Git

Git Bash

1. 1Unix 与 Linux 风格的命令行
2. 基本命令：

- –目录
  - . ：当前目录
  - .. ： 上级目录
- dir：列出当前目录的所有文件
- cd 目录名：进入指定目录
- cd ..：回退到上一个目录，直接cd进入默认目录
- pwd：显示当前所在的目录路径
- ls(ll)：都是列出当前目录中的所有文件，只不过ll（两个ll）列出的内容更为详细
- touch：新建一个文件，如touch index.js 就会在当前目录下新建一个index.js 文件
- rm：删除一个文件，rm index.js 就会把index.js 文件删除
- mkdir：新建一个目录，就是新建一个文件夹
- rm –r：删除一个文件夹，rm–r src 删除src 目录
- rm –rf / ：删除电脑中所有文件，而一切皆文件，连系统都给你删了
- mv：移动文件，mv index.html src index.html 是我们要移动的文件，src 是目标文件夹，当然，这样写，必须保证文件与目标文件夹在同一目录下
- reset：重新初始化终端、清屏
- clear：清屏
- history：查看命令历史
- help：帮助
- exit：退出
- #：代表注释

Git CMD

1. windows 风格的命令行

Git GUI

1. 图形界面的Git

Git 配置

1. 所有的配置文件，其实都保存在本地

查看配置：

1. 查看配置：git config –l
2. 查看系统config：git config ––system ––list
3. 查看当前用户配置：git config ––global ––list

Git 相关配置文件

1. Git\etc\gitconfig：Git 安装目录下的gitconfig ––system 系统级
2. C:\Users\Administrator\.gitconfig 只适用于当前登录用户的配置 ––global 全局
3. 这里可以直接编辑配置文件，通过命令设置后会响应到这里

设置用户名与邮箱（用户标识，必要）

1. 当你安装Git 后首先要设置你的用户名和e–mail 地址。
2. 每次Git提交都会使用该信息，它被永远地嵌入到你的提交中

```git
git config --global user.name ""  #名称
git config --global user.email "" #邮箱
```

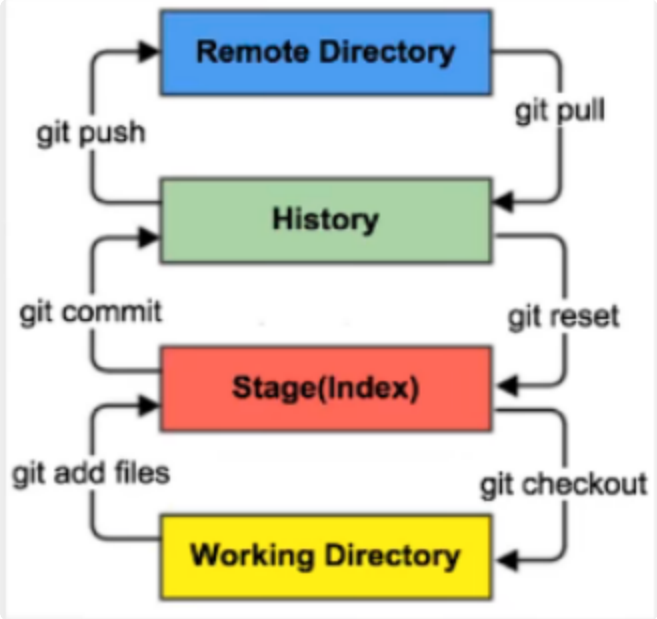
Git 基本理论

工作区域

1. Git 本地有三个工作区域：工作目录（Workinng Directory）、暂存区（Stage/Index）、资源库

（Repository或Git Directory）。如果再加上远程的git 仓库（Romote Directory），就可以分为四个工作区

2. 文件在这四个区域间的转换关系如下：



- Workspace：工作区，就是你平时存放代码的地方
- Index / Stage：暂存区，用于临时存放你的改动，事实上它只是一个文件，保存即将提交到文件列表的信息
- Repository：仓库区（或本地仓库），就是安全存放数据的位置，这里有你提交到所有的数据。其中HEAD指向最新放入仓库的版本
- Remote：远程仓库，托管代码的服务器，可以简单的认为是你项目组中的一台电脑用于远程数据交换
- 本地的三个仓库确切的说应该是git仓库中HEAD指向的版本

