查看下什么是间隙锁 1

什么是MVCC 1

网页怎样去重 2

二级索引 2

消息队列mq 2

朴素贝叶斯算法 3

倒排索引 3

页分裂 3

完美的单例 3

布隆过滤器 3

ajax跨域问题 3

在内存中定位数组的位置，是一下找到这个位置还是从头遍历 3

索引最重要的是区分度要高，其次最好能有顺序。 3

文件排序 3

索引为什么只能是最左前缀 3

查看下事务的原理 3

什么是句柄 3

18、HTTP与socket的区别 3

oss与cdn的区别 4

什么是CDN 4

查看域名对应的IP 4

消息队列Linux实现原理 4

共享内存实现原理 5

信号量实现原理 5

剑指offer 5

查看shadowsocket源码 5

查看blender源码 5

查看open VPN源码 5

看证书的原理 5

对称加密 5

非对称加密 5

Linux中的inode 5

手机抓包原理 6

## 查看下什么是间隙锁

## 什么是MVCC

全称：Multi-Version Concurrent Control。原理类似于Java中的copyonwrite，在写操作的过程中先复制一份原来的数据，读操作会读取复制的数据，这样来保证写操作不会阻塞和影响读操作，该机制一般用于读操作多于写操作的场景中。

在MySQL中，update操作会分成两步，insert和delete。

## 网页怎样去重

## 二级索引

MySQL中每个表都有一个聚簇索引，除此之外的索引都叫非聚簇索引也叫二级索引。

在InnoDB引擎中，主键就是聚簇索引。如果没有定义主键，那么就会选第一个唯一索引并且只含非空列作为主键。如果没有这样的列，MySQL就会自己创建一个主键而且是隐藏的。

## 消息队列mq

## 朴素贝叶斯算法

## 倒排索引

## 页分裂

## 完美的单例

## 布隆过滤器

## ajax跨域问题

## 在内存中定位数组的位置，是一下找到这个位置还是从头遍历

## 索引最重要的是区分度要高，其次最好能有顺序。

## 文件排序

## 索引为什么只能是最左前缀

## 查看下事务的原理

## 什么是句柄

## 18、HTTP与socket的区别

HTTP为短连接，客户端发送请求给服务器，服务器回送响应，请求结束，主动释放连接。通常的做法是，不需要任何数据，也要保持每隔一段时间向服务器发送“保持连接”请求。这样可以保证客户端在服务端是“上线”状态。

Socket是对TCP/IP协议的封装，Socket只是个接口不是协议，通过Socket我们才能使用TCP/IP协议或者UDP协议。

## oss与cdn的区别

简单来说，OSS是存文件的，相当于网盘；CDN是给你分发内容的，相当于缓存。

## 什么是CDN

## 查看域名对应的IP

使用命令nslookup可以查看域名对应的IP或者IP对应的域名。

## 消息队列Linux实现原理

## 共享内存实现原理

## 信号量实现原理

## 剑指offer

## 查看shadowsocket源码

## 查看blender源码

## 查看open VPN源码

## 数字证书的原理

了解证书之前，首先简单介绍下服务器与客户端之间的通信过程。首先，刚开始建立连接的时候使用非对称加密确认身份，之后使用对称加密将通信内容进行加密后，开始正式通信。具体过程如下：

客户端->服务器：你好

服务器->客户端：你好

客户端->服务器：向我证明你是服务器

服务器->客户端：你好，我是服务器{实际发送内容为：“你好，我是服务器”的明文，明文的hash值经过私钥加密后的内容，hash算法，非对称加密算法}

客户端收到之后，根据非对称加密算法，用公钥进行解密，得到hash值，然后根据hash算法对明文进行签名，将签名与hash值进行对比，如果一致，证明对方是私钥的持有者。(这里为什么用hash值对明文进行签名，因为如果不使用hash值，直接对发送过来的任何文本进行加密，有可能被黑客利用，比如黑客可以使用一些有规律的内容发送给服务器，让其加密，然后对返回的内容进行分析，从而对私钥构成威胁)。

确认好身份之后，客户端使用公钥将对称加密的密钥和算法进行加密发送给服务器，之后两者通过对称加密进行通信。(之所以采用对称加密的方式进行通信，是因为公钥可以发布给任何人，每个人都有可能解密私钥加密的内容)

问题：服务器如何发布公钥呢，直接发送给客户端？挂载到网上让客户端下载？

第一个问题，黑客也可以发送给客户端公私钥对，这样客户端无法辨别公私钥对到底是谁的；第二个问题，如何判断挂载到网上的公钥一定是服务器的，也可能是黑客的。为了解决这两个问题证书诞生了。

