Unix

CBourne Again Shell. bash)

Notebook

keep on going

make it done

History

Unix是一种操作系统

苹果系统是基于 Unix 系统开发的

安卓系统是基于 Linux 系统开发的

Linux是由Unix 系统演变出来的

Unix start by Ken Thompson and Dennis Ritchie.

↓ utf-8 character coding

↓

C programming language.

multics：首在同时运行多个程序的操作系统

UNIX = Uniplexed Information and computing Service

POSIX = Portable Operating System Interface

C发布的一系列标准中最受欢迎的

↑

从编程的角度重新定义了UNIX的功能

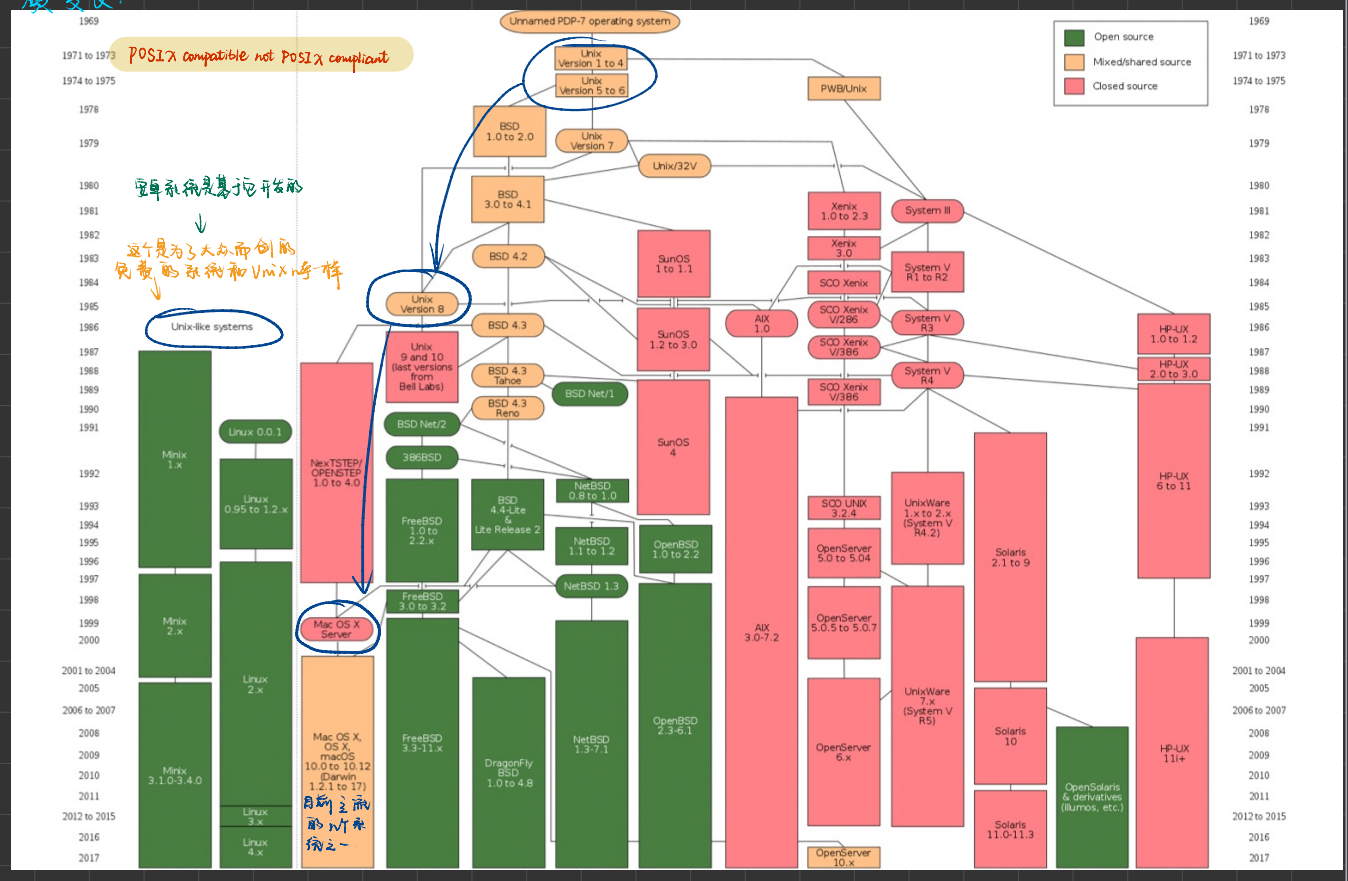
Linux不是Unix的一个版本而相当于一个复制品

history：

起源：(multics) 贝尔实验室 → 在PDP7上编程写出了第一个版的 Unix 操作系统 → 向高校提供源代码

POSIX ← 版本之间出现兼容性 ← 最终形成5个分支 ← 形成很多分支

演变史：



安卓系统是基于Linux开发的

↓

这个是为了大众而创的免费的系统和Unix一样

↓

Unix-like systems

Linux：POSIX compatible not POSIX compliant

Linux is actually just the kernel so "Linux" is actually GNU/Linux

内核

GNU (Gnu's Not Unix) is the set of applications and utilities (pretty much everything bar the kernel)

应用程序 实用程序

→ 所以 Linux 系统其实是 Linux + GNU (Linux只有内核)

using：

- Traditionally console (command-line using a shell) but also lots of GUI goodness nowadays
- The shell remains the standard interface though for real admins □

依旧是传统的控制台，但 shell 作为标准接口与管理员对接。

SSH: Secure Shell

我们可以操作交互式控制台使得可以直接访问它.

connecting to Shell (SSH) 远程连接 (1995年开始使用)

It is the standard way of connecting to a remote Unix server (we use it connect to the Shell server.
It can be used to open an interactive terminal (cshell) but also to copy

像端口转发 ← 电传打字网络是他的来源.

※ telent: 远程登陆
- All communication send in clear text
- Anyone on the network could read the passing traffic (MITM)
  (任何人都可以在运输过程中读取这个流量 所以不是很安全)

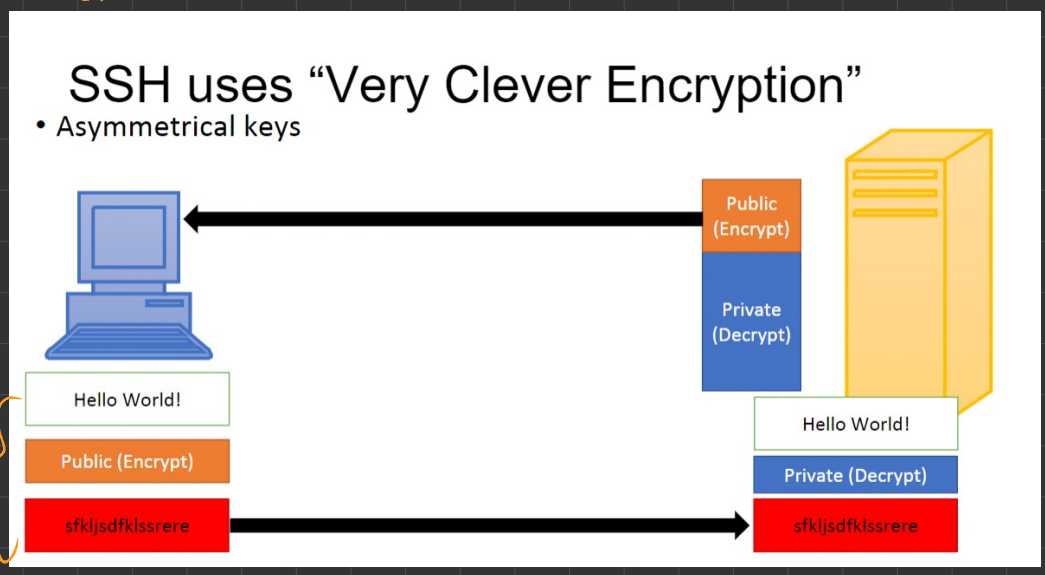
* We still use telnet testing/debugging mail servers and webservers

All communication using SSH is encrypted (more secure)

※ remote Access:

SSH = Secure SHell. All communication using SSH is encrypted.

Encryption:



这个和电报有些相似,两边有对应的密码本→就是所谓的解密和编码过程,而两边的交流依旧照原本的传播,只是没有所谓的解编码交流时的东西是无法知道实际要传递的内容 (这个是很容易破解的现在)

↓

这个过程可以安全地交换我们的密钥,但不能保护真不被拦截和攻击.任何人都可以获得我们的内容

SSH 使用 RSA 很巧妙

它生成密钥后分成两个部分:

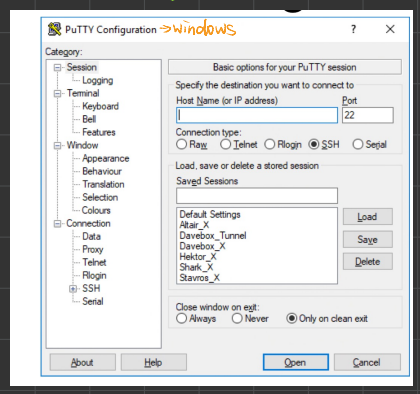
私有的 公共的

密钥有一个公共部分可以用于加密但不能解密 这使我们的交流十分安全

example:

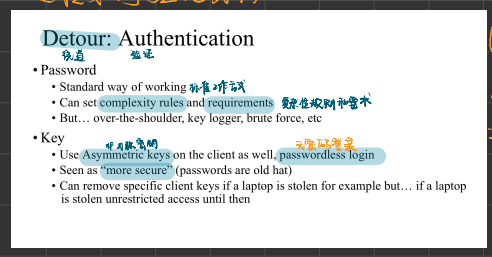
connecting via SSH:

可以看见我们在登录服务器前很少输入密码,因为已经建立基于密钥的登录 (如果电脑被偷可以从服务器中删除密钥)




```
$ ssh dcutting@playground.davecutting.uk
```

远程如何验证身份:



(我们提供公钥或部分远程服务器,那么公钥这使我们可以连接到验证我们的私钥生成公钥)

You provide the public key or part of the public key to remote server and that allows us on connection to validate we have the private key which generated the public key that you have

共享密钥:

Wikipedia ---- SSH :

The Secure Shell Protocol (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.[2] Its most notable applications are remote login and command-line execution.

SSH applications are based on a client–server architecture, connecting an SSH client instance with an SSH server. SSH operates as a layered protocol suite comprising three principal hierarchical components: the transport layer provides server authentication, confidentiality, and integrity; the user authentication protocol validates the user to the server; and the connection protocol multiplexes the encrypted tunnel into multiple logical communication channels.

SSH was designed on Unix-like operating systems, as a replacement for Telnet and for unsecured r emote Unix shell protocols, such as the Berkeley Remote Shell (rsh) and the related rlogin and rexec protocols, which all use insecure, plaintext transmission of authentication tokens

USAGE :

SSH is typically used to log into a remote machine and execute commands, but it also supports tunneling, forwarding TCP ports and X11 connections; it can transfer files using the associated SSH file transfer (SFTP) or secure copy (SCP) protocols.[3] SSH uses the client–server model.

An SSH client program is typically used for establishing connections to an SSH daemon accepting remote connections. Both are commonly present on most modern operating systems, including macOS, most distributions of Linux, OpenBSD, FreeBSD, NetBSD, Solaris and OpenVMS. Notably, versions of Windows prior to Windows 10 version 1709 do not include SSH by default. Proprietary, freeware and open source (e.g. PuTTY,[5] and the version of OpenSSH which is part of Cygwin[6]) versions of various levels of complexity and completeness exist. File managers for UNIX-like systems (e.g. Konqueror) can use the FISH protocol to provide a split-pane GUI with drag-and-drop. The open source Windows program WinSCP[7] provides similar file management (synchronization, copy, remote delete) capability using PuTTY as a back-end. Both WinSCP[8] and PuTTY[9] are available packaged to run directly off a USB drive, without requiring installation on the client machine. Setting up an SSH server in Windows typically involves enabling a feature in Settings app. In Windows 10 version 1709, an official Win32 port of OpenSSH is available.

