安装git

Linux, 尝试输入git, 若提示没有安装; 直接安装: sudo apt-get install git-core或直下载源代码安装, ./config; make; sudo make install

Mac, homebrew或AppStore安装Xcode, Xcode > Preferences > Downloads > Command Line Tools > Install

Window, msysgit是windows版本的git; 完成后在开始菜单 Git > Git Bash

安装完成后需要最后一步配置

git config --global user.name "Your Name"

git config --global user.email "email@example.com"

git config --global参数, 表示该机器上所有的git仓库都会使用该配置, 当然也可以对某个仓库指定不同的用户名和email地址

创建版本库, 仓库

版本库, 又称仓库, repository, 可以理解为一个目录, 整个目录里面所有文件都可以被git管理起来, 每个文件的修改, 删除, git都能跟踪, 以便任何时刻都可以追踪历史, 或者在将来某个时刻可以"还原"

mkdir learngit

cd learngit

git init

Initialized empty Git repository in /your/directory/path/learngit/**.git/**

.git目录用于跟踪管理版本库的，没事别修改

在learngti目录下创建文本, 一定要放在learngit目录下(子目录也行), 因为这是一个git仓库, 放到其他地方git再厉害也找不到这个文件

readme.txt: git is a version control system.

git is free software.

第一步, git add 将文件添加到仓库: git add readme.txt; 第二步, git commit 把文件提交到仓库: git commit -m "wrote a readme.txt"

也可以多次add后, 一次commit

git status: 实时查看仓库当前的状态, 可告诉我们文件被修改及待提交状态

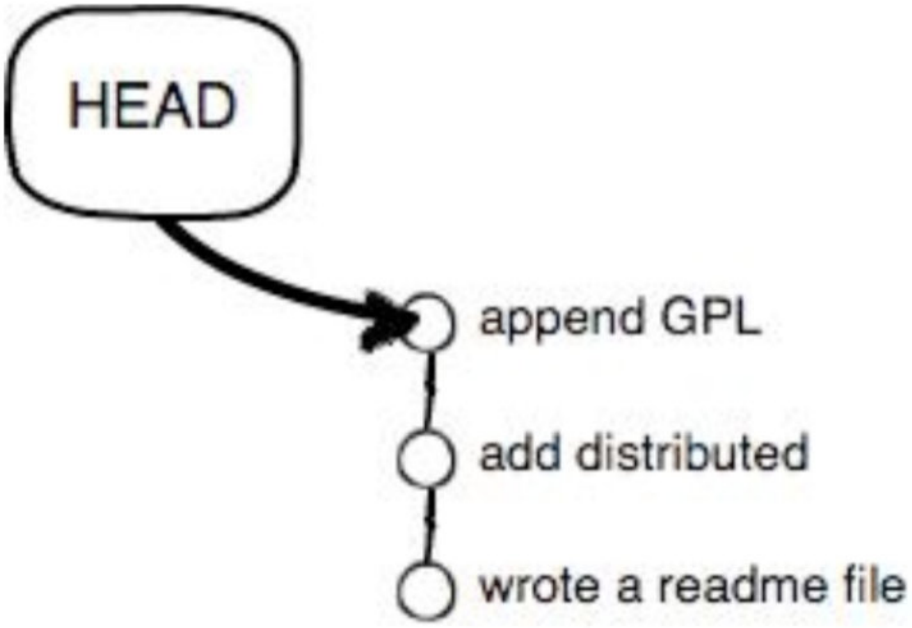
git diff: 查看差异(**没有add前时, add后就不行了**), git diff readme.txt, 查看readme.txt具体被修改内容