证书的内容：

证书的发布机构、证书的有效期、公钥、证书的所有者、加密算法(非对称加密算法)、证书指纹以及指纹算法

### 对称加密

加密与解密使用相同的密钥。一般这种加密方式用在加密大量的数据的时候，因为其运算非常快。

缺点：密钥的管理很困难。比如，密钥在传输的过程中很容易被黑客截获。那么密钥的传输通常使用非对称加密。

常用的对称加密算法为DES和AES

### 非对称加密

加密和解密使用不用的密钥------公钥和私钥。

缺点：运算比较慢

常用的非对称加密算法RSA

## Linux中的inode

文件存储在硬盘上，硬盘的最小存储单位叫做“扇区”(sector)，每个扇区储存512字节。操作系统在读取硬盘的时候不会一个扇区一个扇区的读取，而是多个连续的扇区读取，即一次性读取一个“块”，“块”是文件存储的最小单位。块的大小通常是4KB，即连续8个扇区。

文件的数据储存在块中，那么文件的元信息，即文件的创建者、文件的创建日期、文件的大小等储存的区域叫做inode，即“索引节点”。

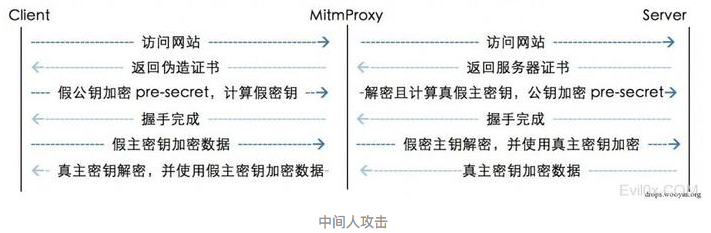
可以通过使用df –i列出inode的信息。

可以使用stat命令查看某个特定文件的inode信息，stat example.txt

inode也会消耗磁盘的大小。索引当inode区域没有可使用空间时，即使还有数据存储空间，但是也无法创建文件了。

每一个inode都有一个号码，操作系统用inode来识别文件。可以通过inode号码来删除文件，使用ls –i来列出文件的号码。

## 手机抓包原理



* 代理服务器作为一个中间人，client先向proxy发出请求，proxy再将请求转发给server
* server返回证书给proxy，proxy伪造一个证书给client。疑问：为什么可以随便伪造一个证书给client？客户端通过以下三种方式来验证证书的合法性：1、查看证书是否过期2、服务器证书上的域名是否和服务器的实际域名相匹配。3、校验证书链。这里可以手动添加信任的证书给客户端。证书当中包含服务器的公钥，来使客户端对后面的对称加密的密钥进行加密，然后传输给服务器。
* client使用假证书的公钥来对pre-secret(主密钥)进行加密，之后proxy解出该pre-secret，使用真的公钥进行加密传输给服务器。服务器使用私钥进行解密。
* 握手完成

后面的两步就是使用对称加密进行通信了。

# OAEP

# Linux进程调度

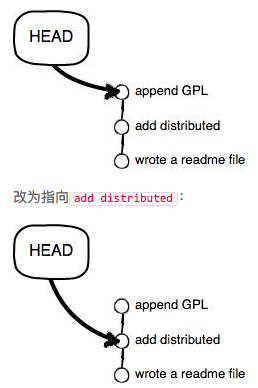
# 函数式编程

# git使用教程

git init命令可以使一个目录编程git仓库

版本回退使用git reset命令，后面加版本号，HEAD表示当前版本。git reset --hard HEAD(HEAD也可以替换成具体的版本号)

git的回退是非常快的，比如，这里有三个提交，append GPL, add distributed, wrote a read me file,git在内部有个指向当前版本的HEAD指针，当回退版本的时候，git仅仅是把HEAD由append GPL指向add distributed，顺便把工作区的文件更新了，如下图，



如果还想回到最新的append GPL版本应该怎么样呢？一般的操作是git log找到最新的commit id。但是这是已经回退到之前的版本，无法查看最新的版本了，怎么办？这时使用git reflog命令查看每一次操作的git命令，就可以找到最新版本的commit id了。

## 工作区和暂存取

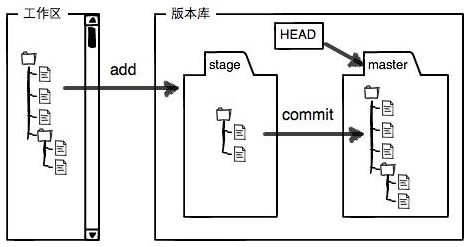
### 工作区

就是在电脑里能看到的目录。

### 版本库

工作区有一个隐藏目录.git，这个不算工作区，而是Git版本库。

Git的版本库中有很多东西，其中最重要的就是stage(暂存区)，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



