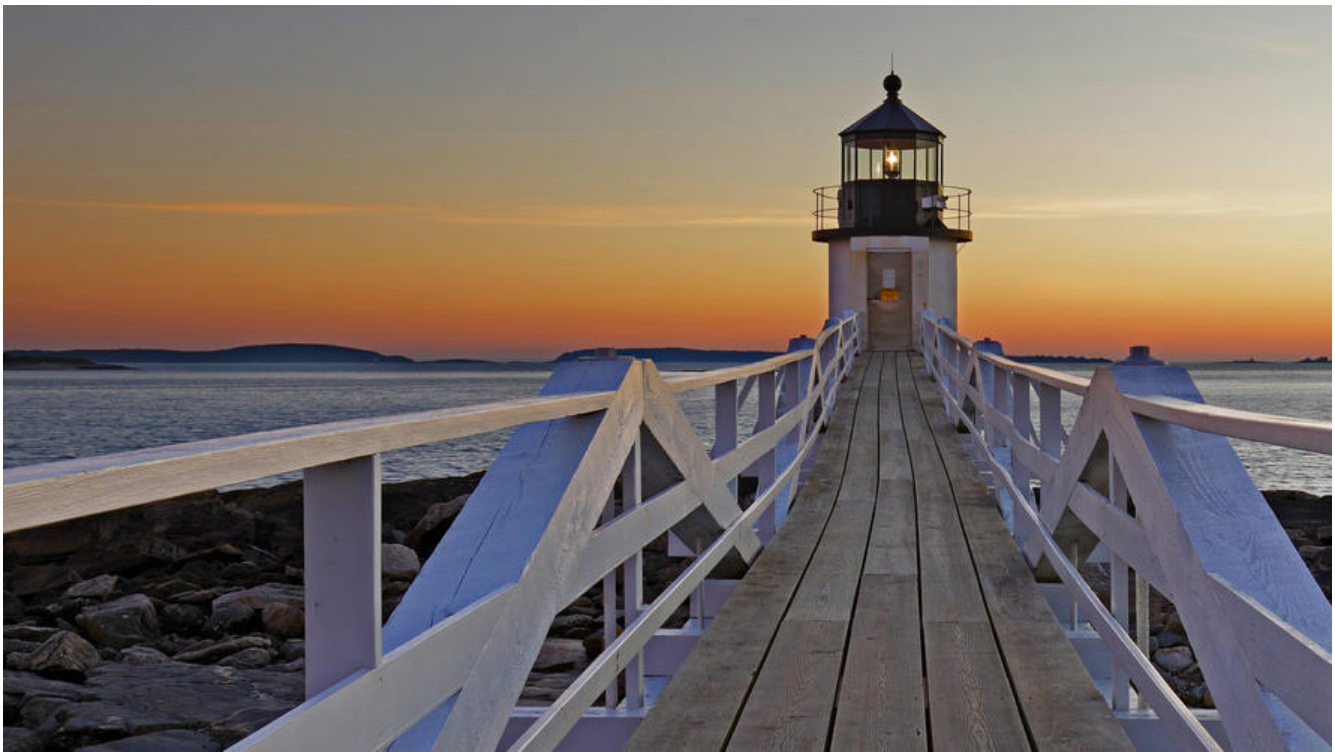
链滴

Git 学习指南

作者：HuixiaZhang
原文链接：https://ld246.com/article/1538814300589
来源网站：链滴



安装Git：

Linux上安装Git：

git　查看系统有没有安装git

sudo apt-get install git　ubuntu安装Git

sudo yum install git　CentOS安装Git

手动编译安装：

下载源码解压

./config

make

sudo make install

git config --global user.name "You Name"　设置用户名

git config --global user.email "You Email"　设置邮箱

--global参数表示这台机器上的所有Git仓库都会设用这个配置

git init　初始化仓库

git add filename　添加文件到仓库

git commit -m "提交说明"　提交文件到仓库

git status　查看仓库状态，跟filename查看指定文件状态，不加查看所有文件状态

git diff filename　查看文件做了那些修改，不指定文件名查看所有

git log　查看提交记录，加--pretty=oneline参数。在一行显示

git reset --hard HEAD^　回退到上个版本，HEAD^^回退到上上个版本，GEAD~100回退上100版本，commit id 回退或者恢复指定提交版本，无需全部id号，确定唯一即可

git reflog　记录每一次操作的命令，回退版本时在恢复，可使用此命令查看 commit id

git diff HEAD -- filename　对比工作区和版本库里面最新版本的区别

修改文件

执行顺序：git add filename

git commit -m ""

git add 提交 工作区文件到暂存区

git commit 提交暂存区文件到当前分支

修改文件

git commit -m ""