git log: 查看历史记录; git log --pretty=oneline

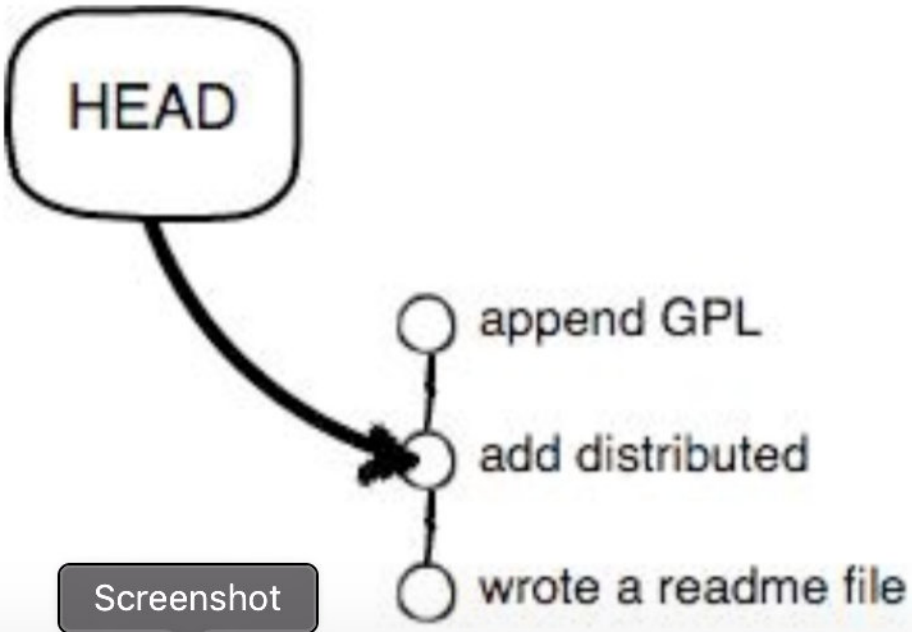
返回上一个版本: git reset --hard HEAD^

返回上上一个版本: git reset --hard HEAD^^

返回任意版本: git reset --hard 3628164 ###3628164位commit id



改为指向“add distributed”：



若退回到某个版本后, 想恢复到新的版本, 找不到新版本的commit id:

git reflog 用来记录每一个命令

HEAD指向的版本就是当前版本, 因此git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit id

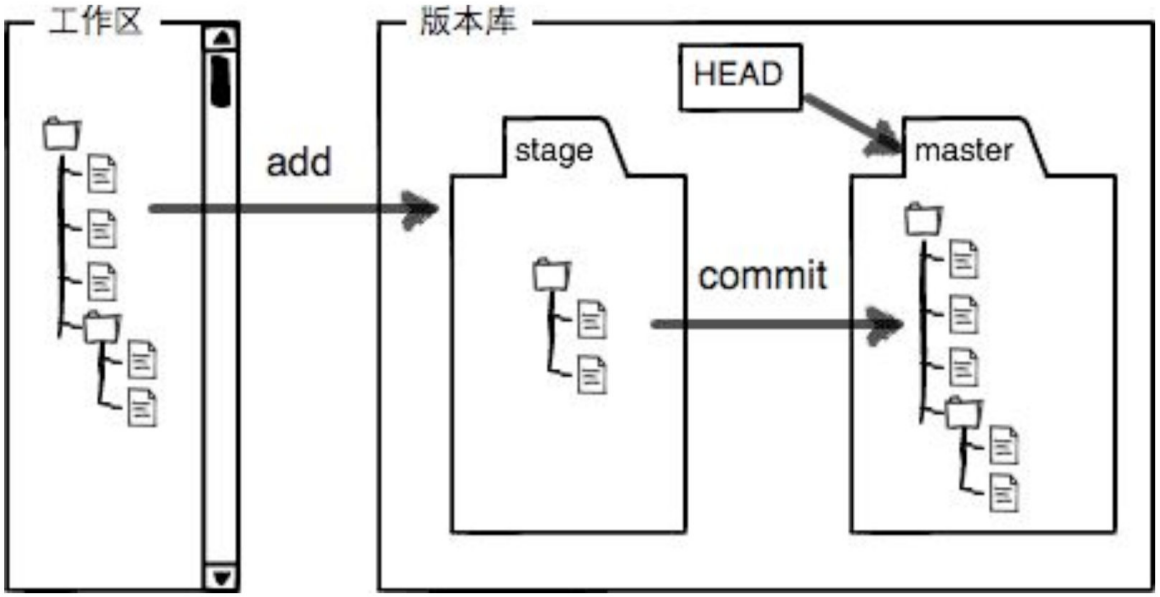
git log查看提交历史

git reflog查看命令历史, 以便确定要回到未来的哪个版本

工作区和暂存区

工作区(working directory): 就是你在电脑能看到的目录, 比如目前的learngit文件夹就是一个工作区

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, 版本库里还有很多东西, 其中最重要的就是stage(或者叫idnex)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



往git版本库添加文件时分两步

1. **第一个步, 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(stage)**
2. **第二步, git commit提交更改, 实际上就是把暂存区(stage)的所有内容提交到当前分支(master)**
3. **简述: 将修改后的文件统统放到暂存区, 然后一次性提交暂存区的修改到分支, 如果没有add到暂存区, 那就不会加入到commit中**

git diff HEAD -- readme.txt 命令可以查看工作区(修改了没有add)和版本库里面最新版本(修改了并且add+commit)的区别

撤销修改: git checkout -- file可以丢弃工作区的修改

两种情况：一种是文件自修改后尚未add到暂存区, 现在撤回就回到和版本库一摸一样的状态; 一种是文件已经添加到暂存区, 又作了修改, 现在撤销修改就回到了添加到暂存区后到状态, 总之, 就是让文件回到最近一次git commit或git add时的状态.

git reset命令既可以退回版本, 也可以把暂存区的修改回退到工作区, HEAD表示最新的版本: git reset HEAD file 可以把暂存区的修改撤掉(unstage), 重新放回到工作区

git checkout -- file: 丢弃工作区的修改

删除文件

1. rm file.txt git checkout -- file.txt
2. git rm file.txt git reset HEAD file.txt
3. git commit -m "remove file.txt" **此时从版本库中彻底删除了, 无法恢复**

恢复删除文件,只能恢复到最近一次提交后修改的内容

远程仓库

1. 创建SSH Key, 在用户目录下, 查看有没有.ssh目录, 如果有, 再看下目录下有没有id_rsa和

id_rsa.pub这两个文件; 如果没有, 创建SSH Key: ssh-keygen -t rsa -C "youremail@example.com"

2. 接下来可在用户主目录里找到.ssh目录, 里面有id_rsa和id_rsa.pub两个文件, 这两个就是SSH Key的秘钥对, id_rsa是私钥, 不能泄漏, id_rsa.pub是公钥, 可以放心告诉他人
3. 登陆github, 在account setting中的ssh key界面add ssh key, 填上title, 在key文本框中粘贴id_rsa.pub文件的内容, 最后add key
4. ssh key的目的就是为了使得github识别出你推送的提交确实是你推送的, 而不是别人冒充的, 而是git支持的ssh协议; 因此, git只要知道了你的公钥, 就可以确认只有你自己才能推送

添加远程仓库

1. 现在已经在本地创建了一个git仓库, 接着在github创建一个git仓库, 并且让这两个仓库进行远程同步, 这样, github上的仓库可以作为备份, 又可以让其他人通过该仓库协作
2. 登陆github, 在右上角" create a new repo", 创建一个新的仓库, repository name填入learngit, 其他保持默认设置, 点击"create repository"成功创建一个新的git仓库
3. **关联本地仓库与远程仓库:git remote add origin git@github.com:huizhen2014/learngit.git**; 添加后,**远程仓库的名字为origin, 为git默认的叫法, 也可以修改为别的,但是origin名字一看就知道是远程库**
4. 将本地库的所有内容推送到远程库: **git push -u origin master; 实际就是将当前分支master推送到远程; -u 参数不但会把本地的master分支内容推送到远程新的master分支, 还把本地的master分支和远程的master分支关联起来, 在以后的推送或着拉取时就可以简化命令**
5. 此后, 只要本地做了提交, 可通过命令推送值github: git push origin master
6.