SSH is important in cloud computing to solve connectivity problems, avoiding the security issues of exposing a cloud-based virtual machine directly on the Internet. An SSH tunnel can provide a secure path over the Internet, through a firewall to a virtual machine.[10] SSH在云计算中很重要，可以解决连接问题，避免直接在互联网上暴露基于云的虚拟机的安全问题。SSH隧道可以通过防火墙通过互联网提供一条通往虚拟机的安全路径

The IANA has assigned TCP port 22, UDP port 22 and SCTP port 22 for this protocol.[11] IANA had listed the standard TCP port 22 for SSH servers as one of the well-known ports as early as 2001.[12] SSH can also be run using SCTP rather than TCP as the connection oriented transport layer protocol.[13]

ifconfig 可以查当前服务器网卡配置等一系列相关信息（在服务器上查）

```
[u40315041@shell ~]$ ifconfig
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 143.117.208.54  netmask 255.255.255.0  broadcast 143.117.208.255
        inet6 fe80::fd55:3e4e:144f:560  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:24:82:22  txqueuelen 1000  (Ethernet)
        RX packets 36902845  bytes 6763612310 (6.2 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33788381  bytes 4621752747 (4.3 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4607  bytes 498947 (487.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4607  bytes 498947 (487.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ip 地址

注：Linux 输入密码时不会显示，应是一种保护方式）

ssh 目前较可靠，专为远程登录会话和其他网络服务提供安全性的协议：

- 有效防止远程管理过程中的信息泄漏
- 传输数据加密，能够防止DNS和IP欺骗
- 传输数据压缩，加快传输速度

SSh 服务端配置默认为 /etc/ssh/sshd_config 可按需修改默认 22 端口号配置

```
#安装服务端/客户端 (Ubuntu)
 $ sudo apt install openssh-server/openssh-client
#查看 ssh 服务是否开启
$ netstat -tlp | grep ssh
#启动/停止/重启 ssh 服务
$ sudo /etc/init.d/ssh start/stop/restart
#命令格式
$ ssh [options] [user@hostname]
#以 colin 用户登录到 192.168.1.198 的 ssh 服务器 2222 端口
$ ssh -p 2222 colin@192.168.1.198
```

options：

- -p：指定 SSh 端口号（默认为 22）
- -i：使用指定私钥文件连接服务器（免密登录）
- user：远程服务器登录的用户名，默认为当前用户
- hostname：远程服务器地址。可以是 IP / 域名 / 别名
- exit 或 logout 命令均可退出当前登录

SSh 高级配置：

配置文件	作用
known_hosts	作为客户端。记录曾连接服务器授权。ssh第一次连接一台服务器会有一个授权提示，确认授权后会记录在此文件中，下次连接记录中的服务器时则不再需要进行授权确认提示
authorized_keys	作为服务端。客户端的免密连接公钥文件
config	作为客户端。记录连接服务器配置的别名

生成密钥

```
# 客户端生成密钥对
$ ssh-keygen

# 上传公钥到服务器
$ ssh-copy-id user@hostname   # 文件会自动上传为服务器特定文件 ~/.ssh/authorized_keys
```

生成密钥对：

```
$ ssh-keygen -t rsa -f ~/.ssh/[KEY_FILENAME] -C [USERNAME]
$ chmod 400 ~/.ssh/[KEY_FILENAME]
```

相关的传输 (一些文件传输)

FTP是非常好的协议,我们用非常稳健.
但他是未加密的
(所有人都可以读你的文件)

Copying Files to/from Unix

- The standard way of copying files always was the File Transfer Protocol (FTP) 复制文件的标准方式始终是文件传输协议
- Reasonably robust but not secure (unencrypted) 相当稳健但不安全(未加密)
- One of my worst-ever security breaches was thanks to an interception of an FTP logon from an administrative user

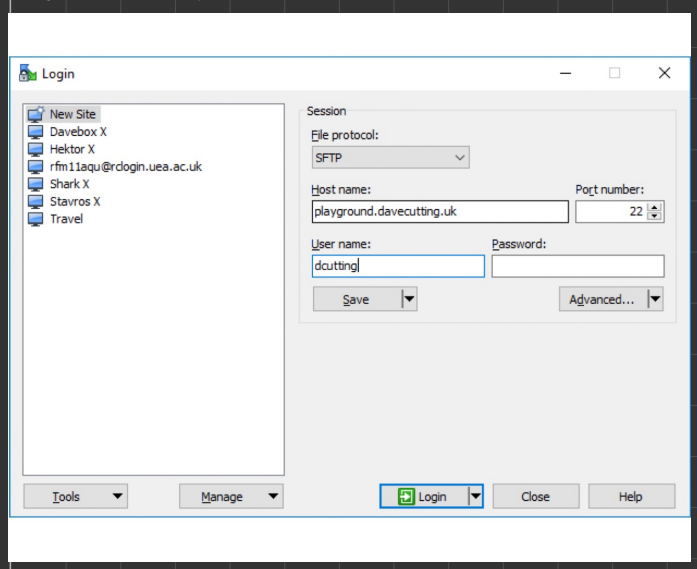
FTPS – FTP using SSL:

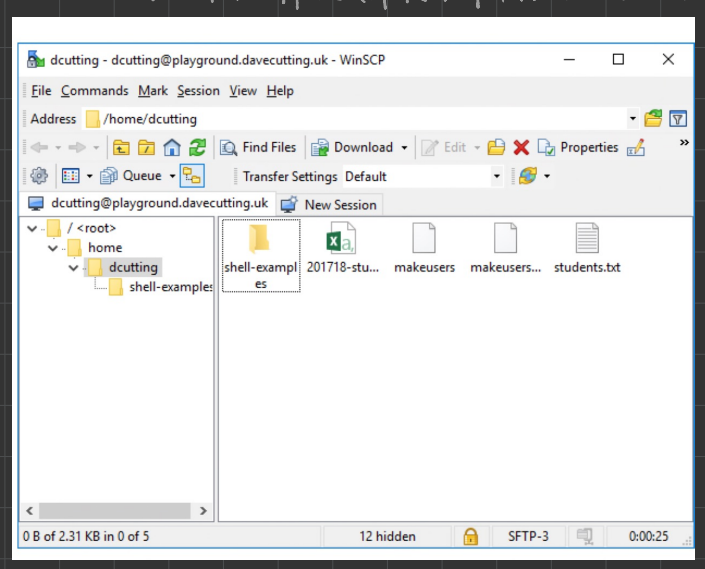
- Same protocol at heart but using SSL (the secure sockets layer) to encrypt comms between A and B (核心协议相同) (安全套接字层) (加密通信)
- Similar to the web security: http becomes https – the "language" is still http but the requests and transfer are encrypted with SSL 请求不作改变,只是请求和传输使用SSL加密
- Support issues in different FTP servers and clients – not widely used

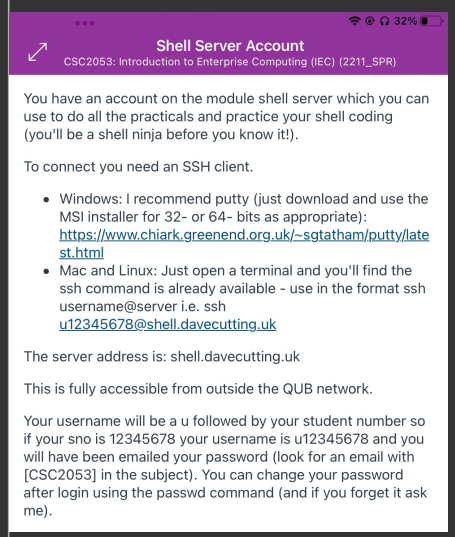
SFTP – SSH File Transfer Protocol:

- SSH works well and is widely used... so let's use that as the basis for transferring files to/from servers
- If the server is configured to support it, we're all good! (前提是服务器配置支持它)

SFTP Client on Windows - WinSCP: 和FTPS作用一样是文件传输作用但用于Windows





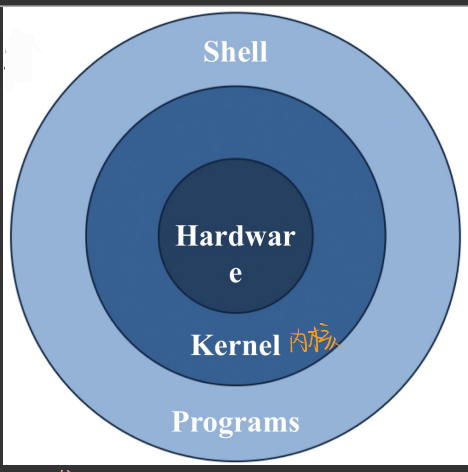


SSH account to link and commend.
远程连接的帐户

SFTP CLI Client on Unix (Linux/Unix/macOS):

