2.Git基础

`取得项目` `更新` `查看历史` `撤销` `标签`

> 大总结：
> `git init`
> 本地目录创建Git项目
>
> `git add [file-name]`
> 添加文件或文件夹的版本控制
>
> `git clone [url]`
> 尽量使用https的url来克隆，这样可以打开ssh通道传输
>
> `git status`
> 检查项目文件状态
>
> `git cat .gitignore`
> `*.[oa]`
> `*~`
> 忽略某些文件提示版本控制问题
>
> `git diff`
> `git --staged`
> 对比未暂存和已暂存的变化

对比已暂存和最新版本的变化

`git commit -m [message-string]`

`git commit -a -m [message-string]`

提交更新

`git rm [path]`

删除文件（commit后生效）

`git mv [from-path] [from-path]`

移动文件（commit后生效）

`git log`

`git log -2`

`git log -p`

`gitk`

查看版本历史，通常使用gitk可视化查看

`git commit --amend`

撤销上次提交并重新提交，新添文件或者改提交说明

`git reset Head [Path]`

取消暂存区域文件

`git checkout -- [path]`

取消已修改文件的修改

`git remote -v`

查看当前拥有的远端服务器

`git remote add [name] [url]`

添加远端服务器

`git fetch [remote-name]`

获取指定远端服务器的所有分支，不会自动进行合并操作

`git push [remote-name] [branch-name]`

更新当前分支到指定服务器的指定分支

`git remote show [remote-name]`

> 查看指定远端服务器的所有分支信息
>
> `git remote rename [old-name] [new-name]`
> 重命名远端服务器的名字
>
> `git remote rm [remote-name]`
> 删除指定的远端服务器
>
> `git tag [tag-name]`
> `git tag -a [tag-name] -m [message-string]`
> `git tag -s [tag-name]`
> `git tag -v [tag-name]`
> 轻量级标签、附属标签、签署标签、验证签署标签
>
> `git tag -a [tag-name] [commit-id]`
> 已提交的版本（通过git log取得校验和）附上标签
>
> `git push [remote-name] [tag-name]`
> 把标签推到服务器分享
>
> `连敲两下TAB键`
> 关键词自动补全
>
> `git config --global alias.[typename] [origin-name]`
> 给Git命令起别名

一、取得项目的Git仓库

> 小总结:
> `git init`创建git项目；
> `git add xxx.c`添加版本控制文件；
> `git clone [url] name`克隆开源项目。

两种方法：
1. 在现存的目录下，通过导入所有文件来创建新的Git仓库；
2. 从已有的仓库克隆出一个新的镜像仓库。

```
//对某个现有目录下的工程进行Git管理
//进入此项目所在的目录，执行：
$ git init
```

```
//初始化后，会出现.git目录，通过git add命令对需要纳入版本控制的文件进行跟踪
$ git add *.c
$ git add README
$ git commit -m 'initial project version'
```

```
//从开源的项目的Git仓库clone
//git clone[url]会在当前目录创建一个grit目录，用于保存下载下来的所有版本并从中取出最新版本的文件拷贝。如果希望自定义新建目录名称，在末尾添加名字。
$ git clone git://github.com/schacon.grit.git mygrit
```

> Git支持许多数据传输协议。上面使用的是`git://`协议，也可以使用`http(s)://`或者`user@server:/path.git`表示的SSH传输协议。

二、记录每次更新到仓库

> 小总结：
> `git status`检查文件状态
> `git add [directory]`把文件标识为暂存状态
>
> `git cat .gitignore`忽略某些文件，不提示控制
> `*.[oa]`忽略.a或.a结尾的文件
> `*~`忽略~结尾的文件
>
> `git diff`对比未暂存与已暂存的变化
> `git diff --staged`对比已暂存与上个版本的变化
> `git commit`提交更新
> `git commit -a`跳过add自动提交modify更新
> `git rm [path]`删除文件（直接可提交）
> `git mv [from-path] [to-path]`移动文件（可直接提交）

工作目录下的文件就两种状态：已跟踪或未跟踪。

```
//检查当前文件的状态，git status
$ git status
Untracked files;未跟踪文件
Changes to be committed;暂存状态
Changes not staged for commit;修改状态
```

```
//跟踪新文件、暂存已修改文件
$ git add [directory]
```

```
//忽略某些文件，创建一个名为.gitignore文件，列出要忽略的文件模式
$ cat .gitignore
*.[oa]
*~
```

文件`.gitignore`格式规范:

- 所有空行或者以注释符`#`开头的行都会被Git忽略；
- 可以使用标准的glob模式匹配；
- 匹配模式最后跟反斜杠`/`说明要忽略的是目录；
- 要忽略指定模式以外的文件或目录，可以用`!`取反。