各文件

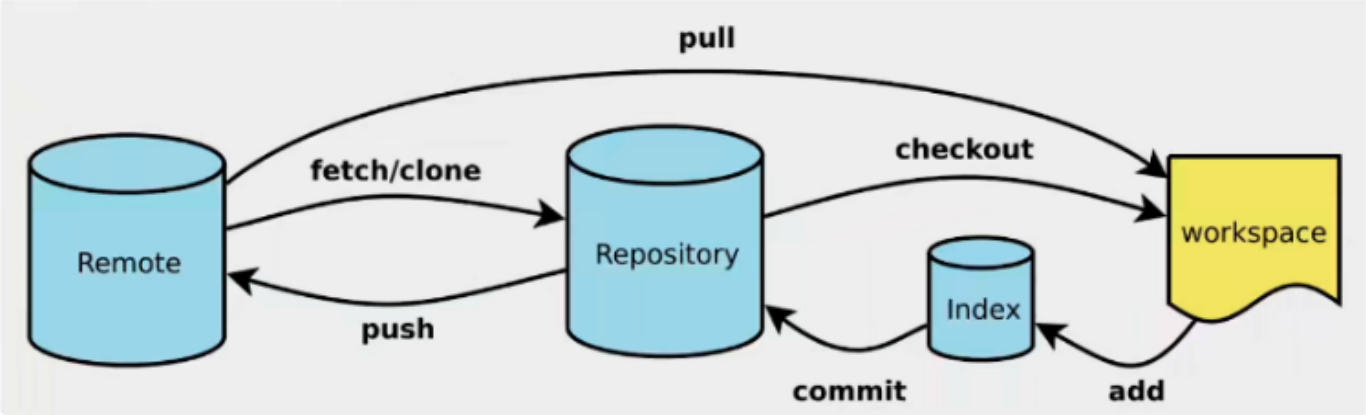
1. Directory：使用Git 管理的一个目录，也就是一个仓库，包含我们的工作空间和Git 的管理空间
2. WorkSpace：需要通过Git 进行版本控制的目录和文件，这些目录和文件组成了工作空间
3. .git：存放Git 管理信息的的目录，初始化仓库的时候自动创建（隐藏文件）
4. Index/Stage：暂存区，或者叫待提交更新区，在提交进入repo之前，我们可以把所有的更新放在暂存区
5. Local Ropo：本地仓库，一个存放在本地的版本库；HEAD会只是当前的开发分支（branch）
6. Stash：隐藏，是一个工作状态保存栈，用于保存/恢复WorkSpace中的临时状态

工作流程

1. 在工作目录中添加、修改文件；
2. 将需要进行版本管理的文件放入暂存区域
3. 将暂存区域的文件提交到git 仓库
4. 因此，git 管理的文件有三种状态：已修改（modified），已暂存（staged），已提交（committed）

Git 项目搭建

1. 工作目录（WorkSpace）一般就是你希望Git 帮助你管理的文件夹，可以是你项目的目录，也可以是一个空目录，建议不要有中文
2. 常用指令：



本地仓库搭建

创建全新的仓库

1. 创建全新的仓库，需要用GIT 管理的项目的根目录执行

```git
# 在当前目录创建一个Git 仓库
$ git init
```

2. 执行后可以看到，仅仅在项目目录多出一个.git的隐藏目录，关于版本等的所有信息都在里面

克隆远程仓库

1. 将远程服务器上的仓库完全镜像一份到本地

```git
# 克隆一个项目和它的整个代码历史（版本信息）
$ git clone [url]
```

Git 文件操作

文件四种状态

版本控制就是对文件的版本控制，要对文件进行修改、提交等操作，首先要知道文件当前在什么状态，不然可能会提交了现在还不想提交的文件，或者要提交的文件没提交上。

- **Untracked**: 未跟踪, 此文件在文件夹中, 但并没有加入到git库, 不参与版本控制. 通过`git add` 状态变为`Staged`.
- **Unmodify**: 文件已经入库, 未修改, 即版本库中的文件快照内容与文件夹中完全一致. 这种类型的文件有两种去处, 如果它被修改, 而变为`Modified`. 如果使用`git rm`移出版本库, 则成为`Untracked`文件
- **Modified**: 文件已修改, 仅仅是修改, 并没有进行其他的操作. 这个文件也有两个去处, 通过`git add`可进入暂存`staged`状态, 使用`git checkout` 则丢弃修改过, 返回到`unmodify`状态, 这个`git checkout`即从库中取出文件, 覆盖当前修改！
- **Staged**: 暂存状态. 执行`git commit`则将修改同步到库中, 这时库中的文件和本地文件又变为一致, 文件为`Unmodify`状态. 执行`git reset HEAD filename`取消暂存, 文件状态为`Modified`

