**技术文档**

# Linux

## install

### rpm安装方式

#### wget安装

1. 下载rpm包

<http://mirrors.163.com/centos/6.2/os/x86_64/Packages/wget-1.12-1.4.el6.x86_64.rpm>

1. rpm安装

rpm ivh wget-1.12-1.4.el6.x86\_64.rpm

### yum安装方式

#### wget安装

yum -y install wget

## Shell

### 系统信息

1. arch ##显示机器的[处理器架构](https://www.baidu.com/s?wd=%E5%A4%84%E7%90%86%E5%99%A8%E6%9E%B6%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nHN9nvc1nHbkPhf3rH0s0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHD4n1fdn1R4Pj6vnWc3n1n4" \t "_blank)
2. uname -m ##显示机器的[处理器架构](https://www.baidu.com/s?wd=%E5%A4%84%E7%90%86%E5%99%A8%E6%9E%B6%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nHN9nvc1nHbkPhf3rH0s0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHD4n1fdn1R4Pj6vnWc3n1n4" \t "_blank)
3. uname -r ##显示正在使用的[内核版本](https://www.baidu.com/s?wd=%E5%86%85%E6%A0%B8%E7%89%88%E6%9C%AC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nHN9nvc1nHbkPhf3rH0s0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHD4n1fdn1R4Pj6vnWc3n1n4" \t "_blank)
4. cat /proc/cpuinfo ##显示CPU info的信息

cat /proc/cpuinfo | grep -E 'processor|model name'

[grep](#_grep)正则表达式获取processor和model name信息

1. cat /proc/interrupts ##显示中断
2. cat /proc/meminfo ##校验内存使用
3. cat /proc/swaps ##显示哪些swap被使用
4. cat /proc/version ##显示内核的版本
5. cat /proc/net/dev ##显示[网络适配器](https://www.baidu.com/s?wd=%E7%BD%91%E7%BB%9C%E9%80%82%E9%85%8D%E5%99%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nHN9nvc1nHbkPhf3rH0s0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErHD4n1fdn1R4Pj6vnWc3n1n4" \t "_blank)及统计
6. cat /proc/mounts ##显示已加载的文件系统

### 关机

1. halt ##立刻关机
2. poweroff ##立刻关机
3. shutdown -h now ##立刻关机（root用户使用）
4. shutdown -h 10 ##10分钟后自动关机

### 用户管理

#### 创建用户

useradd 用户名

-c comment 指定一段注释性描述。

-d 目录 指定用户主目录，如果此目录不存在，则同时使用-m选项，可以创建主目录。

-g 用户组 指定用户所属的用户组。

-G 用户组，用户组 指定用户所属的附加组。

-s Shell文件 指定用户的登录Shell。

-u 用户号 指定用户的用户号，如果同时有-o选项，则可以重复使用其他用户的标识号。

#### 修改用户密码

passwd 用户名

#### 设置用户密码

passwd 用户名

#### 删除用户

userdel 用户名

#### 查看所有用户

cat /etc/passwd | awk -F : '{print $1}'

### 查找

#### grep

## 权限

# Java

## JVM(Java虚拟机)

### 运行时数据区



#### 所有线程共享的数据区（堆空间线程共享）

##### 方法区(Method Area)

存放被虚拟机加载的类信息、常量、静态变量、编译后的代码数据，也称为Permanent Generation。

1. -XX:MaxPermSize=16m ##设置持久代最大为16m。

##### 堆(Heap)

存放对象实例

1. -Xmx3500m ##设置堆最大可用内存。
2. -Xms3500m ##设置堆最小可用内存。
3. -Xmn2g

##设置年轻代大小（整个堆大小=年轻代大小+年老代大小+持久代大小）。

##持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。

##此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。

1. -XX:NewRatio=4

##设置年轻代（包括Eden和两个Survivor区）与年老代的比值（不包括持久代）。设置为4，则年轻代与年老代所占比值为1：4，年轻代占整个堆栈的1/5。

1. -XX:SurvivorRatio=4

##设置年轻代中Eden区与Survivor区的大小比值。

设置为4，则两个Survivor区与一个Eden区的比值为2:4，一个Survivor区占整个年轻代的1/6。

1. -XX:MaxTenuringThreshold=0

##设置垃圾最大年龄。

如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代 。对于年老代比较多的应用，可以提高效率。如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象在年轻代的存活时间 ，增加在年轻代即被回收的概率。