```
~ $ sftp dcutting@playground.davecutting.uk
Connected to playground.davecutting.uk.
sftp> ls
201718-students.csv       makeusers                 makeusers.php
shell-examples            students.txt
sftp> get students.txt
Fetching /home/dcutting/students.txt to students.txt
/home/dcutting/students.txt                   100%  940     0.9KB/s   00:00
sftp>
```

Operating System



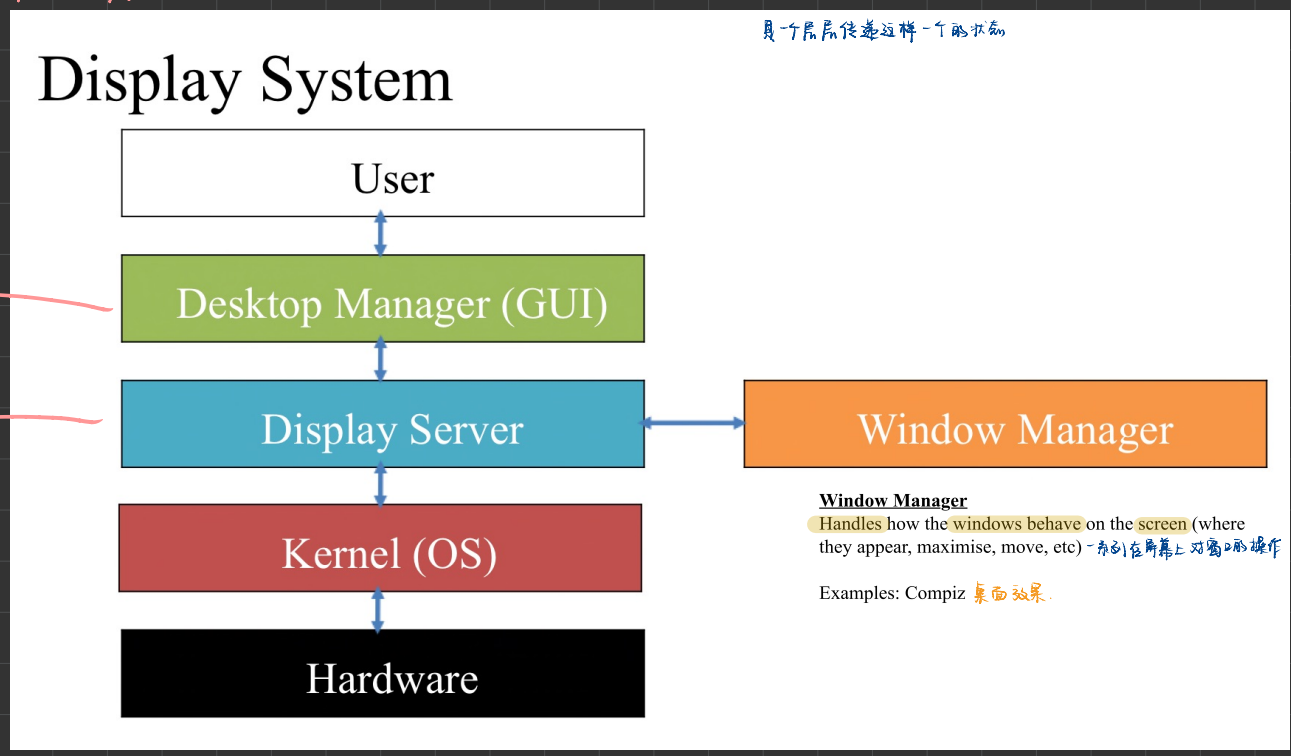
(operating system)

Components：组件

操作系统：operating system

系统软件直接安装在硬件上，可以在系统软件自身安装其他应用软件，由系统软件良成硬件和应用软件之间的交互

展示系统：



Desktop Manager 用户界面
Controls how the desktop environment behaves

Examples: KDE, Gnome

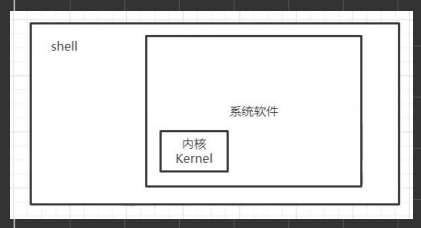
Display Server
Interfaces with the kernel to do the actual displaying (writing to the screen) 内核

Examples: X Windows (X.Org, XFree86), Wayland

Window Manager
Handles how the windows behave on the screen (where they appear, maximise, move, etc) 那些在屏幕上对窗口的操作

Examples: Compiz 桌面效果

在计算机操作的时候，我们不是直接操作内核，而是用shell来操作。



shell是有不同的版本的：

- sh
  - Bourne shell(sh)，Solaris，hpux默认shell
  - Bourne again shell (bash)，Linux 系统默认shell
- bash → command命令
- csh
  - C shell (csh)
  - tc shell (tcsh)

(在声明的时候需要注意的)
eg. #!/bin/bash

vi 文件目录：
可以直接进入文件或者创造文件然后对其直接进行更改

我们写脚本是可以对内核造成毁灭性的伤害的

执行命令文件
- 绝对路径 —— 必须是可执行文件
- sh —— 普通文件、脚本文件都可以
- source —— 普通文件、脚本文件都可以 并且直接在当前进程 执行脚本

Shell

提供命令行界面
(比基于GUI的管理更高效)
(可在任何地方远行)

shell

- the shell is the native state of a Unix admin

Shell是Unix管理员的本机状态

Actually it's more effective and efficient especially for remote or large numbers of systems

更加高效

有许多shell

- csh - the C shell
- tcsh – "T shell" compatible with csh
- ksh – Korn Shell
- bash – The Bourne Again Shell (common standard default now)
- …

Shell difference

- Syntax 语法
- Built-in Operations (most of what we do with shells normally uses external programs anyway) 内置操作
- In-built variables and referencing 内置变量和引用

example

```
#!/bin/csh
set j = 1
while ( $j <= 5 )
 echo "Hello World"
 @ j++
end
```

```
#!/bin/ksh
for i in {1..5}
do
 echo "Hello World"
done
```

```
#!/bin/bash
for (( i=1; i<=5; i++ ))
do
 echo "Hello World"
done
```

shell是一种编程语言，一般做系统的都会要使用的
shell就是操作内核的壳

vi: 创建一个脚本文件
`vi 文件名.sh` 创建一个脚本文件
创建后要在文件的第一行写声明，告诉系统其后径所指的程序即是解释此脚本文件的shell文件: eg. `#!/bin/bash`

shell脚本的执行:
- 输入脚本的绝对路径或相对路径 (执行的必须是可执行的文件)
- bash or sh + 脚本 (当脚本没有x权限时，root和文件所有者通过该方式可以正常执行)
- 在脚本的路径前加"." Or source

区别:
第一种会新开一种和第二种会新开一个bash，不同的bash中的变量无法共享
第三种是在同一个shell里面执行的 使用source时是在同一个shell里面执行的

export: 可以将当前进程的变量传递给子进程去使用 (这样子当前的变量对于子进程就是可见的)
将来配置profile的时候所有的变量前必须加export.
首先从当前进程查询变量，，如果当前进程没有这个变量，默认去父进程查询这个变量

Shell退出当前脚本:
Esc按钮 +: exit (输入)

shell脚本语言

引用变量的时候：$变量名

prompts
$开头

注意shell语言会比较依赖空格来区分一些东西

variable：变量
- 变量命名不加美元符号
- 命名只能使用英文字母、下划线、和数字，首个字符不能以数字开头
- 中间不能有空格、可以使用下划线（_）
- 不能使用标点符号
- 不能使用bash里的关键字（可以使用help命令查看保留关键字）

变量的类型：
- 局部变量：在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动的程序不能访问局部变量
- 环境变量：所有的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。
- Shell变量：shell变量是由shell程序设置的特殊变量。shell变量中有一部分是环境变量，有一部分是局部变量

只读变量：readonly 变量名　　删除变量：unset 变量名

```
eg. name = "sunwukong"
    h1 = "hello， $name"
    h2 = "hello， "$name""
    h3 = "hello， ${name}"
```

shell字符串：
字符串是最有用的数据类型，字符串可以用单引号，也可以用双引号
- 单引号：
  - 单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；
  - 单引号字符串中不能出现单独一个的单引号，但可以成对出现，作为字符串拼接使用。
- 双引号：
  - 双引号里可以有变量
  - 双引号里可以出现转义字符

字符串长度：echo ${#email}（email 是前面已经定义好的字符串）

#就是获得长度的
$就是变量读取的

shell数组：
- 定义数组：括号来表示数组，数组元素用空格来分隔开
  - 数组名 = （值1 值2 值3）
- 读取数组：${数组名[下标]}
- 获取数组中所有元素：${数组名[@]}
- 获取数组的长度：${#数组名[@/*]}

shell注释：#开头的行就是注释会被解释器忽略

多行注释：
```
: <<EOF
...
EOF
```
这个EOF是可以改的，但是前后要保持一样

adding to the path：（这个变量要代表的地址path）

```
$ path = "${path}:/opt/mysql/bin"
```

把这些都加在一起建造一个完整的item

shell脚本语言

参数传递：　脚本内获取参数(parameter)：　$n　　n代表一个数字

$#	传递到这个脚本的参数个数
$*	以一个单字符串显示所有向脚本传递的参数
$$	脚本运行的当前进程ID号
$!	后台运行的最后一个进程的ID号
$?	显示最后哦命令退出状态。0表示没有错误，其他任何值都是有错误
$0	执行的文件名

eg.　在shell文件var.sh中

通过这种方式传递参数：

```
echo "hello， $0" #这个是代表的当前执行文件的名字
echo "hello，$1"
echo "hello，$2"
```

当我们运行这个时：　./var.sh 11 22

这时候会输出：

```
hello ./var.sh  #sh文件的第一行
hello 11  #sh文件的第二行$1对应了文件后面的第一个参数
hello 22  #$2对应了文件后面的第二个参数
```

shell 符号作用：

运算符	说明	example
+	加法	'expr $a+$b'　30
-	减法	'expr $a-$b'　-10
*	乘法	'expr $a * $b'　200
/	除法	'expr $b/$a'　2
%	取余	'expr $b % $a'　0
=	赋值	a = $b　把b的值赋给a
==	相等，用于比较两个数字，相同返回true	[$a == $b]　false
!=	不相等，用于比较两个数字，不相同返回true	[$a != $b]　true

可以运算字符串 →

关系运算符	说明
-eq	等于
-ne	不等于
-gt	大于
-lt	小于
-ge	大于等于
-le	小于等于

逻辑运算符	说明
&&	and
||	or

字符串运算符	说明
=	等于
!=	不等于
-z	字符串长度为零
-n	字符串长度不为零

↑ 后面字符串要用" "引起来 eg. if[-n "$a"]…fi

文件测试运算符	说明
-r	可读性
-w	可写性
-x	可执行性
-f	是否为普通文件
-d	是不是个目录
-s	是否为空
-e	文件是否存在

→ 后面直接加文件变量 eg. if[-s $file]…fi

` ` 这个当中的内容会被当成命令去执行

shell脚本语言

一种输出方式，可以将内容输出到指定的文件当中去

echo 打印数据：

shell中的echo指令与php中的echo指令类似，都是用于字符串的输出

example：

echo [options] [string]　　补充内容

```
## 显示普通字符串
echo "Hello World"
## 显示转义字符
echo "\"Hello World\""
## 显示变量
name="zhangsan"
echo "$name Hello World"
## 显示换行
echo -e "OK! \n"
echo "Hello World"
## 显示不换行
echo -e "OK! \c"
echo "Hello World"
## 显示结果定向至文件
echo "Hello World" > myfile
## 原样输出字符串
echo '$name\"'
## 显示命令执行结果
echo `date`
```

双引号可以转义，但单引号不可以

向指定文件中输出内容

date是命令

option：-n 不换行输出

-e 启用反斜线转义解释

-E 禁用反斜线转义解释（默认）

功能说明：显示文字。

语　　法：echo [-ne][字符串]　或　echo [--help][--version]

补充说明：echo会将输入的字符串送往标准输出。输出的字符串间以空白字符隔开，并在最后加上换行号。

参　　数：-n 不要在最后自动换行　。-e 若字符串中出现以下字符，则特别加以处理，而不会将它当成一般文字输出：

\a 发出警告声；
\b 删除前一个字符；
\c 最后不加上换行符号；
\f 换行但光标仍旧停留在原来的位置；
\n 换行且光标移至行首；
\r 光标移至行首，但不换行；
\t 插入tab；
\v 与\f相同；
\\ 插入\字符；
\nnn 插入nnn（八进制）所代表的ASCII字符；
--help 显示帮助
--version 显示版本信息

test：

test命令用于检查某个条件是否成立，它可以进行数值、字符和文件三个方面测试

数字

参数	说明
-eq	等于为true
-ne	不等于为true
-gt	大于为true
-ge	大于等于为true
-lt	小于为true
-le	小于等于为true

字符串

参数	说明
=	等于为true
!=	不等于为true
-z字符串	字符串长度为0为true
-n字符串	字符串长度不为0为true

文件

参数	说明
-e filename	文件存在为true
-f filename	文件存在且可读为true
-w filename	文件存在且可写为true
-x filename	文件存在且可执行为true
-s filename	文件存在且至少有一个字符为true
-d filename	文件存在且为目录为true
-f filename	文件存在且为普通文件为true
-c filename	文件存在且为字符串型特殊文件为true
-b filename	文件存在且为块特殊文件为true

读取控制台输入：

基本语法：

read(选项)(参数)

选项：

-p：指定读取值时的提示符；

-t：指定读取值时等待的时间（秒）如果没有在指定时间内输入，就不再等待了

参数：

变量：指定读取值的变量名

shell脚本语言——流程控制

/

if:

```
if condition1
then
    command1
elif condition2
then
    command2
...
else
    commandN
fi
```

```
eg.
a = 10
b = 20
if [ $a == $b ]
then
    echo "a equal to b"
elif [ $a -gt $b ]
then
    echo "a greater than b"
else
    echo "a less than b"
fi
```

case:

```
case value in
模式1)
    command1
    command2
    ...
    commandN
    ;;
模式2)
    command1
    command2
    ...
    ;;
esac
```

```
example:
echo '输入1到4之间的数字'
echo '你输入的数字为：'
read num
case $num in
    1) echo '你选择了1'
     ;;
    2) e cho '你选择了2'
    ;;
    3) echo '你选择了3'
    ;;
    4) echo '你选择了4'
    ;;
esac
```

for:

in列表是可选的，如果不是它，for循环使用命令行的位置参数

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

