟踪并管理的是修改，而非文件。
2. 上述的修改指：修改—add—修改—commit,发现第二次的修改没有提交上去。
3. $ git diff HEAD – readme.txt:查看提交到分支的内容和工作区上的readme文件有什么不同。

# 撤销修改

1. $ git checkout – readme.txt:撤销readme文件的修改，无法撤销add、commit操作。
2. $ git checkout dev:创建一个dev分支。
3. 当文件没有add时，执行命令回到和版本库一模一样的状态。
4. 当文件add后没有commit时，执行命令回到add后的状态。
5. $ git reset HEAD readme.txt:把暂存区的修改撤销掉（unstage），即撤回add操作。

# 删除文件

1. 把工作区的文件先提交到版本库，然后删除工作区的文件。这时候git会提示哪些文件本删除。可以选择两种操作：
   1. 将版本库的文件一起删除并提交。
      1. $ git rm test.txt
      2. $ git commit –m “xxx”
   2. 从版本库恢复文件到工作区。其实是用版本库的版本替换工作区的版本，所以无论修改还是删除都可以。要注意的是，只能恢复到最近一次commit时的文件，无法恢复之后的修改。
      1. $ git checkout – test.txt

# 远程仓库

## 创建远程仓库

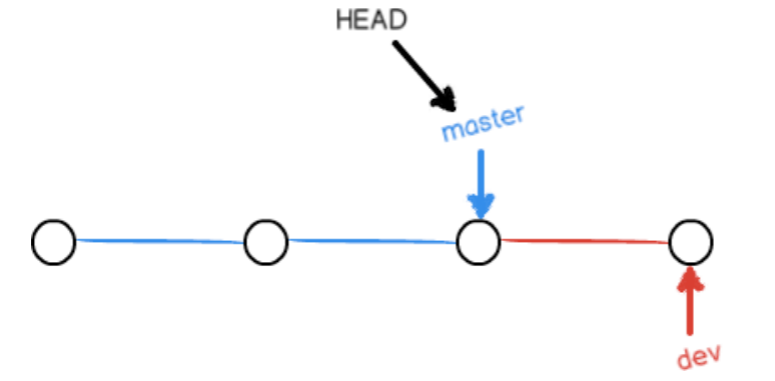
1. Ssh-keygen –t –rsa –C “your email”
2. 在用户主目录下找到id\_rsa、id\_rsa.pub，将公钥设置到github下。（Account settings—SSH Keys）

## 添加远程仓库

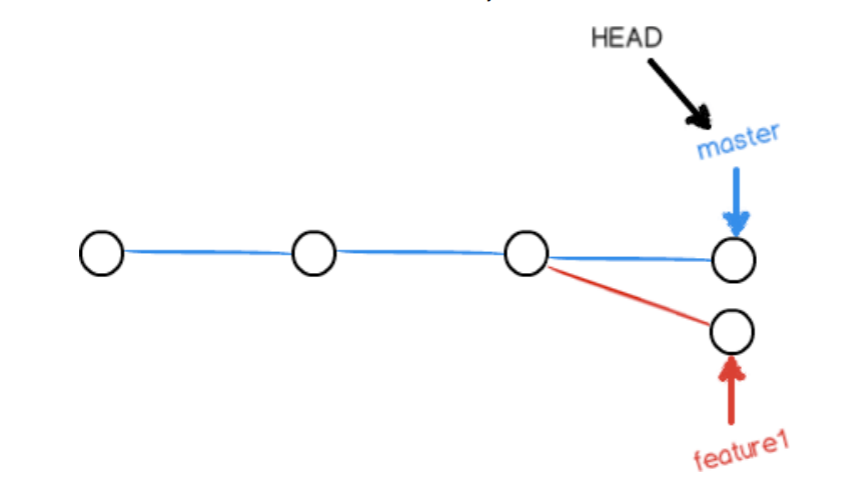
1. 在github处创建一个库,名字为learngit。（Creat a new repo）
2. $ git remote add origin [git@github.com:liningo/learngit.git](mailto:git@github.com:liningo/learngit.git)
3. $ git push –u origin master:将本地版本库推送到远程仓库。实际上是把当前分支推送到远程。因为第一次推送是空的，所以加了-u。这样就把 本地库和远程库关联起来了。以后直接push或pull。
4. $ git push origin master：第二次后的push操作。Origin是远程库名。
5. $ git clone [git@github.com:liningo/gitskills.git](mailto:git@github.com:liningo/gitskills.git):将远程库的gitskill克隆到本地。
6. Git默认使用ssh协议，还支持其它协议，比如https协议。但是使用https协议不仅传输速度慢，还每次需要输入口令。