#### 线程隔离的数据区（栈空间线程私有）

生命周期与线程相同。

##### 虚拟机栈(Virtual Machine Stack)

1. Java方法执行的内存模型。
2. Stack Frame(栈帧)用于存储局部变量表、操作数栈、动态链接、方法出口等信息。
3. 每一个方法从调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。
4. 局部变量表存放编译可知的各种基本数据类型、对象引用类型和returnAddress类型(指向一条字节码指令的地址)。
5. long和double占用2个局部变量空间(Slot)，其余占用1个。
6. 局部变量表所需内存空间在编译期间完成分配，当进入方法时，这个方法需要在帧中分配多大局部变量空间是完全确定的，方法在运行期间不会改变局部变量表的大小。
7. 如果线程请求的栈深度大于虚拟机所允许的深度，将抛出StackOverflowError异常；如果虚拟机栈可以动态扩展（当前大部分的Java虚拟机都可动态扩展，只不过Java虚拟机规范中也允许固定长度的虚拟机栈），如果扩展时无法申请到足够的内存，就会抛出OutOfMemoryError异常。
8. -Xss128k

##设置每个线程的堆栈大小。

##根据应用的线程所需内存大小进行调整。

##在相同物理内存下，减小这个值能生成更多的线程。

##但是操作系统对一个进程内的线程数还是有限制的，不能无限生成，经验值在3000~5000左右。

##### 本地方法栈(Native Method Stack)

1. 与虚拟机栈类似。
2. 虚拟机栈为虚拟机执行Java方法（字节码）服务。
3. 本地方法栈为虚拟机使用的Native方法服务。
4. 有的虚拟机(例如Sun HotSpot)直接把本地方法栈和虚拟机栈合二为一。

##### 程序计数器(Program Counter Register)

1. 当前线程所执行的字节码的行号指示器，记录正在执行的虚拟机字节码指令的地址，如果正在执行Native方法，这个计数器值为空(Undefined)
2. 唯一一个在Java虚拟机规范中没有规定任何OutOfMemoryError情况的区域。
3. 各个线程之间计数器互不影响，独立存储。

### 垃圾回收算法

#### 标记-清除算法(Mark-Sweep)

最基础的收集算法是“标记-清除”（Mark-Sweep）算法，如同它的名字一样，算法分为“标记”和“清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收所有被标记的对象，它的标记过程其实在前一节讲述对象标记判定时已经介绍过了。之所以说它是最基础的收集算法，是因为后续的收集算法都是基于这种思路并对其不足进行改进而得到的。它的主要不足有两个：一个是效率问题，标记和清除两个过程的效率都不高；另一个是空间问题，标记清除之后会产生大量不连续的内存碎片，空间碎片太多可能会导致以后在程序运行过程中需要分配较大对象时，无法找到足够的连续内存而不得不提前触发另一次垃圾收集动作。

#### 复制算法(Copying)

为了解决效率问题，一种称为“复制”（Copying）的收集算法出现了，它将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用完了，就将还存活着的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。这样使得每次都是对整个半区进行内存回收，内存分配时也就不用考虑内存碎片等复杂情况，只要移动堆顶指针，按顺序分配内存即可，实现简单，运行高效。只是这种算法的代价是将内存缩小为了原来的一半，未免太高了一点。

现在的商业虚拟机都采用这种收集算法来回收新生代，IBM公司的专门研究表明，新生代中的对象98%是“朝生夕死”的，所以并不需要按照1:1的比例来划分内存空间，而是将内存分为一块较大的Eden空间和两块较小的Survivor空间，每次使用Eden和其中一块Survivor。当回收时，将Eden和Survivor中还存活着的对象一次性地复制到另外一块Survivor空间上，最后清理掉Eden和刚才用过的Survivor空间。HotSpot虚拟机默认Eden和Survivor的大小比例是8:1，也就是每次新生代中可用内存空间为整个新生代容量的90%（80%+10%），只有10%的内存会被“浪费”。当然，98%的对象可回收只是一般场景下的数据，我们没有办法保证每次回收都只有不多于10%的对象存活，当Survivor空间不够用时，需要依赖其他内存（这里指老年代）进行分配担保（Handle Promotion）。