查看文件状态

```git
# 查看指定文件状态
git status [filename]

# 查看所有文件状态
git status

# 添加所有文件到暂存区
git add .

# 提交暂存区中的内容到本地仓库  -m 提交信息 文件名
# 在手动合并分支后 提交到本地仓库的时候不能加文件名
git commit -m "信息内容" [filename]
```

忽略文件

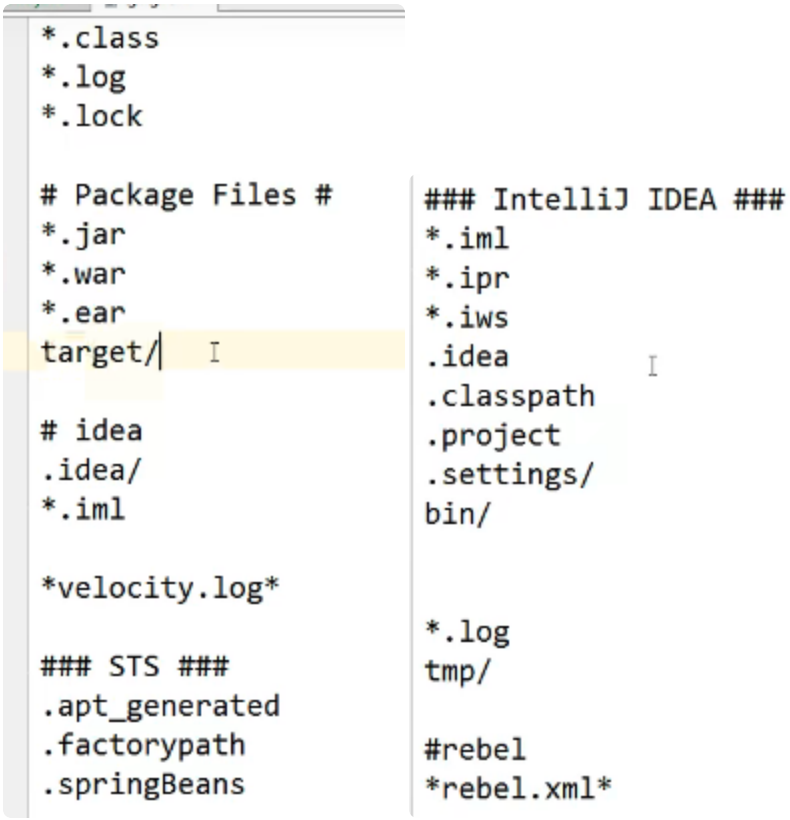
1. 有些时候我们不想把某些文件纳入版本控制中，比如数据库文件，临时文件，设计文件等
2. 在主目录下建立".gitignore"文件，此文件有以下规则：

- 忽略文件中的空行或以井号（#）开始的行将会被忽略
- 可以使用Linux 通配符。例如：星号（*）代表任意多个字符，问号（？）代表一个字符，方括号（[abc]）代表可选字符范围，大括号（{string1，string2，...}）代表可选的字符串等
- 如果名称最前面有一个感叹号（！），表示例外规则，将不被忽略
- 如果名称的最前面是一个路径分隔符（/），表示要忽略的文件在此目录下，而子目录中的文件不被忽略
- 如果名称的最后面是一个路径分隔符（/），表示要忽略的是此目录下该名称的子目录，而非文件（默认文件或目录都忽略）

Git | 复制代码

```
#为注释
*.txt        # 忽略所有.txt结尾的文件
!lib.txt     # 但lib.txt除外
/temp        # 仅忽略项目根目录下的TODO文件，不包含其他目录temp
bulid/       # 忽略bulid/ 目录下的所有文件
doc/*.txt    # 会忽略 doc/notes.txt 但不包括 doc/server/srch.txt
```