# 分支管理

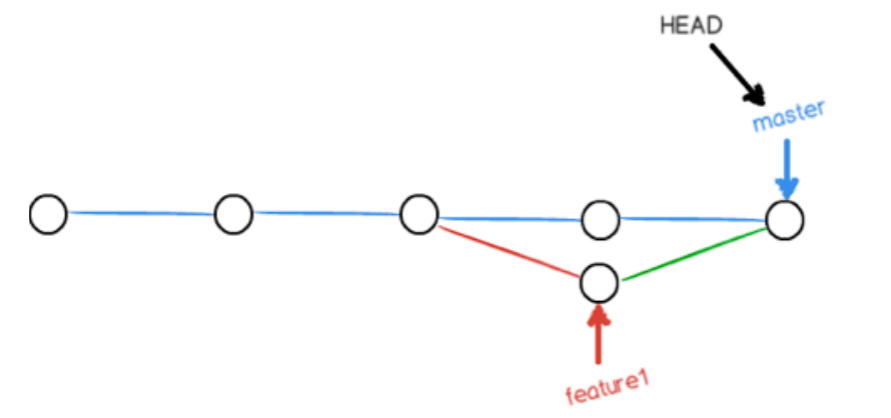
1. $ git checkout –b dev:创建分支dev
2. $ git branch：查看当前的所有分支。
3. $ git checkout master:切换回master分支。
4. $ git merge dev:将dev分支的内容合并到master分支中。其实移动相关指针。
5. $ git branch –d dev：删除dev分支。
6. Git鼓励大量使用分支。
7. 简单合并分支的解析图如下。



# 解决冲突

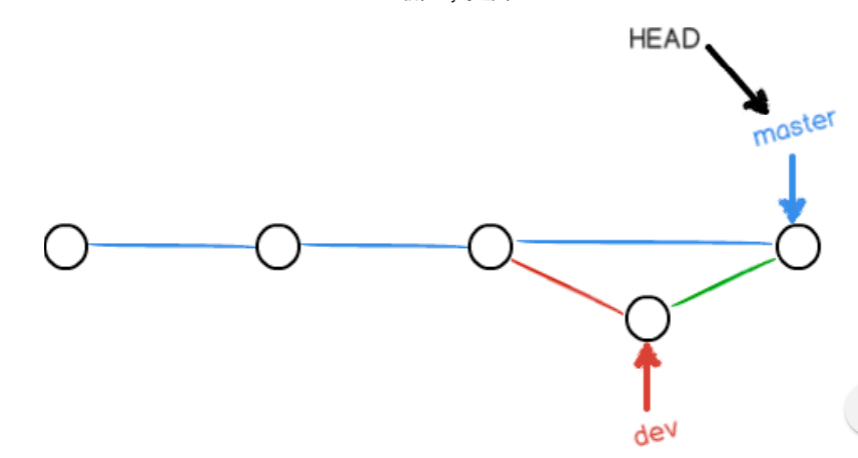


1. 发生如上冲突时，需要解决两个分支的差异文件内容，修改保存后再add、commit。
2. Commit 后分支结构图如下。
3. $ git log –graph –pretty=oneline –abbrev-commit查看分支的情况.

