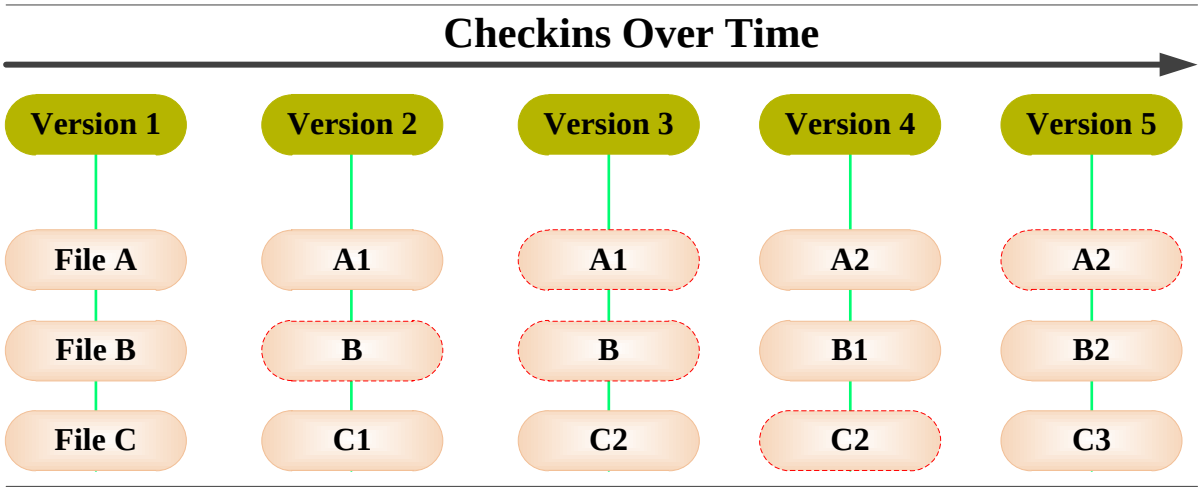
Git 学习

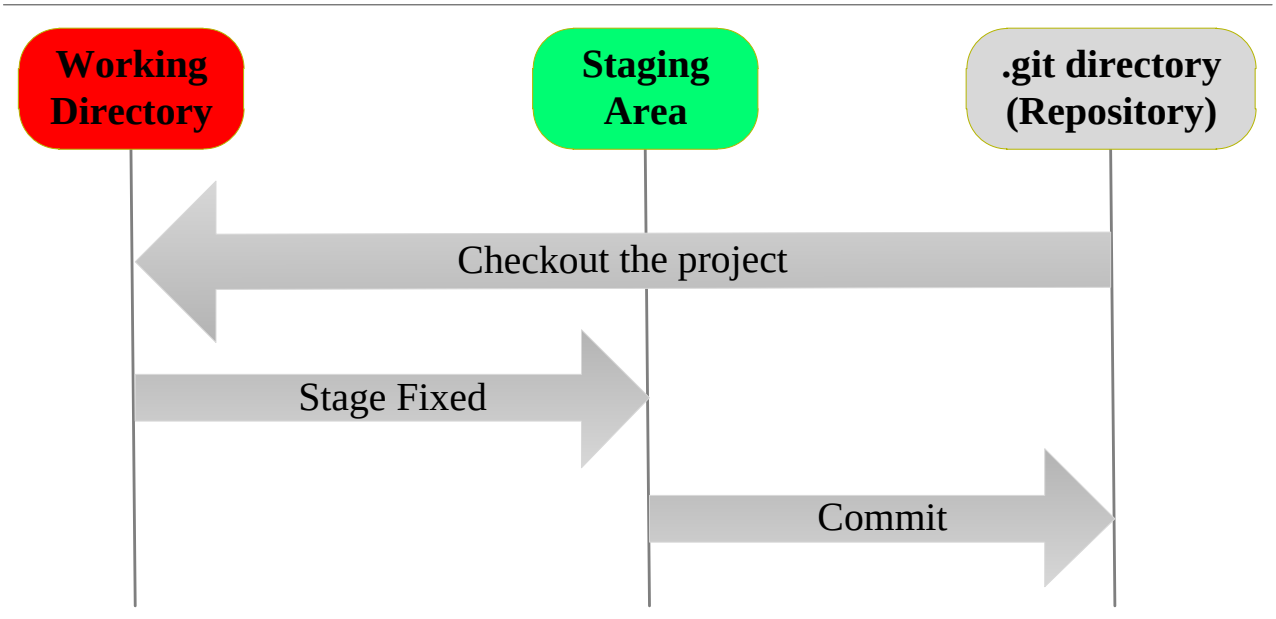
Git 文档官网：https://git-scm.com/book/zh/v2

一、Git 简介：

Git 是把数据看作是对小型文件系统的一组快照。 每次提交更新，或在 Git 中保存项目状态时，它主要对当时的全部文件制作一个快照并保存这个快照的索引。为了高效，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。 Git 对待数据更像是一个快照流。



Git 的几乎所有操作均来自本地，而不需要连接网络。Git 数据库中保存的信息都是以文件内容的哈希值来索引，而不是文件名。



从其它计算机克隆仓库时，拷贝的是图中 Git 仓库目录；工作目录是对项目的某个版本独立

提取出来的内容；暂存区域是一个文件，保存了下次将提交的文件列表信息，一般在 Git 仓库目录中，有时候也被称作“索引”，因此由上可得基本的 Git 工作流程如下：

a、在工作目录中修改文件。

b、暂存文件，将文件的快照放入暂存区域。

c、提交更新，找到暂存区域的文件，将快照永久性存储到 Git 仓库目录。

二、Git 的安装使用

1、Git 的安装

(1)Linux 上的安装(以 CentOS 为例)

1)yum 安装 Git

```
[root@vs1 ~]#yum install git -y
```

Git 官网有在各种 Unix 的系统上安装步骤：http://git-scm.com/download/linux

2)源码安装 Git

```
#安装 git 依赖的安装包
[root@vs1 ~]#yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel -y
#为了能够添加更多格式的文档（如 doc，html，info），需要安装以下的依赖包
[root@vs1 ~]#yum install asciidoc xmlto docbook2x -y
[root@vs1 ~]#wget https://github.com/git/git/archive/v2.11.0.tar.gz
[root@vs1 ~]#tar zxvf v2.11.0.tar.gz
[root@vs1 ~]#cd git-2.11.0
[root@vs1 ~]#make configure
[root@vs1 ~]#./configure --prefix=/usr/
[root@vs1 ~]#make all doc info
[root@vs1 ~]#make install install-doc install-html install-info
```

3)升级 Git：

```
[root@vs1 ~]#git clone git://git.kernel.org/pub/scm/git/git.git
```

(2)Windows 上的安装

在 Windows 上安装 Git 也有几种安装方法。官方版本可以在 Git 官方网站下载。打开 http://git-scm.com/download/win，下载会自动开始。要注意这是一个名为 Git for Windows 的项目（也叫做 msysGit），和 Git 是分别独立的项目；更多信息访问 http://msysgit.github.io/。