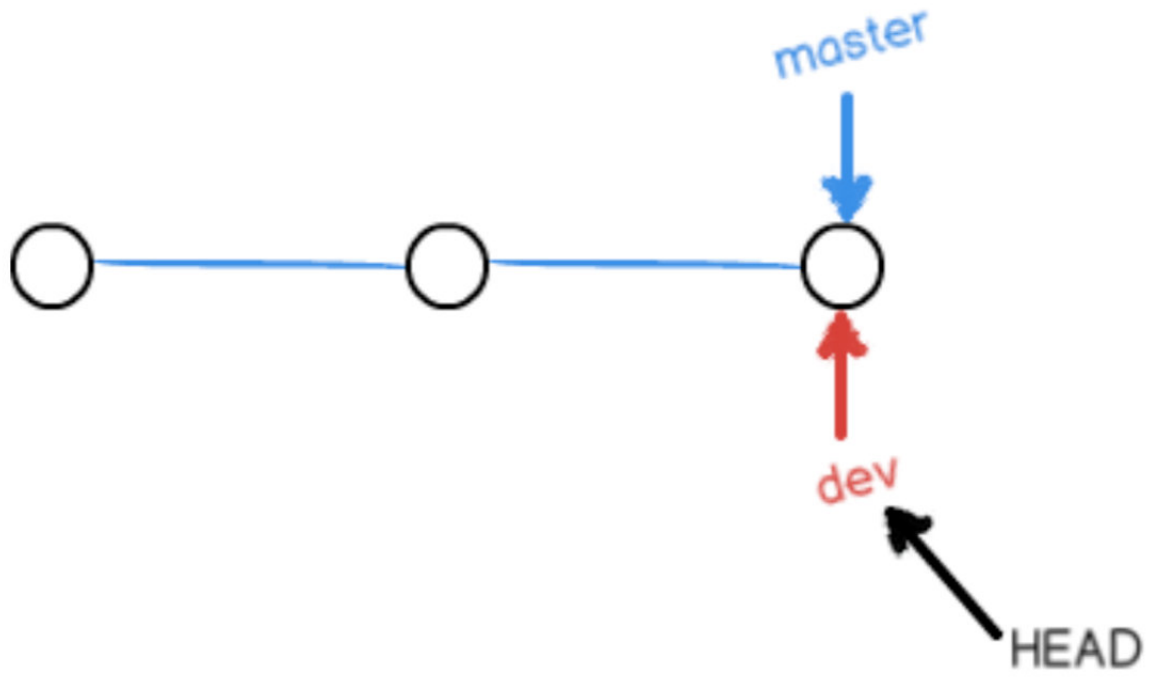
从远程库克隆

1. 针对建立好的远程库, 使用git clone克隆一个本地库: git clone git@github.com:huizhen2014/gitskills.git; 可在任意目录下clone, 克隆后文件夹名字为gitskills