#### 标记-整理算法(Mark-Compact)

复制收集算法在对象存活率较高时就要进行较多的复制操作，效率将会变低。更关键的是，如果不想浪费50%的空间，就需要有额外的空间进行分配担保，以应对被使用的内存中所有对象都100%存活的极端情况，所以在老年代一般不能直接选用这种算法。

根据老年代的特点，有人提出了另外一种“标记-整理”（Mark-Compact）算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。

#### 分代收集算法(Generational Collection)

当前商业虚拟机的垃圾收集都采用“分代收集”（Generational Collection）算法，这种算法并没有什么新的思想，只是根据对象存活周期的不同将内存划分为几块。一般是把Java堆分为新生代和老年代，这样就可以根据各个年代的特点采用最适当的收集算法。在新生代中，每次垃圾收集时都发现有大批对象死去，只有少量存活，那就选用复制算法，只需要付出少量存活对象的复制成本就可以完成收集。而老年代中因为对象存活率高、没有额外空间对它进行分配担保，就必须使用“标记—清理”或者“标记—整理”算法来进行回收。

### 垃圾回收器

#### 串行回收器

只适用于小数据量的情况。

#### 并行回收器（吞吐量优先）

并行收集器主要以到达一定的吞吐量为目标，适用于科学技术和后台处理等。

1. XX:+UseParallelGC

##选择垃圾收集器为并行收集器。此配置仅对年轻代有效。即上述配置下，年轻代使用并发收集，而年老代仍旧使用串行收集。

1. -XX:ParallelGCThreads=20

##配置并行收集器的线程数，即：同时多少个线程一起进行垃圾回收。此值最好配置与处理器数目相等。

1. -XX:+UseParallelOldGC

##配置年老代垃圾收集方式为并行收集。

1. -XX:MaxGCPauseMillis=100

##设置每次年轻代垃圾回收的最长时间，如果无法满足此时间，JVM会自动调整年轻代大小，以满足此值。

1. -XX:+UseAdaptiveSizePolicy

##设置此选项后，并行收集器会自动选择年轻代区大小和相应的Survivor区比例，以达到目标系统规定的最低相应时间或者收集频率等，此值建议使用并行收集器时，一直打开。

#### 并发回收器(响应时间优先)

并发收集器主要是保证系统的响应时间，减少垃圾收集时的停顿时间。适用于应用服务器、电信领域等。

1. -XX:+UseConcMarkSweepGC

##CMS(Concurrent Mark-Sweep)

##设置年老代为并发收集。测试中配置这个以后，-XX:NewRatio=4的配置失效了，原因不明。所以，此时年轻代大小最好用-Xmn设置。

1. -XX:+UseParNewGC

##设置年轻代为并行收集。可与CMS收集同时使用。jdk5.0以上，JVM会根据系统配置自行设置，所以无需再设置此值。

1. -XX:CMSFullGCsBeforeCompaction=5

##由于并发收集器不对内存空间进行压缩、整理，所以运行一段时间以后会产生“碎片”，使得运行效率降低。此值设置运行多少次GC以后对内存空间进行压缩、整理。

1. -XX:+UseCMSCompactAtFullCollection

##打开对年老代的压缩。可能会影响性能，但是可以消除碎片。

### 辅助信息

#### -XX:+PrintGC

##输出形式：[GC 118250K->113543K(130112K), 0.0094143 secs]

                [Full GC 121376K->10414K(130112K), 0.0650971 secs]

#### -XX:+PrintGCDetails

##输出形式：[GC [DefNew: 8614K->781K(9088K), 0.0123035 secs] 118250K->113543K(130112K), 0.0124633 secs]

                [GC [DefNew: 8614K->8614K(9088K), 0.0000665 secs][Tenured: 112761K->10414K(121024K), 0.0433488 secs] 121376K->10414K(130112K), 0.0436268 secs]