2、Git 的配置

Git 自带一个 git config 的工具来帮助设置控制 Git 外观和行为的配置变量。这些变量存储在三个不同的位置：

(1) /etc/gitconfig 文件: 包含系统上每一个用户及他们仓库的通用配置。 如果使用带有 --system 选项的 git config 时，它会从此文件读写配置变量。

(2) ~/.gitconfig 或 ~/.config/git/config 文件：只针对当前用户。 可以传递 --global 选项让 Git 读写此文件。

(3) 当前使用仓库的 Git 目录中的 config 文件（就是 .git/config）：针对该仓库。

每一个级别覆盖上一级别的配置，所以 .git/config 的配置变量会覆盖 /etc/gitconfig 中的配置变量。

(4)设置用户信息（用户名称和邮件地址）————配置文件 config

配置 Git 的时候，加上--global 是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用，每个仓库的 Git 配置文件都放在`.git/config`文件中，而当前用户的 Git 配置文件放在用户主目录下的一个隐藏文件`.gitconfig`中：

```
[root@vs1 ~]#git config --global user.name "maohua"
[root@vs1 ~]#git config --global user.email root@localhost
[root@vs1 ~]#git config --global color.ui true                #设置不同字体的颜色
[root@vs1 ~]#git config --list                                #查看设置的 git 配置信息
[root@vs1 ~]#git config user.name                             #查看设置的某个具体的 git 配置信息
#配置 git 别名有两种方式，其一是配置文件设置：
[root@vs1 ~]#cat .gitconfig                                   #在用户家目录之下配置
[alias]
    co = checkout
    ci = commit
    br = branch
    st = status
##其二是命令行配置：
##       Usage: git config --global alias.别名 '被替代的名称'
##  例：git config --global alias.last 'log -1'  #将'log -1'替换为 last，查看最后一次的 Git 日志信息
[root@vs1 ~]#git config --global alias.last 'log -1'
[root@vs1 ~]#git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Cre
set -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
```