```
example:
for loop in 1 2 3 4 5
do
   echo "The value is : $loop"
done

for str in 'This is a string'
do
    echo $str
done
```

这些都默认以空格来分隔

shell脚本语言——流程控制

/

while:

```
while condition
do
        command
done
```

```
example:
int = 1
while ((  $int <= 5))
do
        echo  $int
        let "int++"
done
```

function 函数:

linux shell可以用户定义函数，然后在shell脚本中可以随便调用。

可以带function fun()定义，也可以直接fun()定义，不带任何参数。

参数返回，可以显示加: return返回，如果不加，将以最后一条命令运行结果，作为返回值。return后跟数值n(0-255

注意传递参数的时候 是从1开始的:
$1就是第一个参数

注意我们参数传递的时候，没有的参数值就会当成空

```
funtionName(){
    command
    return $变量名/$((变量操作))
}
funtionName  #调用函数
```

eg

```
## 函数返回值----------------------------
funWithReturn(){
    echo "这个函数会对输入的两个数字进行相加运算..."
    echo "输入第一个数字: "
    read aNum
    echo "输入第二个数字: "
    read anotherNum
    echo "两个数字分别为 $aNum 和 $anotherNum !"
    return $(($aNum+$anotherNum))
}
funWithReturn
# 函数返回值在调用该函数后通过 $? 来获得。
echo "输入的两个数字之和为 $? !"
```

```
[root@basenode ~]# sh fun.sh
这个函数会对输入的两个数字进行相加运算...
输入第一个数字:
10
输入第二个数字:
20
两个数字分别为 10 和 20 !
输入的两个数字之和为 30 !
```

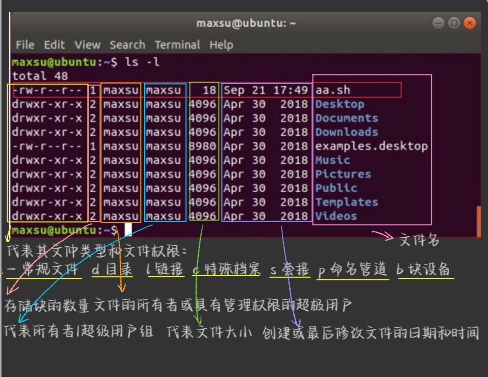
Shell -- file operations

命名、目录、导航和其余的文本命令

Based command :

- List content in current directory — ls
- Copy a file/folder — cp
- Move a file/folder — mv
- Remove an item — rm

在要操作的文件夹中直接打开的命令窗口是这个文件夹的命令位置。



ls :

ls 或 l 显示当前文件夹下的文件信息　显示文件的权限信息

ls -a 显示所有文件，包括以"."开头的隐藏文件

ls -l 显示详细信息（创建时间 权限之类的）

ll or ls -al 显示所有文件包括隐藏文件的详细信息

ll文件名　显示这个文件的详细信息

eg. ll names.txt -> -rw-rw-r--. 1 u40315041 u40315041 44 Feb 1 2021 names.txt

（这个文件名可以不止一个可以是很多个文件的集纳内容，但需要用空格间隔开每个文件）（权限）（时间）（文件名）

cd :

cd 目录名 切换到对应目录下

cd .. or cd../ 切换到上级目录

cd - 切换到上次浏览的目录并打印当前工作目录路径　（相当于返回刚建一样）

cd / 切换到根目录　（最高的目录）→起始点

cd ~ 直接回到自己的home目录

调节操作文件目录位置（更换工作区）

绝对路径：/home 即从根目录开始定位

相对路径：../home 从当前工作录开始定位到需要的目录去

pwd :

pwd 显示当前工作目录路径

touch：创建or修改文件时间属性

touch后可以接多个文件名，可一次性更改or创建多个

就是touch一下就后文件的最新时间变成了当下的时间。

touch文件名　修改文件时间属性，若文件名不存在，则新建一个新的同名文件。

eg

```
[u40315041@shell shell03]$ ll Feb9test1.txt
-rw-rw-r--. 1 u40315041 u40315041 0 Feb  9 15:14 Feb9test1.txt
[u40315041@shell shell03]$ touch Feb9test1.txt
[u40315041@shell shell03]$ ll Feb9test1.txt
-rw-rw-r--. 1 u40315041 u40315041 0 Feb  9 15:16 Feb9test1.txt
[u40315041@shell shell03]$
```

注意创命名的时候不可有"/"这样会使得出错识别成目录而不是文件

指定运行级别：　Linux一共有七个运行级别

0：关机　1：单用户（找回丢失的密码）　2：多用户无网络服务　3：多用户有网络服务

4：保留　5：图形界面　6：重启

这些都是在系统的运行级别配置文件里：　/etc/inittab

基本语法：　Init[12345]

Shell -- file operations

这些选项都是可以叠加的

cp：复制文件 重要!!!

cp 被复制的文件名 复制到的文件夹/目录 （相对路径或绝对路径） 复制生成一个一样的文件到别的目录中

cp 被复制的文件名 复制成的文件名 复制生成一个新的副本文件 (时间属性不会被复制)

eg. cp names.txt namescy.txt

```
[u40315041@shell shell03]$ ls
aaa             Febtest1cy.txt  params          users.csv
attend.sh       foo.txt         somedata.txt    values.sh
bbb             hello.sh        spock_lines.sh  while_example_1
best_male.sh    marks.txt       student_list    while_example_2
best.sh         more_marks.txt  t1              while_example_3
best_v2.sh      multiuser.sh    t2              while_example_4
faves           my_script       t3              while_example_5
Feb9test1cy.txt names           t4              while_example_6
Feb9test1.txt   names.txt       teams.txt       wq.sh
Febtest1        nums            tr.demo
[u40315041@shell shell03]$ cp names.txt namescy.txt
[u40315041@shell shell03]$ ls
aaa             Feb9test1cy.txt  more_marks.txt  params          t4              while_example_3
attend.sh       Feb9test1.txt    multiuser.sh    somedata.txt    teams.txt       while_example_4
bbb             Febtest1         my_script       spock_lines.sh  tr.demo         while_example_5
best_male.sh    Febtest1cy.txt   names           student_list    users.csv       while_example_6
best.sh         foo.txt          namescy.txt     t1              values.sh       wq.sh
best_v2.sh      hello.sh         names.txt       t2              while_example_1
faves           marks.txt        nums            t3              while_example_2
[u40315041@shell shell03]$ cat names.txt
34      Fred
89      Wilma
55      Barnie
67      Betty
28      Dino
[u40315041@shell shell03]$ cat namescy.txt
34      Fred
89      Wilma
55      Barnie
67      Betty
28      Dino
[u40315041@shell shell03]$
```

```
[u40315041@shell shell03]$ ll names.txt namescy.txt
-rw-rw-r--. 1 u40315041 u40315041 44 Feb  9 15:41 namescy.txt
-rw-rw-r--. 1 u40315041 u40315041 44 Feb  1  2021 names.txt
[u40315041@shell shell03]$
```

时间这里可以看出复制的只有内容文件的时间是不会被复制到新文件中的。

cp -p 被复制的文件名 复制成的文件名 不仅复制文件内容还复制文件的时间属性

eg. cp -p names.txt namescy2.txt

```
[u40315041@shell shell03]$ cp -p names.txt namescy2.txt
[u40315041@shell shell03]$ ls
aaa             Feb9test1cy.txt  more_marks.txt  nums            t3              while_example_2
attend.sh       Feb9test1.txt    multiuser.sh    params          t4              while_example_3
bbb             Febtest1         my_script       somedata.txt    teams.txt       while_example_4
best_male.sh    Febtest1cy.txt   names           spock_lines.sh  tr.demo         while_example_5
best.sh         foo.txt          namescy2.txt    student_list    users.csv       while_example_6
best_v2.sh      hello.sh         namescy.txt     t1              values.sh       wq.sh
faves           marks.txt        names.txt       t2              while_example_1
[u40315041@shell shell03]$ cat namescy2.txt
34      Fred
89      Wilma
55      Barnie
67      Betty
28      Dino
```

```
u40315041@shell shell03]$ ll names.txt namescy2.txt
rw-rw-r--. 1 u40315041 u40315041 44 Feb  1  2021 namescy2.txt
rw-rw-r--. 1 u40315041 u40315041 44 Feb  1  2021 names.txt
u40315041@shell shell03]$
```

时间属性被复制

是一种递归的复制 用于考虑整个文件夹

cp -r 被复制的文件夹 复制后的文件夹 复制文件夹（无时间属性）

eg. cp -r shell03 shell03cy

```
[u40315041@shell ~]$ cp -r shell03 shell03cy
[u40315041@shell ~]$ ls
shell03  shell03cy
[u40315041@shell ~]$ ll shell03 shell03cy
shell03:
```

→文件夹里面的内容一样 但是这些文件的时间属性也没有被复制过来

Eg.

cp -r test/ zwj/

无论当前的目录位置，准确的定位源目录和目标目录的位置.

当执行这种的时候若相同文件夹已经存在会依次询问当中的内容可在前面加 \ 来默认覆盖里边的文件.

eg. \cp -r test/ zwj/

cp -rp 被复制的文件夹 复制后的文件夹 复制文件夹并复制其时间属性

cp -v 被复制文件夹 复制文件夹 显示命令执行的过程

cp -rv 被复制文件夹 复制到文件夹/目录 复制文件夹到别的文件夹中

mv：移动文件 or 重命名

mv 源文件或目录 目标文件或目录 移动 or 重命名文件或目录

eg. mv one.txt twin.txt -> 将 one.txt 文件名为 twin.txt

↑这里是目录则是对文件夹的移动

mv 这个指令本就是移动文件用的，但当文件在同一个目录中移动的时候发生的就是重命名

mkdir：新建目录/文件夹

mkdir 目录名 新建一个不存在的目录(文件夹) (这个只能创建一个目录)

mkdir -p 目录 确保目录存在，不存在就新建一个 (这个可以一次性创建多级目录)

mkdir -v 显示命令过程

mkdir ⇒ make directory

-v 是显示过程的命令

Shell -- file operations

rmdir ⇒ remove directory

一级级向前把后面全部 → 删去

rm 比 rmdir 更强一些

rm 可以直接删除指定文件但 rmdir 不可以

rmdir 可以直接删除一个空的目录但 rm 不可以

rmdir：删除空目录 / 文件夹

rmdir 文件名 删除空目录 （这里的文件名就是那个空目录的位置）

rmdir -p 目录 删除指定目录下若该目录的上一级目录已经变成空目录，则将其一并删除

rmdir -v 显示命令执行的过程

rm：删除目录/文件夹

rm 文件名称 or 目录名 删除一个文件或者目录 会有提示询问 回复 y

rm -r 文件名 强制删除目录及目录下所有文件 [比 rmdir -p 好用] （递归删除整个文件夹）

rm -i 文件名 删除前注意询问确认 会出现询问，回复 y = 删除 n = 不删除

rm -f 文件名 强制删除不提醒

rm -rf 目录 删除非空的目录

rm -v 显示命令执行的过程

执行这个命令时候要仔细检查，rm是一个非常危险的命令！！！

（不小心 rm 会删除所有东西）

locate：

locate 文件名 搜索我们要找的文件的path （会显示这个文件夹的位置和里面的有什么文件）

相当于是一个关键词的搜索，有时候需要权限，会需要更新数据库来得到我们的结果

若需要更新数据库 updatedb

locate -v 显示命令的版本信息

帮助指令：

当我们对某个指令不熟悉时使用帮助指令来查看该命令如何使用

eg：这些是全英文的且很简洁

man：man 命令名 可以查看命令的帮助信息

help：help 命令名 可以查看命令的帮助信息

-rwx - wxr - x

user group other

r = 4 = 读取属性

w = 2 = 写入属性

x = 1 = 执行属性

chmod：修改文件权限

chmod [u|g|o|a] [=|+|-] [r|w|x] 文件名 修改文件 or 目录的权限信息

cat：查看文件内容 注只可以看不可修改，只有阅读功能 不可修改

cat 文件名 查看文件内容

cat -n 文件名 显示带有行号的文件内容

cat 文件名 | more 和 | more 结合使用 分页显示文件中的内容 依靠空格来进入下一页

cat > newfilename 创建一个新的文件名叫newfilename

cat 被复制文件 > 复制到的文件 复制一个文件到另一个文件 这种是覆盖形式的

cat 被复制文件 >> 复制到的文件 复制一个文件到另一个文件结尾 这种不会覆盖本来的内容

> and >> 参考追加

cat -s filename 抑制输出中重复的空行

tac filename 以相反的顺序显示内容

cat -E 文件名 突出显示最末行的内容

cat 后面的文件名可以不止一个，可以同时查看多个文件的内容，文件内容会按照文件名的先后顺序来展示 --view multiple files cat file1 file2 ... fileN

cat crontab 可以查看版本信息

more 中的快捷键：

操作	功能说明
space	向下翻一页
Enter	向下翻一行
q	立刻离开more不再显示该文件内容
Ctrl + F	向下滚动一屏
Ctrl + B	返回上一屏
=	输出当前行的行号
:f	输出文件名和当前行的行号

more：

more指令是一个基于VI编辑器的文本过滤器，它以全屏幕的方式按页显分文本文件的内容

more指令中内置了若干快捷键

more 文件名/目录 会显示一部分的文件内容 依靠快捷键操作看里面的其他内容及操作

Shell -- file operations

操作	功能说明
空白键	向下翻动一页；
[pagedown]	向下翻动一页
[pageup]	向上翻动一页；
/字串	向下搜寻「字串」的功能；n：向下查找；N：向上查找；
?字串	向上搜寻「字串」的功能；n：向上查找；N：向下查找；
q	离开 less 这个程序；

less：

less指令用来分屏查看文件内容，它的功能與more指令類似，但是比more指令更加强大，支持各種顯示終端，less指令在顯示文件内容時，並不是一次將整個文件加載之後才顯示，而是根據需要加載内容，對於顯示大型文件具有較高的效率

less 文件名

> and >> :

- > : 输出重新定向　会将原来的文件内容覆盖
- >>: 追加　不会覆盖原来文件的内容 而是追加到文件的尾部

实例：

- ls -l > 文件　　列表的内容写入到文件中（覆盖写）　若文件不存在直接覆盖 文件内容若文件不存在会创建新文件
- ls -al >> 文件　　列表的内容追加到文件的末尾
- cat 文件1 > 文件2　　将文件1的内容覆盖到文件2
- echo"content" >>文件　将content 全部写入文件尾部

tail：与head完全相反

用于输出文件中尾部的内容，默认情况下 tail指令显示文件的后10行内容

- tail 文件名　　查看文件后10行内容
- tail -n N 文件名　　查看文件后n行内容，N可以是任意行数
- tail -f 文件名　　实时追踪该文档的所有更新　（工作中常用到）

eg.