#### -XX:+PrintGCTimeStamps

##可与上面两个混合使用。

##输出形式：11.851: [GC 98328K->93620K(130112K), 0.0082960 secs]

#### -XX:+PrintGCApplicationConcurrentTime

##打印每次垃圾回收前，程序未中断的执行时间。可与上面混合使用

##输出形式：Application time: 0.5291524 seconds

#### -XX:+PrintGCApplicationStoppedTime

##打印垃圾回收期间程序暂停的时间。可与上面混合使用

输出形式：Total time for which application threads were stopped: 0.0468229 seconds

#### -XX:PrintHeapAtGC

##打印GC前后的详细堆栈信息

#### -Xloggc:filename

##与上面几个配合使用，把相关日志信息记录到文件以便分析。

### Eclipse Memory Analysis

#### 安装

在Eclipse help -> Eclipse Marketplace下搜索Memory

## Spring Boot

### Spring Boot四个核心

#### 自动配置

#### 起步依赖

#### 命令行界面(Spring Boot CLI)

#### Actuator

Actuator提供运行时检视应用程序内部情况的能力，包括以下细节:

1. Spring应用程序上下文里配置的Bean。
2. Spring Boot自动配置做的决策。
3. 应用程序取到的环境变量、系统属性、配置属性和命令行参数。
4. 应用程序里线程的当前状态。
5. 应用程序最近处理过的HTTP请求的追踪情况。
6. 各种和内存用量、垃圾回收、Web请求以及数据源用量相关的指标。

Actuator通过Web端点和Shell界面向外界提供信息。

## Maven

## Gradle

# Server

## Nginx

### 简介

Nginx("engine x")是一款是由俄罗斯的程序设计师Igor Sysoev所开发高性能的 Web和 反向代理 服务器，也是一个 IMAP/POP3/SMTP 代理服务器。

在高连接并发的情况下，Nginx是Apache服务器不错的替代品。

### 安装

#### 安装编译工具及库文件

yum -y install make zlib zlib-devel gcc-c++ libtool openssl openssl-devel

#### 安装PCRE

PCRE 作用是让 Nginx 支持 Rewrite 功能。

1. wget <http://downloads.sourceforge.net/project/pcre/pcre/8.35/pcre-8.35.tar.gz>
2. tar zxvf pcre-8.35.tar.gz
3. cd pcre-8.35
4. ./configure
5. make && make install
6. pcre-config –version

#### 安装Nginx

1. wget <http://nginx.org/download/nginx-1.6.2.tar.gz>
2. tar zxvf nginx-1.6.2.tar.gz
3. cd nginx-1.6.2
4. ./configure --prefix=../nginx --with-http\_stub\_status\_module --with-http\_ssl\_module --with-pcre=../pcre-8.35
5. make && make install ##编译安装
6. $home/nginx/sbin/nginx -v ##查看nginx版本

### nginx.conf配置

$home/nginx/conf/nginx.conf

user nginx nginx; ##Nginx用户及组：用户 组。window下不指定。

worker\_processes 8; ##工作进程：数目。根据硬件调整，通常等于CPU数量或者2倍于CPU。

error\_log logs/error.log; ##错误日志存放路径。

error\_log logs/error.log notice; ##错误日志存放路径。

error\_log logs/error.log info; ##错误日志存放路径。

pid logs/nginx.pid; ##pid（进程标识符）存放路径。

http

{

server

{

listen 80; ##监听端口

server\_name localhost; ##域名

index index.html index.htm index.php;

root /usr/local/webserver/nginx/html; ##站点目录

#######location前缀含义:#################

## = ：精确匹配（必须全部相等）。 ##

## ~ ：大小写敏感。 ##

## ~\* ：忽略大小写。 ##

## ^~ ：只需匹配uri部分。 ##

## @ ：内部服务跳转。 ##

location [ = | ~ | ~\* | ^~ ] uri {

}

}

}

#### location

基础知识:

1. location 是在 server 块中配置。
2. 可以根据不同的 URI 使用不同的配置（location 中配置），来处理不同的请求。
3. location 是有顺序的，会被第一个匹配的location 处理。

配置语法:

1. **location** **[ = | ~ | ~\* | ^~ ] uri** { ... }
2. **location** **@name** { ... }