(5)Git 获取帮助的方式：

```
$ git   help   <verb>
$ git   <verb>   --help
$ man   git-<verb>
```

```
#例如，若获取 config 的帮助信息
[root@vs1 ~]#git help config
```

(6)Git 的命令自动补全设置

在用户的家目录的.bashrc 文件中添加一下内容：

```
 [root@vs1 ~]#ls /root/git/contrib/completion/git-completion.bash
/root/git/contrib/completion/git-completion.bash
[root@vs1 ~]#cat ~/.bashrc
sh /root/git/contrib/completion/git-completion.bash
```

(7)Git 的 shell 提示符设置

在用户的家目录的.bashrc 文件中添加一下内容：

```
[root@vs1 ~]#ls /root/git/contrib/completion/git-prompt.sh
export GIT_PS1_SHOWDIRTYSTATE=1
export PS1='\w$(__git_ps1 " (%s)")\$ '              #设置 git 的 shell 提示符所在仓库和分支名称
[root@vs1 ~]#cat ~/.bashrc
sh /root/git/contrib/completion/git-prompt.sh
```

3、Git 的使用

(1)Git 的初始化

```
[root@vs1 ~]#git init                                #在现有目录下初始化 Git 仓库，也可新建目录
[root@vs1 ~]ls .git/
branches  config  description  HEAD  hooks  info  objects  refs
```

Git 初始化的生成的文件介绍：

objects 目录存储所有数据内容

refs 目录存储指向数据（分支）的提交对象的指针

HEAD 文件指示目前被检出的分支

index 文件保存暂存区信息

description 文件仅供 GitWeb 程序使用；

config 文件包含项目特有的配置选项；

info 目录包含一个全局性排除（global exclude）文件，用以放置那些不希望被记录在.gitignore 文件中的忽略模式（ignored patterns）；

hooks 目录包含客户端或服务端的钩子脚本（hook scripts）

(1)Git 的 GUI 客户端

Git 的 GUI 界面：在安装 git 的同时，也安装了可视化工具，gitk 和 git-gui。Git 的其他 GUI 客户端详见：https://git-scm.com/downloads/guis。

(2)Git 的克隆：

克隆仓库的命令格式：git clone [URL] [REPO_NAME]

```
[root@vs1 ~]#git clone https://github.com/libgit2/libgit2          #克隆 Git 的可链接库 libgit2
[root@vs1 ~]#git clone https://github.com/libgit2/libgit2  mylibgit  #克隆远程仓库自定义名称
```

(3)查看状态以及跟踪新文件、提交删除、重命名：

```
[root@vs1 ~]#git status              #检查当前文件状态
[root@vs1 ~]#git status -s           #状态简单浏览
[root@vs1 ~]#git add FILE_NAME       #添加跟踪新文件
#在 git 根目录下添加.gitignore 文件，添加忽略的跟踪文件名称，支持正则表达式，.gitignore 文件本身要放到版本库里，并且可以对.gitignore 做版本管理！
[root@vs1 ~]#cat .gitignore
*.[oa]                               #忽略所有以 o 或 a 结尾的文件
##查看详细的.gitignore 文件：https://github.com/github/gitignore
[root@vs1 ~]#git diff                #查看尚未暂存的文件更新了哪些部分[--cached|staged]
[root@vs1 ~]#git commit -m "添加提交信息"     #提交更新
[root@vs1 ~]#git commit -a -m "添加提交信息" #跳过使用暂存区域，提交之前不用 git add
#删除文件：
[root@vs1 ~]#rm FILE_NAME            #删除磁盘上的文件
[root@vs1 ~]#git rm FILE_NAME        #删除暂存区域区域内的文件
[root@vs1 ~]#git rm --cached FILE_NAME       #删除缓冲区域内的文件
#移动[重命名]文件
[root@vs1 ~]#git mv OLD_NAME_FILE NEW_NAME_FILE
```