> glob模式：shell所使用的简化的正则表达式。
>
> 1. `*`匹配零个或多个任意字符;
> 2. `[abc]`匹配方括号中的任何一个字符；
> 3. `?`匹配一个字符；
> 4. `[0-9]`匹配0到9的数字，英文也可；

```
# 一个完整的.gitignore文件例子
# 此为注释，会被忽略
*.a
!lib.a  #忽略所有a格式，除了lib.a
/TODO #仅忽略根目录下的TODO文件
bulid/ #忽略build/文件夹
doc/*.txt   #仅忽略doc/目录下的txt
```

```
doc/**/*.txt     #忽略doc目录以及子目录的txt
```

```
//查看已暂存或未暂存的更新
$ git diff //比较工作目录中当期文件和暂存区域快照之间的差异，就是修改之后还没有暂存起来的变化内容。
```

```
//提交更新。养成提交前git status的习惯
$ git commit
$ git commit -m 'aaaa'//提交说明直接写
$ git commit -v //把git diff详细内容也写进
```

```
//跳过使用暂存区域
$ git commit -a
```

```
//移除文件。从已跟踪文件中清单中移除（从暂存区域中移除），连带着从工作目录中移除，以后就不会提示未跟踪文件。
$ git rm [path]
//行为有点怪异，如果文件在暂存区域中或者在已修改的状态了，那么这个rm会报错。也就是说你要rm一个文件，那么这个文件必须是很干净的（已经是最新版本的文件且未修改）。除非用强制删除。
$ git rm -f [path]
//还有一种情况是移除版本控制，但是留在工作目录中：
$ git rm --cached [path]
//使用匹配模式删除：
$ git rm log/\*.log//删除log目录及子目录下所有log格式，反斜杠的作用是递归。
$ git rm \*~ //删除目录及子目录下的~结尾文件
```

```
//移动文件（包括重命名操作）
//移动操作的话直接就进入暂存区域了
$ git mv [file_from] [file_to]

//例子
$ git mv README.txt README
```

三、查看提交历史

> 小总结：

`git log`查看版本历史
`gitk`使用可视化工具查看

```
//回顾提交历史：
$ git log    //列出所有更新，最新排前面
$ git log -p     //列出所有差异
$ git log -2     //列出最近两次更新
$ git log -p --word-diff     //单词对比，删除的用[]括起来，添加的{}
```

四、撤销操作

小总结：
`git commit --amend`撤销刚才提交操作并重新提交
`git reset Head [path]`取消暂存区域文件
`git checkout -- [path]`取消对文件的修改

```
//撤销操作，如果和上次提交的一模一样，只是提交信息写错了。那么提交的快照和刚才一样：
$ git commit --amend

//如果发现漏了文件，最终最后一个提交生效：
$ git commit -m '123'
$ git add forgotten_file
$ git commit --amend
```

```
//取消已经暂存的文件，git status时有提示：
$ git reset Head [path]
```

```
//取消对文件的修改，会丢失数据
 git checkout -- [path]
```

五、远程仓库的使用

小总结：
`git remote -v`查看所有远端服务器

`git remote add [name] [url]`添加远端服务器
`git fetch [name]`把远程服务器的所有分支下载到本地中
`git push origin master`推数据到远端服务器
`git remote show [name]`查看远程仓库信息
`git remote rename old new`重命名远程仓库
`git remote rm [remote-name]`删除远程仓库

```
//查看每个远程库的名称
$ git remote
$ git remote -v //（--verbose啰嗦缩写）
```

```
//添加远程仓库
$ git remote add [name] [url]
```

```
//从远程仓库抓去数据，只是把远端数据拉到本地仓库，并不进行自动合并
$ git fetch [name]
//设置了某一本地分支跟踪远端分支后pull命令：
$ git pull//将远端分支自动合并到本地仓库中的当前分支。
```

`git clone`的本质是自动创建本地的master分支并用于跟踪远程仓库中的master分支。

```
//推送数据到远程仓库
$ git push [remote-name] [branch-name]
//默认下本地的仓库为origin，分支为master
$ git push origin master
//必须对仓库有写的权限，如果有人已经推送了更新，那么你只能抓去它们到本地后，才能推自己的更新。
```

```
//查看远程仓库信息
$ git remote show [remote-name]
```

```
//远程仓库的删除或者重命名
$ git remote rename old-name new-name
$ git remote rm [remote-name]
```

六、打标签

> 小总结：
> `git tag`查看所有标签
> `git tag -l 'v1.4.*'`只查找1.4版本的标签
> `git tag -a v1.4 -m '123'`打标签
> `git tag -s v1.5 -m '123'`新建带签署的标签
> `git tag -v [tag-name]`验证带签署的标签
> `git tag v1.6`新建轻量级标签
> `git tag -a v1.5 [某次提交校验和]`后期加注标签，用`git log`查看校验和
> `git push origin [tag-name]`把标签推到服务器

```
//查找所有标签
$ git tag
$ git tag -l 'v1.3.*'
```

> Git的标签分两种，轻量级和含附注。轻量级标签只是指向特定提交对象的一个引用。而含附注是仓库中一个对立的对象，包含了标签名字、地址、标签说明，对象本身也允许使用签署或验证。

```
//含附注的标签：
$ git tag -a v1.4 -m 'my version 1.4'
//使用git show查看标签版本信息：
$ git show [tag-name]

//含签署的标签，要有自己的私钥：
$ git tag -s v1.5 -m 'my version 1.5'
//验证标签，需要有签署者的公钥（存放在keyring中）：
$ git tag -v [tag-name]//查看这版本的信息

//轻量级标签：
$ git tag v1.6
```

```
//提交后忘记加上标签
$ git log //查找某次提交的校验和
```

```
$ git tag -a v1.5 -m 'message' [校验和]
```

```
//分享标签：
//一般情况下，git push不会把标签推到远端服务器上，要显示分享标签
$ git push origin v1.5//就是把v1.5这对象推上去
```

七、技巧和窍门

> 小总结：
> `自动补全功能`：直接敲两次TAB即可
> `Git命令别名`

```
//Git命令别名
$ git config --global alias.co checkout
$ git config --global alias.br branch

$ git config --global alias.unstage 'reset HEAD --'     //移除版本控制

$ git config --global alias.last 'log -1 HEAD'    //查看最后一次提交的信息

//写外部命令而非Git子命令（使用!感叹号）：
$ git config --global alias.visual '!gitk'
```