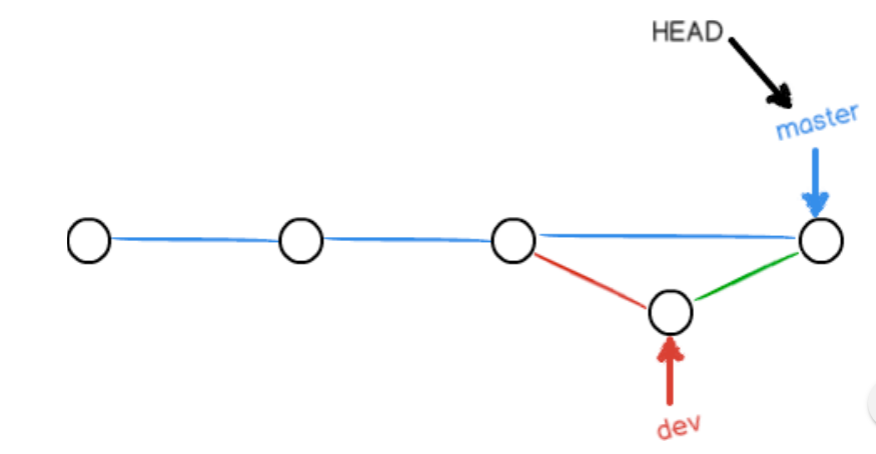


# 分支策略

1. Git默认使用Fast Forward方式合并分支。这个方式的坏处是会丢失合并的那个分支信息。在$ git log –graph… 下不能查看得到本次合并。示意图如下：

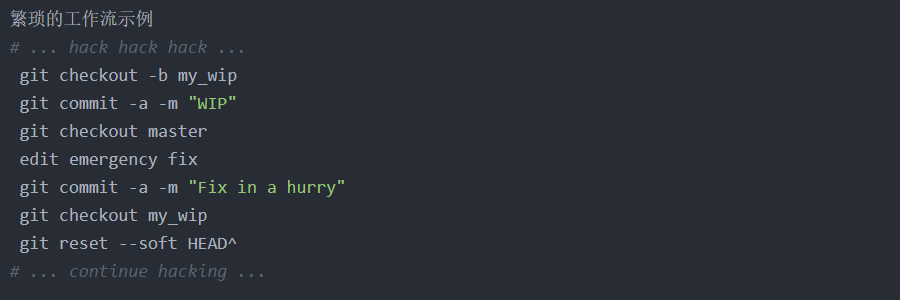


1. $ git merge –no-ff –m “xxx”：禁止使用Fast Forward方式。因为这样会生成commit id，所以需要使用-m注释本次合并。在$ git log –graph… 下能查看得到本次合并。示意图如下：

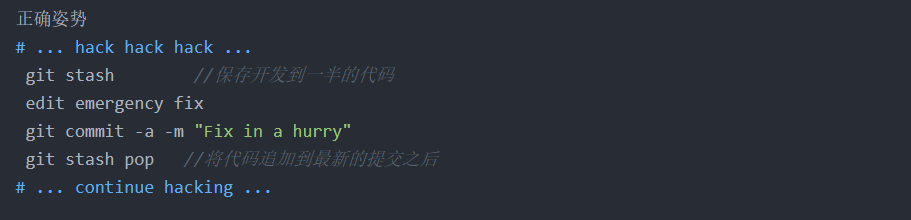


# Bug分支

1. 当开发进行到一半,老板过来跟你说"线上有个bug,你现在给我改好,不然扣你鸡腿".当然,你可以开一个新的分支,把当前代码提交过去,回头再merge,具体代码如下：



1. 我们可以通过git stash来简化这个流程：



1. $ git stash命令将当前工作保存起来，可以通过$ git stash list查看。stash@{0}是最新的，stash@{1}次之。
2. $ git stash pop:将保存起来的文件恢复到工作空间，同时删除stash内容。
3. $ git stash apply: 将保存起来的文件恢复到工作空间，但不删除stash内容。
4. $ git stash apply stash@{0}：恢复某一块stash内容。