4. 参考配置：



```
*.class
*.log
*.lock

# Package Files #
*.jar
*.war
*.ear
target/

# idea
.idea/
*.iml

*velocity.log*

### STS ###
.apt_generated
.factorypath
.springBeans
```

```
### IntelliJ IDEA ###
*.iml
*.ipr
*.iws
.idea
.classpath
.project
.settings/
bin/


*.log
tmp/

#rebel
*rebel.xml*
```

5. 注意：

- .gitignore 文件从保存那一刻起即生效
- 最好是在一开始就配置 .gitignore 文件，此文件对已经纳入版本管理的文件或者文件夹无效
- 若文件或者文件夹已经纳入版本管理，又想要被忽略，可以进行如下操作：

Git | 复制代码

```
git rm -r --cached
# 将文件脱出版本管理
```

配置公钥

```Git
# 进入 C:\User\Administrator\.ssh 目录
# 生成公钥 -t rsa 是加密
$ ssh-keygen -t rsa
```

仓库

1. 开源许可证一般GPL 2.0 或者GPL 3.0 够用

分支

git分支中常用指令

```git
# 列出所有本地分支
git branch

# 列出所有远程分支
git branch -r

# 新建一个分支，但仍然停留在当前分支
git branch [branch-name]

# 新建一个分支，并切换到该分支 如果该分支已存在，切换到该分支
git checkout -b [branch]

# 切换到指定分支
git switch [branch]

# 合并指定分支到当前分支
git merge [branch]

# 删除分支
git branch -d [branch-name]

# 删除远程分支
git push origin --delete [branch-name]
git branch -dr [remote/branch]
```

冲突解决

1. 原因：合并分支时，两个分支在同一个文件的同一个位置有两套完全不同的修改。git无法替我们决定使用哪一个。
2. 解决方法：认为决定新代码的内容（在git bush 中可以使用vim进入编辑器进行操作，把无关信息删除，只留下想要留下信息
3. 注意：
    a. 冲突解决后，添加到暂存区，保存到本地库
    b. 保存到本地库

保存本地库 Git 复制代码

```git
# 此时使用git commit命令时不能带文件名
git commit -m "提交信息"
```

版本控制

基本语法 Git 复制代码

```git
# 查看版本信息
git reflog

# 查看版本详细信息
git log

# 版本穿梭 版本号是一串16进制的数字
git reset --hard [版本号]
```

vim编辑器

1. 全部命令

命令	说明
i	实现的是在光标之前的插入
I	大写的i实现在光标所在行的最前面插入
a	实现在光标后插入
A	实现在光标所在行的行尾插入
o	实现在光标所在行的上方插入新行
O	是现在光标坐在行的下方插入新行

命令	说明
:w	保存编辑后的文件内容，但不退出vim编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vim时指定的文件中。
:w!	强制写文件，即强制覆盖原有文件。如果原有文件的访问权限不允许写入文件，例如，原有的文件为只读文件，则可使用这个命令强制写入。但是，这种命令用法仅当用户是文件的属主时才适用，而超级用户则不受此限制。
:wq	保存文件内容后退出vim编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vim时指定的文件中，然后退出vim编辑器。另外一种替代的方法是用ZZ命令。
:wq!	强制保存文件内容后退出vim编辑器。这个命令的作用是把内存缓冲区中的数据强制写到启动vim时指定的文件中，然后退出vim编辑器。
ZZ	使用ZZ命令时，如果文件已经做过编辑处理，则把内存缓冲区中的数据写到启动vim时指定的文件中，然后退出vim编辑器。否则只是退出vim而已。注意，ZZ命令前面无需加冒号“：”，也无需按Enter键。
:q	在未做任何编辑处理而准备退出vim时，可以使用此命令。如果已做过编辑处理，则vim不允许用户使用“:q”命令退出，同时还会输出下列警告信息：No write since last change (:quit! overrides)
:q!	强制退出vim编辑器，放弃编辑处理的结果。如果确实不需要保存修改后的文件内容，可输入“:q!”命令，强行退出vim编辑器。
:w filename	把编辑处理后的结果写到指定的文件中保存
:w! filename	把编辑处理后的结果强制保存到指定的文件中，如果文件已经存 在，则覆盖现有的文件。
:wq! filename	把编辑处理后的结果强制保存到指定的文件中，如果文件已经存在，则覆盖现有文件，并退出vim编辑器。

2. 常用命令：
    a. i 进入编辑模式
    b. :wq 保存并退出
    c. Esc键 退出编辑模式