```
[root@hadoop1 home]# tail -f mydate.txt
2018年 03月 18日 星期日 23:07:27 CST
2018年 03月 18日 星期日 23:07:37 CST
2018年 03月 18日 星期日 23:07:38 CST
2018年 03月 18日 星期日 23:07:38 CST
2018年 03月 18日 星期日 23:07:40 CST
总用量 52
drwxr-xr-x. 3 root root   4096 3月  18 20:09 animal
-rw-r--r--. 1 root root    534 3月  18 23:14 a.txt
-rw-r--r--. 1 root root   1172 3月  18 23:17 b.txt
-rw-r--r--. 1 root root     18 3月  18 23:21 c.txt
-rw-r--r--. 1 root root     18 3月  18 23:21 c.xt
-rw-r--r--. 1 root root      0 3月  18 01:06 hello
-rw-r--r--. 1 root root      0 3月  18 20:45 hello.txt
-rw-r--r--. 1 root root    744 3月  18 23:22 info.txt
-rw-r--r--. 1 root root    785 3月  18 23:23 mycal
-rw-r--r--. 1 root root    215 3月  18 23:07 mydate.txt
drwxr-xr-x. 2 root root   4096 3月  18 20:39 test
drwxr-xr-x. 2 root root   4096 3月  18 04:12 tiger
drwx------. 4 xf   xf     4096 3月  18 04:45 xf
drwx------. 4  500    500 4096 3月  18 04:08 xm
drwx------. 5 zwj  wudang 4096 3月  18 20:53 zwj
```

刚打开时

在文件上作出改动后这边立马显示所有的更改内容

\n代表换行
\t代表Tab键

echo：

echo "输出内容" or 变量

head：

head用于显示文件的开头部分内容，默认情况下head指令显示文件的前10行内容

- head 文件名　　查看文件头10行内容
- head -n N 文件名　　查看文件前数行内容，N可以是任意行数

shell：搜索查找类

查找以...结尾的文件：

find -name *.txt

find：查找文件

find指令将从指定目录向下递归地遍历其各个子目录，将满足条件的文件or目录显示在终端

find [搜索范围] [搜索选项]

选项说明：（↑目录 or 文件夹）

选项	功能
-name <查询方式>	按照指定的文件名查找模式查找文件
-user <用户名>	查找属于指定用户名所有文件
-size <文件大小>	按照指定的文件大小查找文件

好就什么都不如直接跟size

大于 → +　小于 → -

后面接 size参数要带单位

locate：

locate指令可以快速定位文件路径。locate指 令利用事先建立的系统中所有文件名称及路径的locate数据库实现快速定位给定的文件。locate指令无需遍历整个文件系统，查询速度较快。为了保证查询结果的准确度，管理员必须定期更新locate时刻

locate 文件名

注意：

由于locate指令基于数据库进行查询，所以第一次运行前，必须使用updatedb指令创建locate数据库

grep：（用的比较多）查找文件中的内容

grep过滤查找，管道符，"|"表示将前一个命令的处理结果输出处转递给后面的命令处理

grep [选项] 查找内容 源文件（可以是目录）→ 直接查找

常用选项：

选项	功能
-n	显示匹配行及行号
-i	忽略字母大小写

查找的内容可以是多个，用空格间隔开来就好啦

如果是那种带有空格的就需要用双引号将整个字符串括起来

eg 查找hello.txt文件中的yes　cat hello.txt | grep yes → 管道方式

显示行号：cat -n hello.txt | grep yes

区分大小写：cat -ni hello.txt | grep yes

cut：切分文本

用指定的规则来切分文本

cut -d'分割参照'（-f1，2，3...[列数]）文件名　按照分割参照来分割文本，显示出指定的几列or全部

eg. cut -d':' -f1，2，3 password　这个时候按照":"分割文本，展示出前三列

源：这个是在文档中查询指定的数据

pipeline 管道：

管道就是把前面执行的结果传递给后面

重定向：

改变数据输出的位置，方向

0 in | out 2 err

ls / 1>lucky 标准输出

ls / >lucky 标准输出

ls abcd 2>lucky 错误输出

> 替换 >> 追加

ls /1 >> lucky

ls / 1> lucky

结合使用

ls /etc /abd > lucky 2 >&1

ls /etc /abc >> lucky 2 >&1

不管结果是否正确都输出到指定位置中去

信息黑洞

ls /etc/abc >> /dev/null 2>&1

原来显示结果的位置改变到重新定向的位置

grep"字符串"　后面我们需要输入一些字符串，然后grep会将匹配的字符串再次输出　control D 结束输入字符串然后匹配得结果

grep"字符串" < 文件名　从文件当中查找　注文件需要全名包括后面的后缀

使用管道：cat 文件名 | grep "字符串"　也是从文件中查找

Shell -- file operations

注：切割前也是当作字符串的，只有添加-n的情况下才会依照数字顺序的，所以在字符串中3排在21之后，因为字符串21的第一个是2比3要小，没有字符串第一个相同的情况下才会比较第2个

无论切割后是当字符串比较的，所以不是依照数字大小而是从头到尾一个一个的依据大小来比较的而不是单纯看整体大小

sort：对文本进行排序　后面加不加 -g是一样的

sort 文件名 对文本中的进行排序　这个时候直接按照首字母排序

sort -t'分割参照' -kn 文件名　对每一行的数据进行切分，按照第n列进行排序

sort -t'分割参照' k2 -r　文件名 逆序

sort -t'分割参照' -k2 -n 文件名　按照数字大小进行排序，如果有字母，字母在前

-n → 按照数字升序排序

wc：统计单词的数量

wc 文件名　统计单词数量

-l line

-w word　以空格来分隔单词

-c char

sed：

sed命令是操作，过滤和转换文本内容的强大工具。常用功能有增删改查（增加，删除，修改，查询），其中查询的功能中最常用的2大功能是过滤（过滤指定字符串），取行（取出指定行

sed [选项] [命令] [输入文件]

注意：输入文件可以是文件也可以是从标准管道获得的

下面两个表格中选择

sed软件有两个内置的存储空间：

- 模式空间（pattern space）：是sed软件从文本读取一行文本然后存入的缓冲区（这个缓冲区是在内存中的），然后使用sed命令操作模式空间的内容。
- 保持空间（hold space）：是sed软件另外一个缓冲区，用来存放临时数据，也是在内存中，但是模式空间和保持空间的用途是不一样的。Sed可以交换保持空间和模式空间的数据，但是不能在保持空间上执行普通的sed命令，也就是说我们可以在保持空间存储数据。

选项：

option[选项]	解释说明（带*的为重点）
-n	取消默认的sed软件的输出，常与sed命令的p连用。*
-e	一行命令语句可以执行多条sed命令
-f	选项后面可以接sed脚本的文件名
-r	使用扩展正则表达式，默认情况sed只识别基本正则表达式*
-i	直接修改文件内容，而不是输出到终端，如果不使用-i选项sed软件只是修改在内存中的数据，并不会影响磁盘上的文件*

特殊符号	解释说明（带*的为重点）
!	对指定行以外的所有行应用命令*
=	打印当前行行号
~	“First ~step”表示从First行开始，以步长Step递增
&	代表被替换的内容
;	实现一行命令语句可以执行多条sed命令*
{ }	对单个地址或地址范围执行批量操作
+	地址范围中用到的符号，做加法运算

sed -command[sed命令]	解释说明（带*的为重点）
a	追加，在指定行后添加一行或多行文本*
c	取代指定的行
d	删除指定的行*
D	删除模式空间的部分内容，直到遇到换行符\n结束操作，与多行模式相关
i	插入，在指定行前添加一行或多行文本* 所以i后需要指定行数
h	把模式空间的内容复制到保持空间
H	把模式空间的内容追加到保持空间
g	把保持空间的内容复制到模式空间
G	把保持空间的内容追加到模式空间
x	交换模式空间和保持空间的内容
l	打印不可见的字符
n	清空模式空间的内容并读入下一行
N	不清空模式空间，并读取下一行数据并追加到模式空间*
p	打印模式空间内容，通常p会与选项-n一起使用*
P（大写）	打印模式空间的内容，直到遇到换行符\n结束操作
q	退出Sed
r	从指定文件读取数据
s	取代，s#old#new#g ==> 这里g是s命令的替代标志，注意和g命令区分。*
w	另存，把模式空间的内容保存到文件中
y	根据对应位置转换字符
: label	定义一个标签
b label	执行该标签后面的命令
t	如果前面的命令执行成功，那么就跳转到t指定的标签处，继续往下执行后续命令。否则，仍然继续正常的执行流程

注 sed的运行方式：

Sed软件从文件或管道中读取一行，处理一行，输出一行；再读取一行，再处理一行，再输出一行.

效果⇒高效性 快速

一次一行的设计使得sed软件性能很高，sed在读取非常庞大的文件时不会出现卡顿的想象。大家都用过ui命令，用ui命令打开几十M或更大的文件，会发现有卡顿现象，这是因为ui命令打开文件是一次性将文件加载到内存，然后再打开，因此卡顿的时间长短就取决于从磁盘到内存的读取速度了。而且如果文件过大的话还会造成内存溢出现象。Sed软件就很好的避免了这种情况，打开速度非常快，执行速度也很快。

eg 流程：

现有一个文件person.txt，共有五行文本，sed命令读入文件person.txt的第一行“101，chensiqi，CEO”，并将这行文本存入模式空间（sed软件在内存中的一个临时缓存，用于存放读取到的内容，比喻为工厂流水线的传送带。）

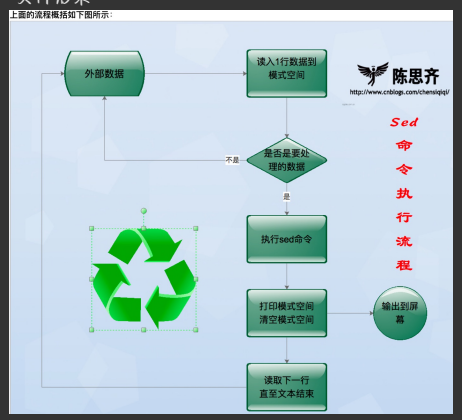
模式空间的流程：

1，判断第1行是否是需要处理的行，如果不是要处理的行就重新从文件读取下一行，如果是要处理的行，则接着往下走。

2，对模式空间的内容执行sed命令，比如a（追加），i（插入），s（替换）...

3，将模式空间中经过sed命令处理后的内容输出到屏幕上，然后清空模式空间

4，读取下一行文本，然后重新执行上面的流程，直到文件结束



输入内容结论：

- 如果引号里面是普通字符串的话，你可以任意使用单引号或者双引号；
- 如果引号里面是变量或者带反引号的命令的话，你想要变量解析的结果或者命令执行的结果，那就使用双引号；你想要引号内的原样字符串，那就使用单引号。
- 只要大家理解上面的用法，明白自己想要什么，那么你想用什么引号就用什么引号。
- 其实就二选一，非此即彼，这个不行就试试那个。

单行增加和多行增加没有太大的区别就是\的差别

参考：https://www.cnblogs.com/chensiqiqi/p/6382080.html

Shell -- file operations

/

继sed：

删除：d

这里的 sed commands 就是 d