前缀含义:

1. **=** **##精确匹配**

location = / {

#规则

}

# 则匹配到 `http://www.example.com/` 这种请求。

1. **~ ##大小写敏感**

location ~ /Example/ {

#规则

}

#请求示例

#http://www.example.com/Example/ [成功]

#http://www.example.com/example/ [失败]

1. **~\* ##大小写忽略**

location ~\* /Example/ {

#规则

}

# 则会忽略 uri 部分的大小写

#http://www.example.com/Example/ [成功]

#http://www.example.com/example/ [成功]

1. **^~ ##只匹配以 uri 开头**

location ^~ /img/ {

#规则

}

#以 /img/ 开头的请求，都会匹配上

#http://www.example.com/img/a.jpg [成功]

#http://www.example.com/img/b.mp4 [成功]

1. **@ ##nginx内部跳转**

location /img/ {

error\_page 404 @img\_err;

}

location @img\_err {

##规则

}

**##**以 /img/ 开头的请求，如果链接的状态为 404。则会匹配到 @img\_err 这条规则上。

### 启动Nginx

$home/nginx/sbin/nginx

### 重新载入配置文件

$home/nginx/sbin/nginx -s reload

### 重启Nginx

$home/nginx/sbin/nginx -s reopen

### 停止Nginx

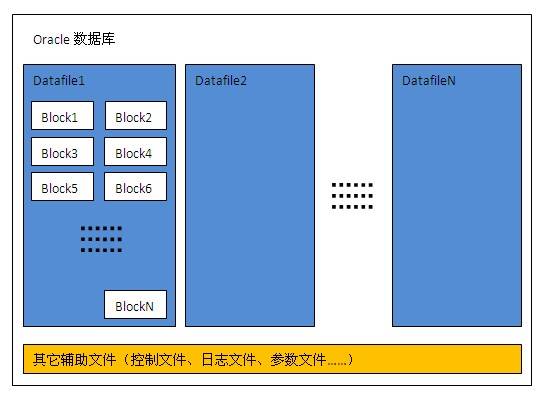
$home/nginx/sbin/nginx -s stop

# 数据库(database)

## Oracle

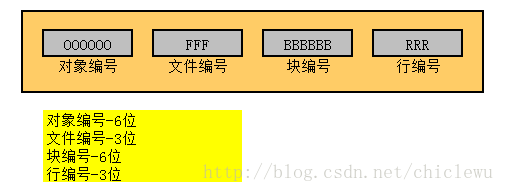
### 数据块(Block)

数据块是数据库中数据在磁盘中存储的最小单位，也是一次IO访问的最小单位，一个数据块通常可以存储多条记录，数据块大小是DBA在创建数据库或表空间时指定，可指定为2K、4K、8K、16K或32K字节。下图是一个[Oracle](http://lib.csdn.net/base/oracle)数据库典型的物理结构，一个数据库可以包括多个数据文件，一个数据文件内又包含多个数据块:



### ROWID

ROWID是每条记录在数据库中的唯一标识，通过ROWID可以直接定位记录到对应的文件号及数据块位置。ROWID内容包括对像号、文件号、数据块号、记录槽号，如下图所示：



### 索引

#### 索引使用条件

1. INDEX\_COLUMN = ?
2. INDEX\_COLUMN > ?
3. INDEX\_COLUMN >= ?
4. INDEX\_COLUMN < ?
5. INDEX\_COLUMN <= ?
6. INDEX\_COLUMN between ? and ?
7. INDEX\_COLUMN in (?,?,...,?)
8. INDEX\_COLUMN like ?||'%'（后导模糊查询）
9. T1. INDEX\_COLUMN=T2. COLUMN1（两个表通过索引字段关联）