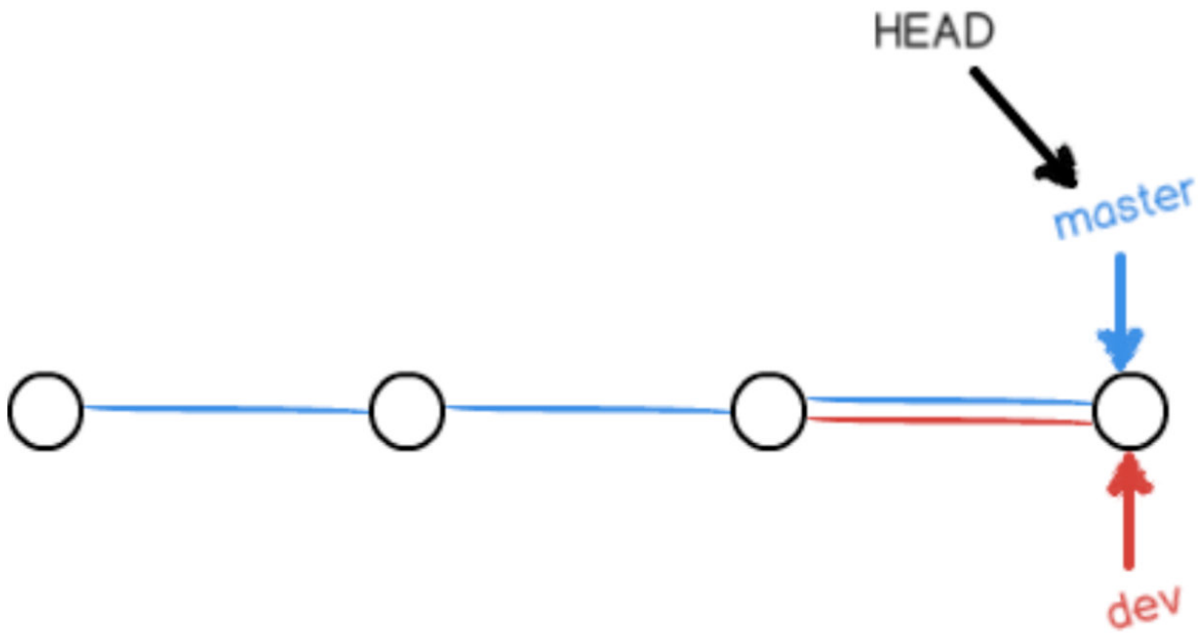
创建与合并分支

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

当我们创建新的分支, 例如dev, git新建了一个指针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