地址范围	含义
10{sed-commands}	对第10行操作
10，20{sed-commands}	对10到20行操作，包括第10，20行
10，+20{sed-commands}	对10到30（10+20）行操作，包括第10，30行
1~2{sed-commands}	对1，3，5，7.....行操作
10，${sed-commands}	对10到最后一行（$代表最后一行）操作，包括第10行
/chensiqi/{sed-commands}	对匹配chensiqi的行操作　行中有这个字符串的才删除
/chensiqi/，/Alex/{sed-commands}	对匹配chensiqi的行到匹配Alex的行操作　就是从有chsiqi这个字符串的那行开始一直删到 A那行包括A行
/chensiqi/，${sed-commands}	对匹配chensiqi的行到最后一行操作
/chensiqi/，10{sed-commands}	对匹配chensiqi的行到第10行操作，注意：如果前10行没有匹配到chensiqi，sed软件会显示10行以后的匹配chensiqi的行
1，/Alex/{sed-commands}	对第1行到匹配Alex的行操作
/chensiqi/，+2{sed-commands}	对匹配chensiqi的行到其后的2行操作

awk：　这是一门语言　https://www.cnblogs.com/chensiqiqi/p/6481647.html

awk一般是对文件不好取但是是对文本进行操作的

awk获取本机ip：

可以直接获取到那段数字

cmp: 比较两个文件是不是完全相同的

cmp [option] 文件1[文件2[SKIP1[SKIP2]]] 输出不同的位置 和1 就是不同 输出0就是相同

cmp -b 文件.txt 文件2.txt 输出不同的行数的同时输出不同的内容

diff: 显示文件之间的差异并作出改变

diff [options] file1 file2 如果没有差异就不会输出任何东西

option: a = add c = change d = delete

test:

test -e filename (foo is a example file)

test -f foo normal file?

test -d foo directory ?

test -r foo readable file?

test -w foo writeable file

test -s foo file bigger than 0?

MD5 comparison:

- MD5 (Message Digest 5) is a hashing algorithm
- Turns data into a representative sum – in this case 128-bit
- Was originally widely used for security (passwords MD5) but was found to be vulnerable to brute force
  - Only 128 bits and weaknesses in algorithm
- Widely used for file checksum (for accidental corruption mainly!)

ln:
软链接也叫符号链接，类似于windows里的快捷方式，主要存放了链接其他文件的路径
ln -s [原文件or目录] [软链接名] 给原文件创建一个软链接
这时候创建的软链接点进去会直接链入到原文件，但pwd查看依旧会在软链接所在的目录path而不是原文件的path
如何删除软链接同样是用 rm -rf 软链接名 → 末尾不可以有/

history:
查看已经执行过历史命令，也可以执行历史命令
history 显示所有的历史命令
history n 显示最近的n条指令
! n 执行编号为n的命令

date: 显示当前日期
date 显示当前日期
date +%Y 显示当前年份
date +%m 显示当前月份
date +%d 显示当前哪一天
date "+%Y-%m-%d %H:%M:%S" 显示年月日时分秒
date -s "字符串时间" 设置时间

cal: 查看日历
cal 显示本月日历
cal 年份 显示这一年的日历

gzip/gunzip: 压缩/解压缩
grip 文件名 压缩文件，只能将文件压缩为*.gz文件
gunzip 文件名.gz 解压缩文件
注意：压缩后原文件不存在只留有结果文件

后面可以多个文件名同时存在同时对多个文件进行操作

zip/unzip: 压缩/解压缩 这个项目对于打包发布中很有用
zip [选项] XXX.zip 将要压缩的内容 压缩文件和目录的指令（压缩后的名字）
unzip [选项] XXX.zip 解压缩文件

zip常用选项:
-r 递归压缩，即压缩目录
eg. zip -r mypackage.zip /home/
会形成一个mypackage.zip文件到我们执行这个指令的位置，这个压缩文件包含了全部home文件里的内容.

unzip常用选项:
-d <目录> 指定解压后文件的存放目录

tar:

tar指令是打包指令，最后打包后的文件是.tar.gz的文件。

tar [选项] XXX.tar.gz 打包内容　打包目录，压缩后的文件格式.tar.gz

选项	功能
-c	产生.tar打包文件
-v	显示详细信息
-f	指定压缩后的文件名
-z	打包同时压缩
-x	解包.tar文件

eg. tar -zcvf a.tar.gz a1.txt a2.txt　将a1和a2的txt文件压缩成一个a.tar.gz文件

(常用的组合方式)　(生成的文件名)　(被压缩的文件(不局限于单一的,可以对多个文件操作)(这个可以是文件也可以是文件夹/目录))

tar -zxvf a.tar.gz　解压a.tar.gz文件　这是解压到当前目录

tar -zxvf myhome.tar.gz -C /opt/　解压文件到指定目录但这个指定目录必须存在

(解压的文件名)

uniq: 去掉连着重复的行

example:
$ cat faves.txt
Fred
Fred
Wilma
Wilma
Wilma
Fred
Betty
$ cat faves.txt |uniq faves.txt
Fred
Wilma
Fred
Betty

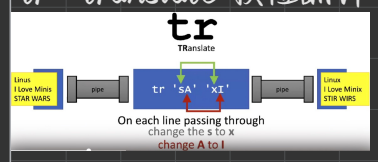
uniq 文件名　去除这个文件里面连续重复的行

nano: 编辑文件

nano 文件名　进入文件并且可以在里面编辑内容

nano是linux的文件编辑器：作用是进入文件，然后编辑文件。
执行命令后，会直接进入文件当中，但是在最底下会有一排命令提醒
我们退出文件后会回到原来的命令面板

tr: translate 换位翻译

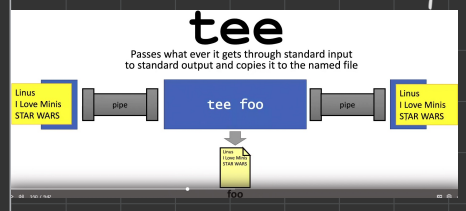


tr 'sA' 'xI'　更改s成x A成I 精准一对一更改

重新定向之类的命令都是可以叠加的，只要在中间叠加管道符号或者直接添加在后面就好

tee=

listen to standard input Send it to standard output But also copies it to a named file



tee foo　　foo用于储存中间文件

example=

cat tr.demo | tee original.txt |tr 'sA' 'xI' | tee modified.txt

修改后的内容去到 modified.txt中

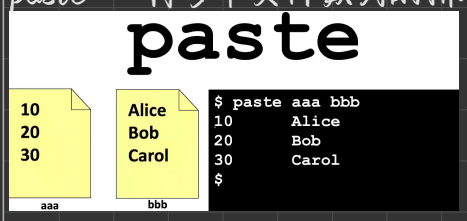
但是这里没有重新定向所以内容会直接输出给我们看

cat tr.demo | tee orginial.txt | tr 'sA' 'xI' > modified.txt

由于重新定向 修改后的内容直接输出到modified.txt文件中

不会输出显示给我们看

paste= 将多个文件数列的拼凑在一起



paste file1 file2 ...fileN　　文件按照次序纵向合并在一起

用户-组-权限

管理员可以进入任何的用户但是用户进入管理员界面是需要密码的
注意管理员的权限是更大的

用户：user

- 新增用户
  - useradd luckyboy（用户名）（用户名→自己设置）
  - 会创建同名的组和home目录
- 设置密码 —— passwd luckyboy（用户名） 输入这个命令后会出现输入密码的提示
- 删除用户
  - userdel -r luckyboy（用户名）
  - 级联删除home目录和组
- 修改用户信息
  - usermod -l luckyss lucjyls 修改用户名 —— home目录和组名称是不会被修改的
  - usermod -L luckyss 锁定用户名
  - usermod -U luckyss 解锁用户名
- 常用文件
  - cat /etc/shadow —— 用户名和密码
  - cat /etc/passwd —— 用户名，编号，组编号，home目录，命令吗，目录
    - 6.5系统0-499 普通 500+
    - 7.6系统0-999 普通1000+
- 切换账户 —— su luckyboy

组：group

- 创建组 —— groupadd 组名
- 删除组 —— groupdel 组名
- 修改组名字 —— groupmod -n 修改后的名字 修改前的名字
- 查看用户对应的组
  - groups
  - groups schoolbody —— 当我们创建用户的时候，会默认创建一个同名的组
- 修改用户组的权限
  - usermod -g 组名 组名 （主组）
  - usermod -G 组名 组名（附属组）

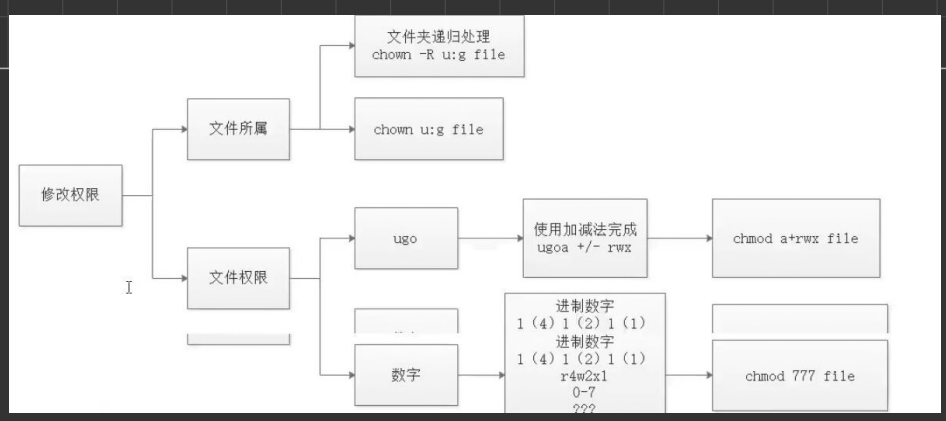
被修改的组名放后面

权限：right

权限项	读	写	执行	读	写	执行	读	写	执行
字符表示	r	w	x	r	w	x	r	w	x
数字表示	4	2	1	4	2	1	4	2	1
权限分配	文件所有者			文件所属组用户			其他用户		

权限的UGO模型
- 三组权限
- 属主权限：属组的权限：其他的权限
- 所有说：将来修改文件的权限可以从rwx和ugo两个方面进行修改

修改文件的权限
- 修改文件所属
  - chown n1 /var/lucky1
  - chown n1:m1 /var/lucky2
  - 修改文件夹时，让子目录递代修改 —— chown -R n1:m1 school
  - chgrp m2 lucky3 —— 当用户的组被修改之后，需要重新卷陆才能获取新的组的权限
- 修改文件的rwx
  - chmod o+w lucky4
  - chmod ug+rw lucky4
  - chmod ugo-rw lucky4
  - （权限RWX分别对应数字4 2 1 5=4+0+1 r-x） —— chmod 664 lucky4 -> (rw- rw- r--)



who=　　登陆列表

允许显示当前登陆到unix or Linux的 系统用户 他会显示当前登陆到系统的所有用户的信息 此外还会附加显示一些统计数据

很多时候会显示一些附加信息: 正在登录的终端和登陆时间。who 命令的输出通常取决于用户以及他们如何使用它, 区别在于使用的参数

finger=

file example

这些文件都是我们可以编辑查看的，他们会以二进制的方式出存在unix中

我们可以检测这些文件

中间有二进制数据库

这些会记录内核的日程操作

他们直接连接到数据库中

这些日志之类的文件记录了
所有的行为和数据所以我们通常通过监测数据来查看和监管
该计算机的一切行为

A crontab file

eg.

A crontab file