对于在本地删除错了的文件，可用版本库中的文件替换本地文件：`git checkout -- test.txt`

(4)查看 git 提交历史：

git log 的常用选项

-p	按补丁格式显示每个更新之间的差异。
--stat	显示每次更新的文件修改统计信息。
--shortstat	只显示 --stat 中最后的行数修改添加移除统计。
--name-only	仅在提交信息后显示已修改的文件清单。
--name-status	显示新增、修改、删除的文件清单。
--abbrev-commit	仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。
--relative-date	使用较短的相对时间显示（比如，“2 weeks ago”）。
--graph	显示 ASCII 图形表示的分支合并历史。
--pretty	使用其他格式显示历史提交信息，可用的选项包括： oneline，short，full，fuller 和 format（指定格式）

```
[root@vs1 ~]#git log -p -2           #[-p]用来显示每次提交的内容差异,[-2] 来仅显示最近两次提交
[root@vs1 ~]#git log --stats         #查看每次提交的简短的统计信息
[root@vs1 ~]#git log --pretty=oneline        #将每次提交信息放在一行输出
##基于--pretty 的子选项还有：--pretty=[oneline|short|full|fuller|format]
```

```
[root@vs1 ~]#git log --pretty=format:"%h - %an, %ar : %s"     #按照一定的格式输出
##format 常用的占位符及其代表含有可参考：https://git-scm.com/book/zh/v2/ch00/pretty_format
[root@vs1 ~]#git log --pretty=format:"%h - %an, %ar : %s" --graph      #形象展示合并和分支
```

git log --pretty=format:"*****" 常用的选项

%H	提交对象（commit）的完整哈希字串	%ae	作者的电子邮件地址
%h	提交对象的简短哈希字串	%ad	作者修订日期（可用--date= 定制格式）
%T	树对象（tree）的完整哈希字串	%ar	作者修订日期，按多久以前的方式显示
%t	树对象的简短哈希字串	%cn	提交者(committer)的名字
%P	父对象（parent）的完整哈希字串	%ce	提交者的电子邮件地址
%p	父对象的简短哈希字串	%cd	提交日期
%an	作者（author）的名字	%cr	提交日期，按多久以前的方式显示

```
#git log 限制输出长度
[root@vs1 ~]#git log --pretty="%h - %s" --author=gitster --since="2008-10-01" --before="200
8-11-01" --no-merges -- t/
##在输出的历史信息当中指定作者、提交的起止日期、不是合并的信息以及在哪一个目录之下的，类似于正则表达式的筛选
```

git log 输出的选项

-(n)	仅显示最近的 n 条提交
--since|after	仅显示指定时间之后的提交。
--until|before	仅显示指定时间之前的提交。
--author	仅显示指定作者相关的提交。
--committer	仅显示指定提交者相关的提交。
--grep	仅显示含指定关键字的提交
-S	仅显示添加或移除了某个关键字的提交

(5)git 的撤销操作：

```
[root@vs1 ~]#git commit -m "initial commit"
[root@vs1 ~]#git add forgotten_file
[root@vs1 ~]#git commit -amend              #追加未提交的文件
[root@vs1 ~]#git checkout -- FILE_NAME      #撤销对文件的额修改，FILE_NAME 是已修改的追踪文件
[root@vs1 ~]#
```

(6)Git 的版本回退：

在 Git 中，用 HEAD 表示当前版本，上一个版本就是 HEAD^，上上一个版本就是 HEAD^^，当然往上 100 个版本写 100 个^比较容易数不过来，所以写成 HEAD~100。现在，我们要把当前版本“append GPL”回退到上一个版本“add distributed”，就可以使用 git reset 命令：

```
[root@vs1 ~]#git reset --hard HEAD^
[root@vs1 ~]#git reset --hard COMMIT_ID     #使用 git reflog 查看需要回到哪一个 COMMIT_ID
```