#### 索引不使用条件

|  |  |
| --- | --- |
| **查询条件** | **不能使用索引原因** |
| INDEX\_COLUMN <> ?  INDEX\_COLUMN not in (?,?,...,?) | 不等于操作不能使用索引 |
| function(INDEX\_COLUMN) = ?  INDEX\_COLUMN + 1 = ?  INDEX\_COLUMN || 'a' = ? | 经过普通运算或函数运算后的索引字段不能使用索引 |
| INDEX\_COLUMN like '%'||?  INDEX\_COLUMN like '%'||?||'%' | 含前导模糊查询的Like语法不能使用索引 |
| INDEX\_COLUMN is null | B-TREE索引里不保存字段为NULL值记录，因此IS NULL不能使用索引 |
| NUMBER\_INDEX\_COLUMN='12345'  CHAR\_INDEX\_COLUMN=12345 | Oracle在做数值比较时需要将两边的数据转换成同一种数据类型，如果两边数据类型不同时会对字段值隐式转换，相当于加了一层函数处理，所以不能使用索引。 |
| a.INDEX\_COLUMN=a.COLUMN\_1 | 给索引查询的值应是已知数据，不能是未知字段值。 |

##经过函数运算的字段要使用索引可以使用**函数索引**，这种需求建议与DBA沟通。

有时候我们会使用多个字段的组合索引，如果查询条件中第一个字段不能使用索引，那整个查询也不能使用索引。

如：我们company表建了一个id+name的组合索引，以下SQL是不能使用索引的

select \* from company where name=?

Oracle9i后引入了一种index skip scan的索引方式来解决类似的问题，但是通过index skip scan提高性能的条件比较特殊，使用不好反而性能会更差。

#### B-TREE索引

B-TREE索引也称为平衡树索引(Balance Tree)，它是一种按字段排好序的树形目录结构，主要用于提升查询性能和唯一约束支持。B-TREE索引的内容包括**根节点**、**分支节点**、**叶子节点**。

**叶子节点内容**：索引字段内容+表记录ROWID。

**根节点，分支节点内容**：当一个数据块中不能放下所有索引字段数据时，就会形成树形的根节点或分支节点，根节点与分支节点保存了索引树的顺序及各层级间的引用关系。



##### 组合索引

一个索引由多个字段组成，称为组合索引。

##### 反向索引

##### 函数索引

#### 位图索引

#### 全文索引

# 消息队列框架

# 版本管理(版本控制)

## 常用术语

### 仓库（Repository）

受版本控制的所有文件修订历史的共享数据库。

### 工作空间（Workspace)

本地硬盘或Unix 用户帐户上编辑的文件副本。

### 工作树/区（Working tree）

工作区中包含了仓库的工作文件。您可以修改的内容和提交更改作为新的提交到仓库。

### 暂存区（Staging area）

暂存区是工作区用来提交更改（commit）前可以暂存工作区的变化。

### 索引（Index）

索引是暂存区的另一种术语。

### 签入（Checkin）

将新版本复制回仓库。

### 签出（Checkout）

从仓库中将文件的最新修订版本复制到工作空间。

### 提交（Commit）

对各自文件的工作副本做了更改，并将这些更改提交到仓库。

### 冲突（Conflict）

多人对同一文件的工作副本进行更改，并将这些更改提交到仓库。

### 合并（Merge）

将某分支上的更改联接到此主干或同为主干的另一个分支。

### 分支（Branch）

从主线上分离开的副本，默认分支叫master。

### 锁（Lock）

获得修改文件的专有权限。

### 头（HEAD）

头是一个象征性的参考，最常用以指向当前选择的分支。

### 修订（Revision）

表示代码的一个版本状态。Git通过用SHA1 hash算法表示的ID来标识不同的版本。

### 标记（Tags）

标记指的是某个分支某个特定时间点的状态。通过标记，可以很方便的切换到标记时的状态。

## Git

### 创建本地ssh

ssh-keygen -t rsa -C "13322808776@189.cn

### 验证是否配置成功

ssh -T git@github.com

### 本地git仓库关联GitHub仓库

git remote add origin git@github.com:bsc2012/documents.git

### 初始化[git](http://lib.csdn.net/base/28" \o "Git知识库" \t "_blank)仓库

git init

### 添加文件

git add ./\*

### 删除文件

git rm filename

### 提交缓存

git commit -m '提交'

### 提交到远程GitHub仓库

git push -u origin master

### ****与GitHub远程仓库同步****

git pull origin master

## github

<https://github.com/bsc2012>

账号:bsc2012 密码:bsc\*\*\*\*\*\*\*213