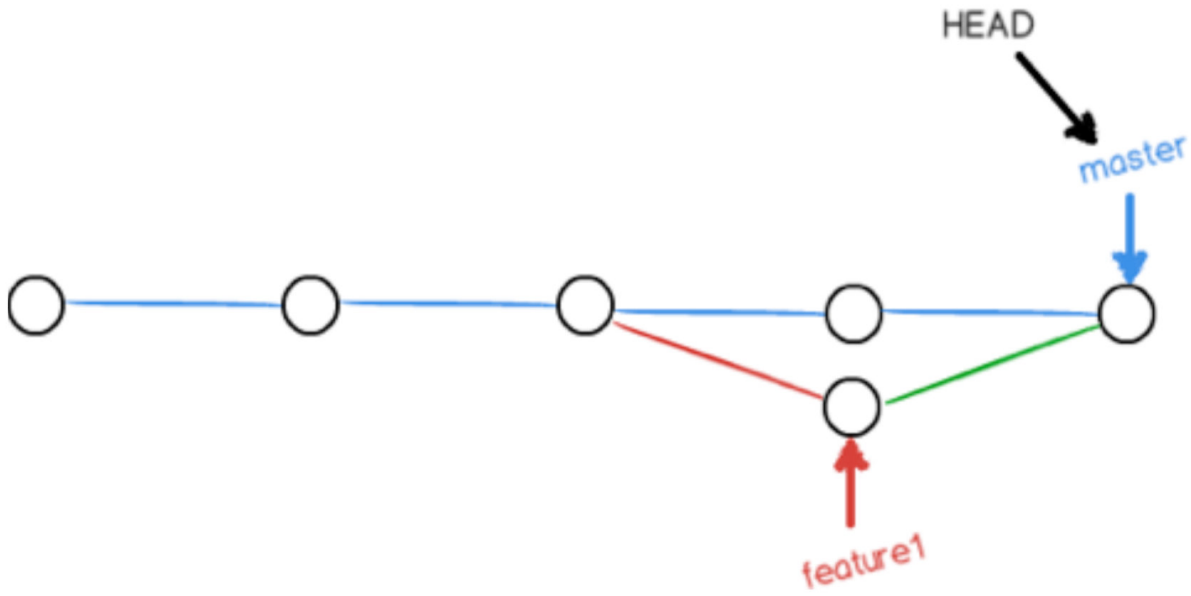


假如新提交后, dev指针往前移动一步, 而master指针不变; 当我们完成dev工作后, 将dev合并到master上



删除分支后dev后, 就只剩一条分支master

1. git checkout -b dev, 创建分支, 同时切换到分支dev; -b相当于: git branch dev; git checkout dev
2. 使用git branch命令查看当前分支: git branch, 当前分支前面会标一个*号
3. 完成dev分支任务提交后,切换回master分支, 在dev分支做的修改不可见, 此时合并dev分支内容到master: git merge dev, 该过程属于Fast-forward模式, 然后可放心删除dev分支: git branch -d dev
4. 当dev分支和master分支均有提交时, 此时的merge无法实现, 会产生冲突, 表现在提交的文件内容上. 此时需要手动修改后: git add file.txt; git commit -m "confilct fixed"



使用带参数的git log查看分支合并情况

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

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

分支管理策略

通常, 合并分支时, git会使用"Fast-Forward"模式,但是该模式, 删除分支后, 会丢掉分支信息; 若强制禁止使用"Fast-Forward"模式, git就会在merge时生成一个新的commit, 这样从分支历史就可以看到分支信息: --no-ff

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

master分支应该是非常稳点的, 也就是仅用来发布新版本, 平时不能在上面干活; 干活都在dev分支上, 再把dev分支合并到master

合并分支时加上--no-ff参数就可以用普通模式合并, 合并后的历史右分支, 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

当工作一半出现急需修复的bug时, 使用stash功能, 把当前工作现场"储藏"起来, 等以后恢复后继续工作

git stash

创建分支完成修复提交后: git add file.txt; git commit -m "fix bug"后, 切换到master分支, 完成合并, 最后删除分支

回到dev分支, 查看隐藏等工作现场

git stash list

恢复工作现场

git stash apply,恢复后可使用git stash drop删除

或着, git stash pop, 恢复到同时把stash内容也删除, 再用git stash list查看, 就看不到任何stash内容了

针对多次stash, 恢复时, 先git stash list查看, 然后恢复指定stash

git stash apply stash@{0}

Feature分支

每次添加新功能, 最好创建一个featrue分支,在上面完成后, 合并, 最后删除该feature分支

针对提交了尚未合并, 需要删除的分支: git branch -D feature-vulcan

多人协作

git remote -v 查看详细信息

推送分支: git push origin master

推送其他分支: git push origin dev

另一台电脑(注意要把ssh key添加到github)或着同一个电脑的另一个目录下克隆

git clone git@github.com:huizhen2014/learngit.git

此时克隆下来之能看到master分支, 若想要在dev分支上开发, 须创建远程origin的dev分支到本地

git checkout -b dev origin/dev

此时可以在dev上继续修改, 然后, 时不时地把dev分支push到远程

git push origin dev

假如两人同时向分支提交内容, 且提交内容有冲突, 推送失败. 此时需要先用git pull把最新的提交orgin/dev抓下来, 然后在本地合并, 解决冲突, 再推送

git pull <remote> <branch>

若提示两个项目不同, 无法合并, git需要参数: --allow-unrelated-histories, 允许不相关历史合并

git pull origin master --allow-unrelated-histories

此时由于没有指定本地dev分支与远程origin/dev分支的链接, 根据提示, 设置dev和origin/dev的链接

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

git pull origin/dev dev

如果git pull提示"no tracking information", 则说明本地分支和远程分支的链接关系没有创建, 用命令git branch --set-upstream-to branch-name origin/branch-name

自定义git

忽略特殊文件

避免git status提示' untracked files...', 可在git工作的根目录下创建特殊的.gitignore文件, 然后把要忽略的文件名填进去, git就会自动忽略这些文件

配置别名

git config --global alias.st status

```
git config --global alias.ci commit

git config --global alias.br branch

git config --global alias.unstage 'reset HEAD'

git config --global alias.last 'log -1'

git config --global alias.lg "log --color --graph --pretty=format: '%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr)%C(bold blue)<%an>%Creset' --abbrev-commit"
```