每次修改，如果不 `add` 到暂存区，那就不会加入到 `commit` 中

A、对于只是在工作区的文件的撤销可如下进行：`git checkout -- readme.txt` 意思就是，把

`readme.txt`文件在工作区的修改全部撤销，命令中的`--`很重要，没有`--`，就变成了“切换到另一个分支”的命令。

B、对于已经提交到暂存区的文件的撤销可如下两步进行：

1)先撤销暂存区中文件的修改：`git reset HEAD readme.txt`

2)在撤销工作区中文件的修改：`git checkout -- readme.txt`

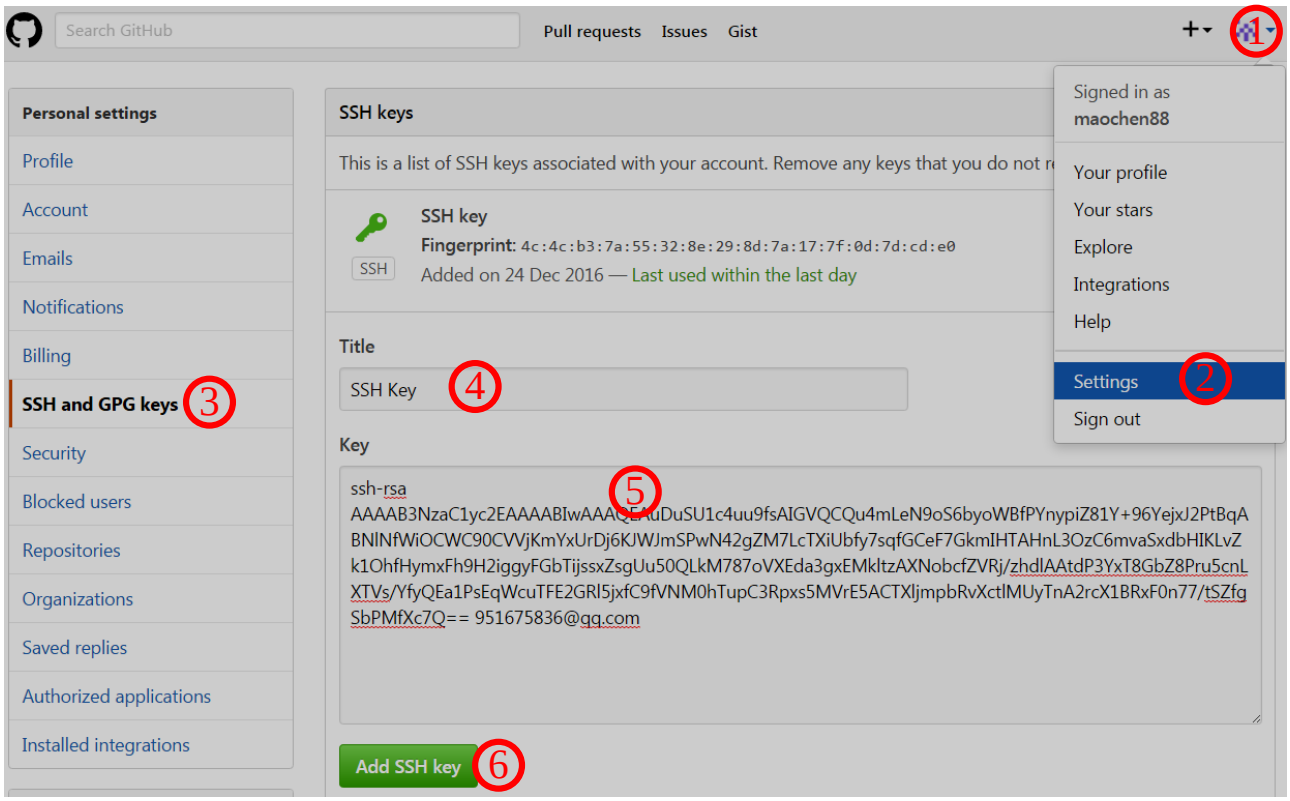
C、对于已经提交到版本库的文件撤销采用版本回退的机制：