由于没有执行git add 把工作区文件提交到暂存区 修改不生效

三种情况下的撤销修改：

1、文件只是修改没有 git add 也就没有 git commit -m ""

git checkout -- filename　注意 "--"必须，否则是切换分支

2、文件修改后 git add 到暂存区 没有 git commit -m "" 提交到分支

git reset HEAD filename　把暂存区的修改撤销掉，重新放回工作区

git checkout -- filename　丢弃工作区的修改

3、文件修改后 git add 到暂存区 并且 git commit -m "" 提交到分支

git reset --hard HEAD^　回退上个版本

删除文件

vi filename　新建一个文件

git add filename　提交文件到暂存区

git commit -m ""　提交文件到版本库

rm filename　从本地删除文件

两种情况
　从版本库删除文件
从版本库恢复文件

1、git rm filename　从版本库删除文件
　git commit -m ""　并且提交到版本库

2、git checkout -- filename　从版本库恢复文件到本地，git rm filename 和 git checkout -- filen me　执行过git rm 后执行git checkout -- filename 会报错，文件已经从版本库删除，无法再从版库恢复

远程仓库：

创建SSH Key:

ssh-keygen -t rsa -C "youremail@example.com"　替换邮箱，一路回车

添加远程仓库：

git remote add origin git@server-name:path/repo-name.git　关联远程仓库

git remote add origin git@github.com:username/repo-name.git

git remote rm origin　删除远程关联仓库

git push -u origin master　第一次推送master分支的所有内容

git push origin master　推送最新修改

克隆远程仓库：

git clone git@github.com:username/repo-name.git　克隆远程仓库到本地

git clone https://github.com/username/repo-name.git　https协议克隆仓库

查看分支：

git branche　查看所有分支，当前分标*

创建分支：

git checkout -b dev　创建分支并切换，加上-b相当于：

git branch dev　创建分支

git checkout dev　切换分支

合并分支：

git merge dev　把当前分支合并到master分支

git branch -d dev　删除分支

合并分支前先切换到master分支：

git checkout master　切换分支

git merge dev　合并dev分支到master分支

git branch -d dev　删除dev分支

解决冲突：

git checkout -b feature　创建一个新的分支

修改readme.txt 添加一行 Creating a new branch is quick AND simple

在feature 分支提交

git add readme.txt

git commit -m "AND simple"

切换到master分支
git checkout master
修改readme.txt 添加一行 Crateing a new branch is quick & simple
在master分支提交

git add readme.txt
git commit -m "& simple"

在master分支合并feature分支
git merge feature　提示冲突，必须手动解决

使用git status查看冲突文件

查看readme.txt 内容
git用<<<<<<<,=======,>>>>>>>标记不同分支的内容

修改readme.txt内容 Creating a new branch is quick and simle
在提交
git add readme.txt
git commit -m "conflict fixed"

git log --graph --pretty=oneline --abbrev-commit　查看分支合并情况

git log --graph　查看分支合并图

git branch -d feature　删除分支

分支管理策略：

合并分支时，如果可能Git会用Fast forward模式，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，这样从分支历史上可看出分支信息

使用--no-ff的方式merge：

git merge --no-ff -m "merge with no-ff" dev 以禁用fast forward方式提交，因为本次合并需要创一个新的commit，所以加上-m参数，把commit描述写进去

合并后使用git log查看分支历史：

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

分支管理原则：

master分支应该是稳定的，仅用来发布新版本，平时不再上面干活

dev分支，是不稳定的用来干活，到版本发布时，把dev分支合并到master分支上

每个人都有一个自己分支，时不时往dev分支上合并

Bug分支

场景：当你接到一个代号101的bug时，很自然去创建一个分支issue-101来修复，但是在dev分支上工作还没进行提交，因为工作还没有完成，bug又要尽快修复，用到git提供的stash功能，把当前工现场“储藏”起来，等恢复现场后继续工作。

git stash　"储藏"当前工作现场

确定在那个分支上修改bug

git checkout master　切换到待修复bug分支

git checkout -b issue-101　创建并切换到bug分支

修复后

git add filename　提交工作区文件到暂存区

git commit -m ""　提交暂存区文件到版本库

git branch master　切换到master分支

git merge --no-ff -m "" issue-101(bug分支名)

git checkout dev　切换到dev分支

git stash list　查看“储藏”工作现场

恢复工作现场:

git stash apply　恢复后stash内容并不删除

git stash drop　删除stash

git stash pop　恢复同时删除stash内容

多次stash恢复:

git stash list　查看有哪些stash

git stash apply　恢复指定satsh

feature分支:

场景：开发新功能，先创建一个feature分支，开发好合并到master分支，最后删除feature分支

git checkout -b feature-vulcan　创建feature分支

git add vulcan.java　(开发完成)新创建的文件添加到暂存区

git checkout dev　切换到到dev分支，准备合并

但是现在因为某些原因，新功能取消，虽然白干了，但是机密资料的分支还是要销毁

git branch feature-vulcan　删除分支，Git提示分支没有合并不能删除，修改将丢失，如果要强行除，需要使用大写-D参数

git branch -D feature-vulcan　强制删除未合并过的分支

多人协作:

git remote　查看远程仓库信息

git remote -v　显示仓库详细信息，若是没有推送权限，看不到push地址

git push origin master　推送分支

git push origin branch-name　推送指定分支

抓取分支:

多人协作时，大家都会往master和dev分支上推送各自修改

git clone git@github.com:username/repo-name.git　默认抓取master分支

git checkout -b dev origin/dev　创建远程origin的dev分支到本地

git add filename　添加文件到暂存区

git commit -m ""　提交文件到版本库

git push origin dev　推送到指定分支，要是有最新提交和当前推送的提交有冲突，则无法推送。
git pull　先抓取远程分支修改，在本地合并，解决冲突，在推送

git branch --set-upstream-to=origin/dev dev　指定本地分支与远程分支origin/dev的分支链接

添加远程仓库并切换分支：
先在git添加SSH Key
git clone git@github.com:username/repo-name.git　克隆远程仓库
git checkout -b branch-name　本地创建与远程仓库分支一样的分支
git branch --set-upstream-to=origin/远程分支名 本地分知名

Rebase:
git rebase　把分叉的提交历史“整理”成一条直线

创建标签：
git branch　查看当前所属分支
git checkout branch-name　切换到需要打标签的分支上
git tag tag-name　给当前分支打标签
git tag　查看所有标签
git tag tag-name commit-id　给已经提交的commit打标签
git show tag-name　查看标签信息
git tag -a tag-name -m "" commit-id　创建带有说明的标签，-a指定标签名，-m指定说明文字

操作标签：
git tag -d tag-name　按照标签名删除标签
git push origin tag-name　推送指定标签到远程仓库
git push origin --tags　推送所有标签到远程仓库

从远程仓库删除标签：
git tag -d tag-name　删除本地标签
git push origin :refs/tags/tag-name　删除远程仓库tag

使用GitHub:
参与一个开源项目，找到想参与的开源项目，fork一下，在从自己的仓库clone到本地，修改后推送远程仓库，如果希望开源项目接受自己的修改，就在GitHub上发起一个 pull request给作者。
1、在GitHub上Fork开源项目
2、从自己的仓库clone Fork后的项目
3、在GitHub上发起pull request，贡献代码到官方仓库

使用码云：
添加ssh key
git remote add origin git@gitee.com:username/repo-name.git　关联本地仓库和远程仓库
git push

git pull

若是报错：fatal: remote origin already exists.说明本地仓库已经关联一个叫origin的仓库了

git remote -v　查看远程仓库信息

git remote rm origin　删除已关联的远程仓库

git remote add git@gitee.com:username/repo-name.git

git remote -v　已经正确关联可以愉快的push了

同步本地代码到两个远程仓库：

git remote rm origin　删除已关联的origin仓库

git remote add github git@github.com:username/repo-name.git　先关联GitHub，远程库名叫g
thub

git remote add gitee git@gitee.com:username/repo-name.git　在关联Gitee，远程库名叫gitee

git mote -v　查看远程仓库信息，可以看到两个仓库

同时和多个远程仓库同步：

git push github master　推送到GitHub

git push gitee master　推送到Gitee

同时码云也可以Pull Request

自定义Git：

git config --global color.ui true　让Git显示颜色

忽略特殊文件：

针对需要又不能提交的文件，如配置文件

在Git工作区根目录创建一个 .gitignore文件，把需要忽略的文件名填进去，Git会自动忽略这些文件

或者直接使用GitHub为我们准备的各种配置文件：

https://github.com/github/gitignore

添加好之后把.gitignore 文件提交到Git

在使用git status检验，是否输出：working directory clean

/xxx/xxx/*　忽略某个路径下的所有文件

git add -f filename　强制添加文件到Git，文件被添加到.gitignore忽略了，则需要使用此命令，强
添加

配置别名：

git config --global alias.st status　直接用git st就等于git status

git config --global alias.co checkout

git config --global alias.ci commit

git config --global alias.br branch

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

git config --global alias.last 'log -1'　显示最后一次提交内容

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"

配置git的时候加上--global时针对当前用户起作用，不加针对当前仓库起作用

配置文件在：.git/config

别名在[alias] 后面，要删除别名，删除对应行就行了

当前用户配置文件放在用户主目录下的一个隐藏文件.gitconfig中

git鼓励别名

搭建git服务器：

Ubuntu下搭建git服务器：

1、安装Git：

sudo apt-get install git

2、创建git用户

sudo adduser git

3、创建证书登录：

收集所有需要登录的用户公钥，就是id_rsa.pub文件，导入到/home/git/.ssh/authorzed_keys文件，一行一个

4、初始化仓库

先选择一个目录作为git仓库，假定是/srv/sample.git

cd /srv

sudo git init --bare sample.git　创建一个裸仓库

因为git仓库纯粹是为了共享，所以不让用户直接登录到服务器上更改工作区，并且服务器上的Git仓通常都是以.git结尾。然后把owner改为git：

sudo chown -R git:git sample.git

5、禁用shell登录

编辑/etc/passwd 找到类似：

git: x:1001:1001:,,,:/home/git:/bin/bash

改为：

git: x:1001:1001:,,,:/home/git:/usr/bin/git-shell

这样git用户可以通过ssh使用git但是无法登录shell,因为我们为git用户指定的git-shell每次一登陆就动退出

6、克隆远程仓库

git clone git@server:/srv/sample.git

管理公钥，把每个人的公钥收集起来放到：

/home/git/.ssh/authorized_keys

公钥管理使用gitosis

权限管理使用gitolite