```
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie Exp $)
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
05 23 * * * /home/mjh/crude_backup.sh
```

Log files=

eg.

Eg. Log Files

Apache webserver

```
194.46.107.11 - csc218 [08/Mar/2007:00:00:22 +0000] "GET /modules/csc218/materials/ HTTP/1.1" 401 482
194.46.107.11 - csc218 [08/Mar/2007:00:00:28 +0000] "GET /modules/csc218/materials/ HTTP/1.1" 304 -
194.46.107.11 - csc218 [08/Mar/2007:00:00:30 +0000] "GET /modules/csc218/materials/style.css HTTP/1.1" 304 -
194.46.107.11 - csc218 [08/Mar/2007:00:00:31 +0000] "GET /modules/csc218/materials/se.ico HTTP/1.1" 304 -
194.46.107.11 - - [08/Mar/2007:00:00:32 +0000] "GET /modules/csc218/materials/new.gif HTTP/1.1" 304 -
81.99.70.75 - - [08/Mar/2007:00:00:32 +0000] "GET /modules/csc218/materials/messages.gif HTTP/1.1" 304 -
194.46.107.11 - - [08/Mar/2007:00:00:32 +0000] "GET /modules/csc218/materials/nb.gif HTTP/1.1" 304 -
81.99.70.75 - - [08/Mar/2007:00:00:32 +0000] "GET /modules/csc218/materials/tick.gif HTTP/1.1" 304 -
194.46.107.11 - - [08/Mar/2007:00:00:33 +0000] "GET /modules/csc218/materials/redball.gif HTTP/1.1" 304 -
81.99.70.75 - - [08/Mar/2007:00:00:33 +0000] "GET /modules/csc218/materials/approved_aaa.gif HTTP/1.1" 304 -
208.70.29.178 - - [08/Mar/2007:00:01:06 +0000] "GET /robots.txt HTTP/1.1" 404 -
```

Sendmail (MTA)

```
Jul 15 17:11:21 thor.foo.com sendmail[22398]: e6FFBLP22398: from=<jan(a)foo.com>,
size=589, class=0, nrcpts=1, msgid=<200007151510.e6FFAC316448(a)odin.foo.com>,
proto=ESMTP, daemon=MTA, relay=jan(a)odin.foo.com [192.168.1.1]

Jul 15 17:11:21 thor.foo.com sendmail[22400]: e6FFBLP22398: to=<gerrit(a)bar.com>,
delay=00:00:00, xdelay=00:00:00, mailer=esmtp, pri=30589, relay=frigga.bar.com.
[192.168.1.3], dsn=2.0.0, stat=Sent (e6FFAFv24566 Message accepted for delivery)
```

Exit Status Objectives

- Display the exit status of a command just run
- Interpret the status (okay vs. not okay)
- Make shell perform one set of commands or another depending whether a command was successful
- Compare number values

shell和 命令相结合

变量可以等于一个文件命令的结果：

变量名 = $(命令行)

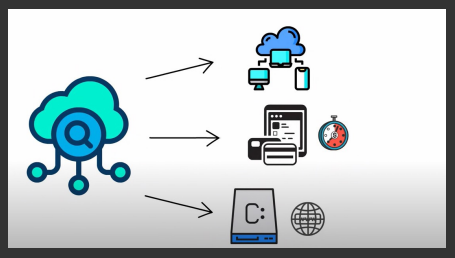
```
eg. TOPSTUDENT=$(cat mark.txt | sort -n | tail -n1 | cut f2)
    BOTTOMSTUDENT=$(cat mark.txt | sort -n | head -n1 | cut f2)
    echo The best student was $TOPSTUDENT and the lowest scoring student was $BOTTOMSTUDENT
```

我们call这个文件的时候就会执行这段代码