`git reset --hard HEAD^` #回退到上一个版本

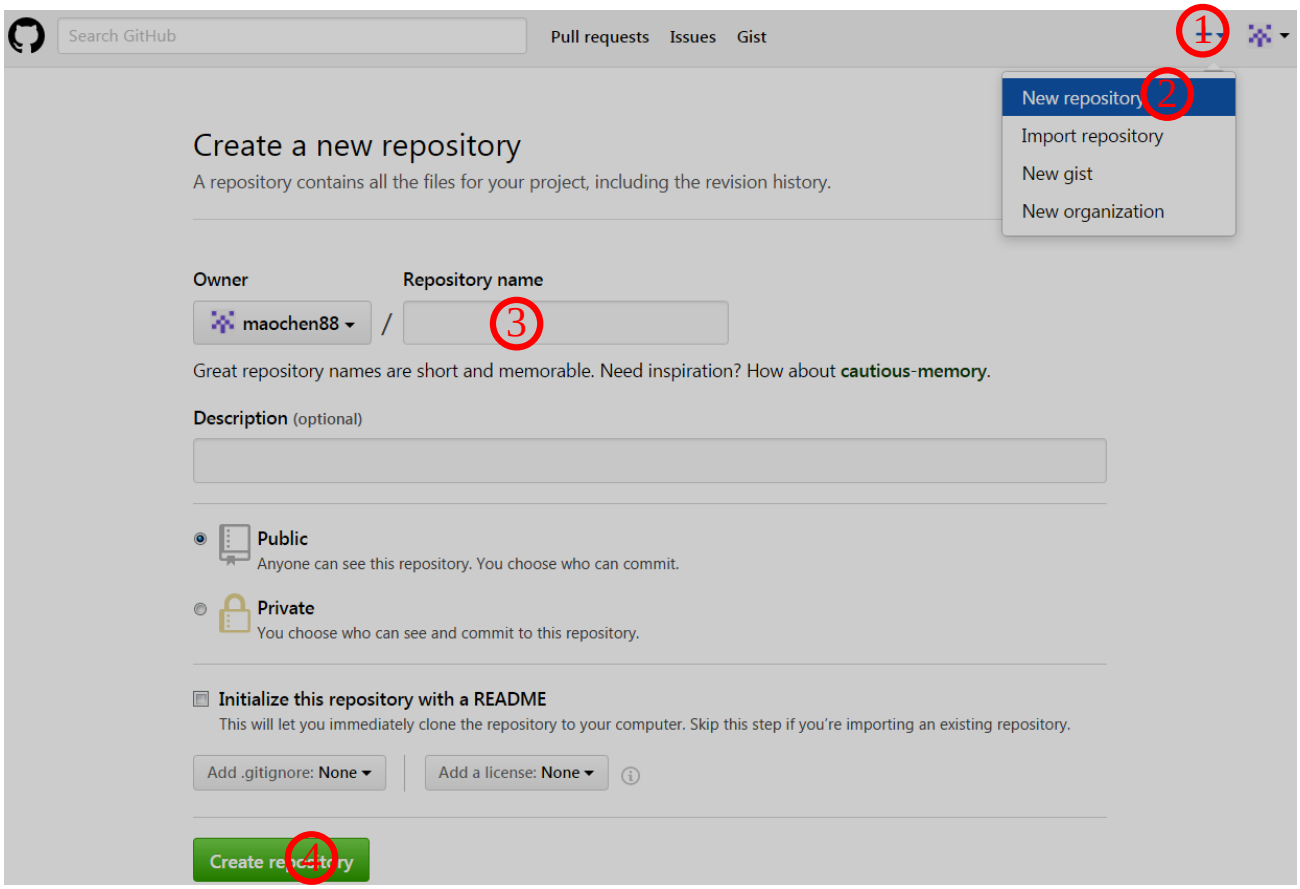
`git reset --hard COMMIT_ID` #回退到指定的版本，使用`git reflog`查看对应的`COMMIT_ID`

(7)添加远程库(www.github.com)

首先需要在 github 官网申请账户，由于你的本地 Git 仓库和 GitHub 仓库之间的传输是通过 SSH 加密的，所以，需要一点设置，创建 SSH Key(`ssh-keygen -t rsa -C "youremail@example.com"`)并将 id_rsa.pub 文件内容复制到 github 账户目录之下：