git add就是把文件放到stage，git commit就是把stage的内容提交到当前分支。

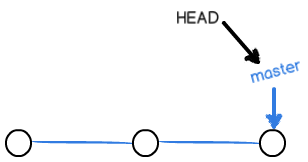
### 远程仓库

Git是分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上，通过克隆就可以实现了。

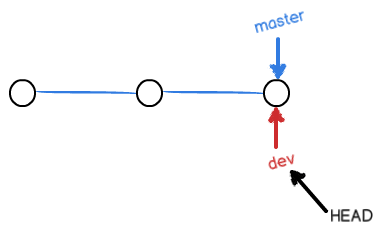
git支持多种协议，比如<https://和git://>，但是git协议更快

## 分支管理

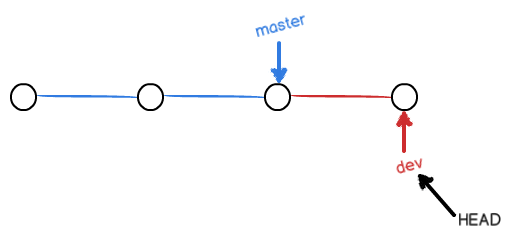
之前已经知道，每次提交，git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前只有一条时间线，在Git里，这个分支就叫做主分支master。



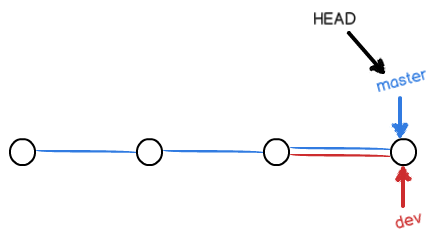
当我们创建新的分支例如dev时，Git新建一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上。



新提交一次，dev指针向前移动一步，而master指针不变，



假如我们在dev上的工作完成了，就可以把dev合并到master分支上，最简单的方法就是把master指向dev的当前提交，就完成了合并：



创建并且切换dev分支，git checkout –b dev，相当于两个命令，创建分支git branch dev，切换分支git checkout dev。

将dev分支合并到master分支上，首先切换回master分支，然后使用命令git merge dev就完了合并。Fast-forward表示这次合并是“快进模式”，就是直接将master指向dev的当前分支，所以合并速度很快。合并完成之后就可以删除dev分支，git branch –d dev

# redis数据结构

# 访问一个网页的过程

# Lucene原理

# 随机数原理

# 微服务

# 协程原理

协程可以认为是比线程更小的执行单元，既然是一个执行单元，那么协程自带CPU上下文。当一个协程切换到另一个协程时，只需要保存和恢复CPU上下文，协程就可以继续执行。

协程的切换比线程消耗更少的资源。因为线程的切换不只是CPU上下文的切换，操作系统为了程序的高效性，每个线程都有自己的缓存Cache资源，操作系统还需要帮线程恢复这些资源。所以协程知识单纯的操作CPU上下文，操作系统一秒钟切换上百万次都没问题。

# 回调函数，OS自动调用

# 分布式ID生成器

# 拜占庭将军问题

故事大概：拜占庭帝国非常富有，周围有10个邻国，都对其垂涎欲滴。但是拜占庭的城池固若金汤，任何单独的一个邻邦都不能入侵成功。同时也有可能被其他9个邻邦入侵。10个邻国当中至少有一半以上同时入侵才能成功。

然而其中一个或者几个邻邦本身答应好一起进攻，但实际过程出现背叛，那么入侵可能会被歼灭。于是每个都小心谨慎，不敢轻易相信邻国。

在该问题中，各邻国最重要的事情是：所有将军如何通过**达成共识**去攻打拜占庭帝国。

其中邻国之间如果出现叛徒，那么就会出现下列问题：

* 叛徒可能欺骗某些将军自己将采取行动；
* 叛徒可能怂恿其他将军行动；
* 叛徒可能迷惑其他将军，是他们接受不一致的信息，从而感到迷惑。

针对拜占庭问题的深入研究，科学家得出结论：**如果叛徒的数量大于或等于1/3，拜占庭问题不可解**。

解决过程可以用一个军官模型来解释：

假设有一名军官A，手下有两名士兵B和C。

A给两个士兵下达进攻指令，假如B是叛徒，那么B可能告诉C自己收到的是撤退命令，那么C就会产生疑惑，自己不知道是进攻还是撤退。

如果A是叛徒，给B的命令是进攻，给C的命令是撤退，那么B和C交换命令的时候各自都会产生疑惑。

所以在上述模型中，三者当中只要有一个人是叛徒那么拜占庭问题就不可解。

# paxos一致性协议

# 虚拟化技术

# 分布式锁

# CRC

# nosql底层结构

# http代理

Charles就是一个http代理，在客户端和服务器之间进行转发

# 幂等性

# 同源策略