# Feature分支

1. 开发一个新功能最好新建一个分支，避免打乱原分支。

# 多人协作

# 标签管理

1. 发布一个版本时，我们通常先在版本库中打一个标签，这样就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以标签也是版本库的一个快照。
2. Git的标签虽然是版本库的一个快照，但它其实就是一个指向某个commit的指针。这点和分支很像，但是指向分支的指针可以移动。
3. $ git tag v1.0:默认的标签是打在最新提交的commit上的。如果突然想起昨天提交的commit还没有打标签，可以先找到它的commit id，再打上标签。
4. $ git log –pretty=oneline –abbrev-commit; $ git tag v2 62231:给commit id为62231打上v9的标签。
5. $ git show v2:查看标签为v2的提交信息，包括commit id。
6. $ git tag -a v0.1 -m "version 0.1 released" 3628164：创建带有说明的标签。-a是标签名，-m是标签说明。
7. $ git tag -d v0.1：删除标签
8. $ git push origin v1.0：推送v1.0到远程。
9. $ git push origin –tags：一次性推送全部未推送到远程的标签。
10. 命令git push origin :refs/tags/tagname：可以删除⼀个远程标签（已提交标签）。

# 自定义git

1. $ git config --global alias.st status：告诉git，以后st就表示status。Global是全局参数，就是说所有这部电脑上都可以使用这个别名。

# Git整体工作流程

从一般开发者的角度来看，git有以下功能：

1、从服务器上克隆完整的Git仓库（包括代码和版本信息）到单机上。

2、在自己的机器上根据不同的开发目的，创建分支，修改代码。

3、在单机上自己创建的分支上提交代码。

4、在单机上合并分支。

5、把服务器上最新版的代码fetch下来，然后跟自己的主分支合并。

6、生成补丁（patch），把补丁发送给主开发者。

7、看主开发者的反馈，如果主开发者发现两个一般开发者之间有冲突（他们之间可以合作解决的冲突），就会要求他们先解决冲突，然后再由其中一个人提交。如果主开发者可以自己解决，或者没有冲突，就通过。

8、一般开发者之间解决冲突的方法，开发者之间可以使用pull 命令解决冲突，解决完冲突之后再向主开发者提交补丁。

# Git的分布式理解

1. 为什么说git是一个分布式的版本控制服务器。这是相对于它的核心功能即版本控制而言的。每一个人都可以拥有属于自己的仓库，然后进行相应的版本控制，最终将各自的版本合并到公司的仓库中。
2. 相对SVN来说，它只能在中心服务器上进行版本控制。

# 集中式和分布式的区别

1. Git就是分布式版本控制系统，对应的是集中式的版本控制如SVN。
2. 简单的说，分布式的版本控制就是每个人都可以创建一个独立的代码仓库用于管理，各种版本控制的操作都可以在本地完成。每个人修改的代码都可以推送合并到另外一个代码仓库中。
3. 而像SVN这样，只有一个中央控制，所有的开发人员都必须依赖于这个代码仓库。每次版本控制的操作也必须链接到服务器才能完成。
4. 很多公司喜欢用集中式的版本控制是为了更好的控制代码。如果个人开发，就可以选择Git这种分布式的。

# 分布式和集群的区别

1. 简单说，分布式是以缩短单个任务的执行时间来提升效率的，而集群则是通过提高单位时间内执行的任务数来提升效率。
2. 举例子说
   1. 如果一个任务由10个子任务组成，每个子任务单独执行需1小时，则在一台服务器上执行该任务需10小时。
   2. 采用分布式方案，提供10台服务器，每台服务器只负责处理一个子任务，不考虑子任务间的依赖关系，执行完这个任务只需一个小时。(这种工作模式的一个典型代表就是Hadoop的Map/Reduce分布式计算模型）
   3. 而采用集群方案，同样提供10台服务器，每台服务器都能独立处理这个任务。假设有10个任务同时到达，10个服务器将同时工作，1小时后，10个任务同时完成，这样，整身来看，还是1小时内完成一个任务！
3. 集群一般被分为三种类型，高可用集群如RHCS、LifeKeeper等，负载均衡集群如LVS等、高性能运算集群;分布式应该是高性能运算集群范畴内。
4. 分布式：不同的业务模块部署在不同的服务器上或者同一个业务模块分拆多个子业务，部署在不同的服务器上，解决高并发的问题。
5. 集群：同一个业务部署在多台机器上，提高系统可用性