在 github 上新建一个远成仓库，并在本地关联起来，首先在远程 github 上新建一个 learngit 仓库：



关联一个远程库，使用命令 `git remote add origin git@github.com:maochen88/learngit.git`；

关联后，使用命令 `git push -u origin master` 第一次推送 master 分支的所有内容；

每次本地提交后，就可以使用命令 `git push origin master` 推送最新修改至远程仓库。

从远程 github 克隆仓库到本地：`git clone git@github.com:maochen88/gitskills.git`

(8)与分支有关的操作

查看分支：`git branch`

创建分支：`git branch <name>`

切换分支：`git checkout <name>`

可使用一条命令：`git checkout -b <branch_name>`

创建+切换分支：`git checkout -b < branch_name>`

合并某分支到当前分支：`git merge < branch_name>` git 默认的合并模式是 `Fast forward`，此种模式下，删除分之后会丢掉分支信息，可强制禁用该模式使用 `--no-ff` 选项

删除分支：`git branch -d < branch_name>`

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。解决冲突的方法是：在本地修改冲突的文件内容为自己需要的，然后正常添加提交 `add、commit` 即可

查看分支合并图：`git log --graph --pretty=oneline --abbrev-commit`

(9)创建修复 bug 的分支()

保存当前分支(假设当前在 dev 分支)中工作区的内容：`git stash`

回到需要修复 bug 的分支：`git checkout master`

在 bug 分支上创建分支已修复 bug：`git checkout -b bug-100`

在创建的 bug 分支上修复 bug：`.....`

修复好 bug 回到 bug 分支：`git checkout master`

合并刚才修复的 bug：`git merge --no-ff -m “merge bug fix 100” bug-100`

删除新建的 bug 分支：`git branch -d bug-100`

回到项目最初的分支：`git checkout dev`

回到修复 bug 之前时的工作区中有两种方式：一是用 `git stash apply`，恢复另一种方式是用 `git stash pop`，恢复的同时把 stash 内容也删了。第一种的方式较繁琐如下：

查看保存的工作区：`git stash list`

回到指定的工作区：`git stash apply stash@{N}`

删除指定的工作区：`git stash drop`

对于没有合并的分支，如删除需要强行删除使用`-D`选项：`git branch -D <name>`

(10)推送分支到远程仓库

查看远程仓库的名称：`git remote -v`。

将本地的分支推送到远程分支上：`git push <remote_branch_name> <local_branch_name>`

(11)多人协作的工作模式：

首先，推送自己的修改`git push <remote_branch_name> <local_branch_name>`；

如果推送失败，则因为远程分支比你的本地更新，需要先用`git pull`从远程抓取分支；

如果合并有冲突，则解决冲突，并在本地提交；

没有冲突或者解决掉冲突后，再用`git push origin branch-name`推送就能成功！

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。

建立本地分支和远程分支的关联，命令`git branch --set-upstream-to=<upstream>`

(12)创建标签 tag

用于新建一个标签，默认为当前位置，可指定 commit id：`git tag <tag_name> [COMMIT_ID]`

可以指定标签信息：`git tag -a <tag_name> -m "blablabla..." [COMMIT_ID]`

删除标签：`git tag -d <tag_name>`

推送某个标签到远程，使用命令：`git push origin <tag_name>`

一次性推送全部尚未推送到远程的本地标签：`git push origin --tags`

如果标签已经推送到远程，要删除需要两步，先从本地删除，再从远程删除：

1)本地删除：`git tag -d <tag_name>`

2)远程删除：`git push origin :refs/tags/<tag_name>`

三、搭建 Git 服务器

在已安装 Git 的 Linux 主机上执行以下操作：

1、创建一个 git 用户，运行 Git 服务：

```
[root@vs1 ~]#useradd git -s `which git-shell`        #添加 git 用户，并设置 shell，不能登录系统
[root@vs1 ~]#echo "123" | passwd --stdin git         #设置密码
[root@vs1 ~]#su - git
[git@vs1 ~]$ssh-keygen -t rsa -f ~/.ssh/id_rsa -P ""#生成秘钥文件
```

2、初始化一个 Git 的仓库：

```
[git@vs1 ~]$mkdir /tmp/git_repo
[git@vs1 ~]$git init --bare /tmp/git_repo            #初始化一个 git 仓库
[root@vs1 ~]#chown -R git:git /tmp/git_repo          #设置仓库的属主属组为 git
```

3、从远程主机克隆 git_server：

```
#将公钥文件上传 git_server 上的 git 用户家目录下：
[root@vs2 ~]#ssh-copy-id -i ~/.ssh/id_rsa.pub git@git_server
[root@vs2 ~]#git clone git@git_server:/tmp/git_repo          #clone 远程 git_server 上的仓库
```