```
[3052149@shell demo]$ bash performance.sh
The best student was Wilma and the lowest scoring student was Dino
```

Cloud= 云服务

	Cloud Computing=	
	On-premises	Cloud Hosted
	• Physically sits in our estate • We (usually) own the hardware • We feed and water it (power and AC) • We manage the infrastructure • We handle backups • ...	• It sits elsewhere • Remote data centre(s) • We might still own and operate it but most likely it's... someone else's problem • At least parts of it (power, AC, backups, network, etc...) are someone else's problem
scalability=	可以缩小范围但是导致基础设施和维护成本方面的重大损失	允许你只需为使用量付费并提供更容易和更快的扩展方式或缩小配置
server storage=	前提系统需要大量的空间用于服务器	云计算会带来电力和维护方面的麻烦 由管理和维护服务器的云服务提供商提供 可以节省金钱和空间
data security=	本地系统由于物理和传统IT安全措施的复杂组合而提供较少的数据安全性	云计算提供更多更好的安全性，可以避免发生数据丢失情况下的持续监控和管理
Data Loss=	使用本地设置进行数据恢复的机会非常小	具有强大的灾难恢复措施以确保更快更容易的数据恢复
Maintenance=	维护本地系统还需要额外的硬件和软件 维护团队在相当程度上增加了成本	云系统由云服务供应商维护，从而降低成本 和资源分配



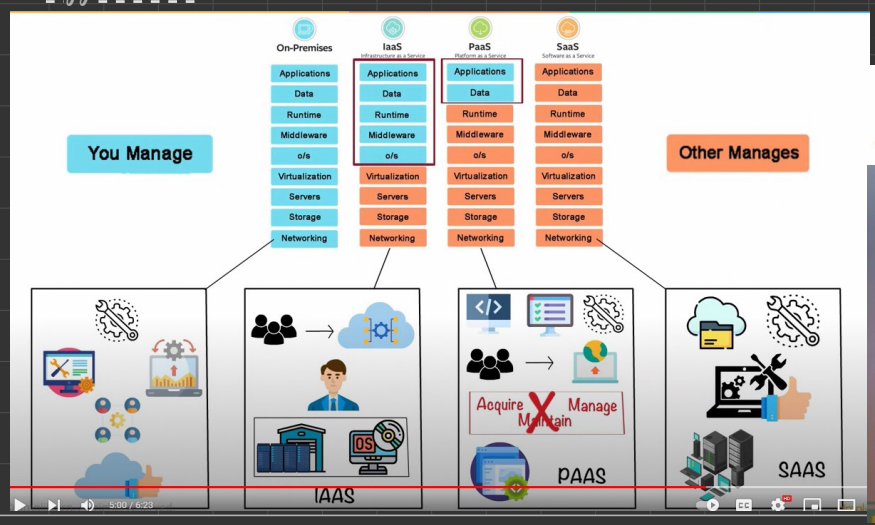
云计算是指在互联网上以随用付的方式提供按需计算器服务

cloud computing

- deployment model 部署模型
  - publice cloud 云基础设施可以通过互联网向公众提供这些由云服务供应商拥有
  - private cloud 私有云基础基础设施由一个组织独家运营这可以由ta管理e组织或第三方
  - hybrid cloud 混合云 是公共云和私有云功能的组合
- service model 服务模型 (XAAS)
  - IAAS (infrastructure as a service) 基础架构即服务如果你的组织需要储存或虚拟机等资源，用户可以访问基本的计算基础架构管理员通常使用这些基础架构。我们只需要管理数据运行时中间件应用程序和操作系统其余的由云供应商
  - PAAS (platform as a service) 运行证或平台即服务提供云平台并运行用于开发测试和管理应用程序的时间环境如果需要创建软件应用程序的平台，次服务器模型使用户无需获取管理和维护相关构架即可部署应用程序 pass只需要处理应用程序和数据其余组件确如运行时的中间件系统服务器存储和其他组件由与服务提供商处理
  - SAAS (software as a service) saas或软件服务涉及用于托管和管理我们的软件应用程序 软件的云服务提供商满足硬件要求 如果不想担心拥有任何IT设备的麻烦，那么不必管理解决方案的任何方面 saas model 与saas一起使用 云服务商处理orga所需要的解决方案的所有组件

类比公共汽车 私家车 和 出租车

difference =



最受欢迎的几个cloud=

特征

- on demand service 需要时使用
- network access 网络访问 以互联网作为媒介
- shared resources 共享资源
- scalability可拓展性 允许资源的弹性

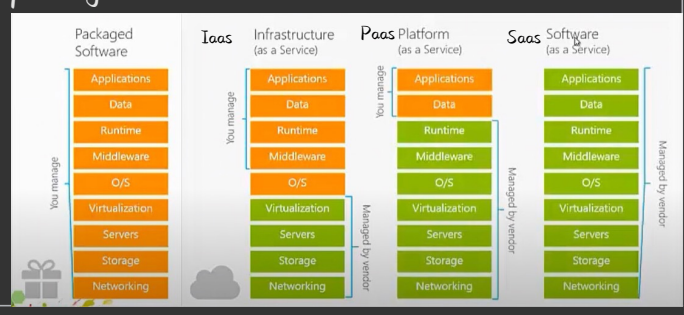
On-Premises需要管理和维护每个组件 而云计算是不需要的

cloud computing is a paradigm that allow on-demand network access to share computing resources.

允许按照网络访问共享计算及资源的范例

a model for managing, storing & processing data online via the internet.

是一种通过internet在线 管理、储存个处理数据的模型.



Cloud：云服务

/

这和我们生产东西很像，当我们生产一个东西的时候不会从原料开始就是自主生产的，那么我们就需要采集原料，超市就像我们的public cloud 由多个供应商多个解决方案（multiple vendors and multiple solutions allow you to pick and choose the right tools and solutions for your tastes）我们可以选择合适的工具和解决方案

注意的是私有云是在私有网络上构建和使用的。在企业或组织中常常会使用私有网络这样更便于传输数据。

deployment model 部署模型

public cloud：

use.th incredible resource for developers. It brings the ability to provision resources on demand and only pay for what you use. This leads to greater developer efficiency while bringing down overall costs.

他带来了按需配置资源的能力

公共雲由第三方雲服務提供商擁有和運營，這些雲服務提供商通過 Internet 提供服務器和存儲等計算資源。Microsoft Azure 是公共雲的一個示例。對於公共雲，所有硬件、軟件和其他支持基礎設施都由雲提供商擁有和管理，你可以使用網絡瀏覽器訪問這些服務并管理你的帳戶。了解有關公共雲的更多信息。

private cloud：

私有云是指由單個企業或組織專門使用的雲計算資源。私有云可以物理地位於公司的現場數據中心。一些公司還向第三方服務提供商付費以託管他們的私有云。私有云是在私有網絡上維護服務和基礎設施的雲。

Hybrid Cloud：

混合雲將公共雲和私有云結合在一起，通過允許在它們之間共享數據和應用程序的技術結合在一起。通過允許數據和應用程序在私有云和公共雲之間移動，混合雲為你的業務提供了更大的靈活性、更多的部署選項，並有助於優化你現有的基礎架構、安全性和合規性。

Cloud: 云服务

常见的软件：

- In ye olde days all software was installed locally on our computers
- Even software that used servers on our network (i.e. for storage) would generally have so me locally-installed software on the workstations as well
- The advent of the Internet (and more specifically web technologies) turned this on its head

- SaaS - the provider hosts and manages the software 托管和管理软件, users connect to it remotely 远程
- No local installs, no local setup, no local infrastructure... 本地安装 本地设置 本地基础设施都不需要
- Usually this is web-based and over the Internet (though it might be a private IP network as well)
- All the user needs is a general-purpose browser and network link

The majority of Internet offerings are SaaS

- Google Search is SaaS as is the web version of Office 365 谷歌搜索、网络版office365
- Early on all web services were SaaS but "proper software" was all still locally installed (ran natively on a system)
- Locally installed software is a pain to write, maintain, install, manage, etc etc etc - different OS, different requirements, etc etc etc

good

- universally accessible from any platform
- no need to commute, you can work from anyplace
- excellent for collaborative working
- vendor provides modest software tools
- allows for muti-tenancy
- No local install
- Instant updates 不用担心数据丢失
- Access from anywhere
- Subscription model versus licencing (cheaper up-front investment)
- No customer infrastructure

bed

- portability & brower issues
- internet performance may dictate overall performance
- compliance restrictions

- Provides a platform to run and manage applications 提供一个平台来运行和管理应用程序
- The service provided is the infrastructure right up to the application (servers, network, operating system, database engine, ...) 服务器、网络、操作系统、数据引擎...
- We write the app, they run the app
- PaaS clouds can be public, private, or hybrid depending on access needs

PaaS Examples

- Google App Engine
- Heroku
- Microsoft Azure
- Amazon AWS

PaaS but local (on prem) - very private PaaS

- It's possible to leverage the power and simplicity of the technology (write code, throw at stack, code runs!) but locally 这个过程和远程连接进行unix的shell操作是一样的。
- Various "PaaS stacks" are coming online to deliver "truly private PaaS" (run on a local network only) but using the same tech

good

- cost effective repid development(it's scalable)
- faster, market for developers
- easy development of web applications
- private or public deployment is possible

bad

- developers are limited to the providers' language & tools
- migration issues - such as the risk of vendor lock-in

SaaS and Salesforce

- One of the early big players in SaaS (and who helped popularise the term) was Salesforce.com
- Founded in 1999 SF offered customer relationship management (CRM) software to its customers as an online only service 在线服务 提供客户关系管理(CRM)软件作为仅在线服务向客户提供
- In addition to being web-based it still offered high levels of customisation and configuration (and could be changed to customer needs) which was new for web-based products 提供了高水平的定制和配置（可以根据客户的需求进行更改）

Cloud: 云服务

/

相关的应用：

- The servers and storage and all the bits required for them including network links
- We manage the servers, they buy, connect, and run them
- This is incredibly commonly used by small(er) Internet firms who don't want to have to buy expensive hardware and run expensive data centres
- Provided servers can by physical or virtual
- We install and maintain the OS (or we may pay for services to do some or all of that for us) 与用服务器一一对应完全是IAAS，租用物理服务器以及来自数据中心某人的连接
- They own the hardware and are responsible for keeping it fed (power, AC) and connected (network)

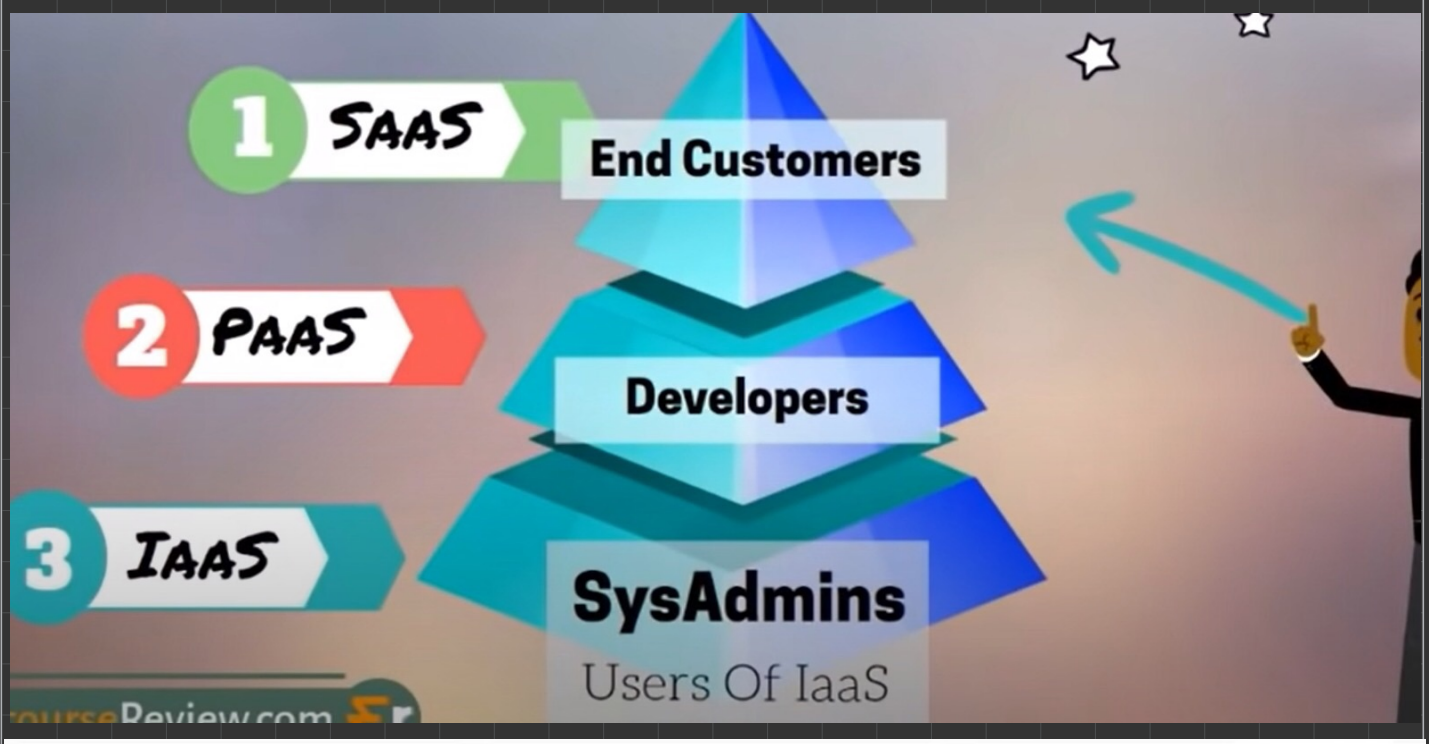
提供的服务器可以是虚拟的也可以是物理的

good

- The cloud provide the infrastructure
- enhanced scalability— dynamic workloads are supported
- Iaas is flexible

bad

- security issues
- network & service delays



Cloud 云服务

example stack=

further= DaaS-Desktop as Service

- DaaS providers offer a customised complete remote desktop solution with everything installed... workers just need a thin client (and a phone will do!)
- Zero infrastructure beyond some keyboards, displays and monitors

Gaming as a Service= GaaS

- Google Stavia
- GeForce Now
- ... Previous examples such as OnLive didn't end well
- Thin-client based technology with processing done "in the cloud"

good	bad
• No expensive and delicate infrastructure to buy and maintain (cheaper up-front and no less specialist skills required)无需购买和维护昂贵而精密的基础设施(前期成本更低，并且所需的专业技能也不少) • Concentrate on "what we do" - don't just be an IT company • Scalable (often instant and/or automatic [elastic]) 可拓展(通常是即时/自动[弹性]) • Let the hosting experts do the hosting • Fast/instant provision of new resources paid by use/time快速/即时提供按使用/时间付费的新资源	• Lack of control over our infrastructure - who's basket are we putting all our eggs in? Can we trust X with our critical processes?

Mitigation

- Service level agreements with strict penalties
- Independent certification of processes including security (have they gained a government seal of approval for example?)
- Many clouds make light work - spread the risk between providers

cloud 云服务

/

其实在之前就有类似的概念，那时候是因为人们无法家家户户都买的起大型电脑，所以在小型电脑问世之前，人们都是使用远程连接的方式来使用电脑的，但是后面小型电脑的产生，家家户户都拥有了电脑，而现在则是我们需要随时可以登录和使用我们的数据并且不需要携带电脑，通过任何一台计算机可以访问数据和程序可以更加方便办公

于是就衍出了云计算等一切云相关的应用

Future:

- The future is bright... after some teething problems
- Some organisations may well want to jealously guard their data 保护他们的数据
- But many actually see cloud offerings (from the big players) as more secure and better run (but this might just be risk transference!)
- Platform hosting such as Heroku (and big companies) have shaken up the world of provision - write it, send it, job done!
- People expect to be able to access their data (and their services) from any device and location 人们希望能从任何设备和地点访问他们的数据
- Ubiquitous computing means ubiquitous services and accessibility
- In the same way as in the 1960s it was unthinkable a small business would have a computer to run it's books... will this be the same in the 2030s (we all just have a phone/screen and the compute power and data is hosted "in the cloud")

Summary

- Cloud computing ideas aren't that new - let's put our processing and data into central storage run by someone else 让我们把我们的处理和数据放到由他人管理的中央存储中
- Newer developments though have moved towards different levels of cloud provision (IaaS to SaaS) 云被认为是降低了IT风险 不再有昂贵的机器是我们的责任
- Cloud was seen as de-risking IT (no more expensive machines that are our responsibility) but it introduces a whole new world of risk (who has our data, who manages our services, etc...) 但是引入了全新的风险：我们的数据将在别人的手中

Deployment Models:

- private cloud: provisioned for exclusive use by a single organisation
  提供给单一组织或家庭使用
- public cloud: provisioned for open use by the general public(i.e. use by general customers paying for access, sharing resources)
  提供给广大公众使用（即由普通客户付费访问，共享资源）
- community cloud: provisioned for use by a specified community/group of users(blurry but referenced in literature)
  提供给特定的社区/用户 群体使用（模糊不清，但在文献中提到过）
- Hybrid cloud: a mixture of two or more distinct cloud infrastructures
  两个或者多个不同的云基础设施的混合体

我们不需要一些维护机器的人员 网络维护之类的都不需要

我们只需要专注于自己想做的事情就好，而不需要去考虑别的东西，更不需要对其进行维护花费

但是数据容易遭到黑客攻击

独立认证之类的方式可以保证安全性，就像是我们用microsoft的email的时候如果我们使用新的设备或者在新的地址登录的时候会需要使用手机号码之类的进行安全验证

这样子可以提高安全性

pro=优点

- No expensive and delicate infrastructure to buy and maintain (cheaper up-front and no less specialist skills required)没有昂贵而微妙的服基础设施需要购买和维护 前期费用较低
- Concentrate on "what we do" - don't just be an IT company
- Scalable (often instant and/or automatic [elastic]) 可拓展（通常是即时和/或自动「弹性」）
- Let the hosting experts do the hosting 让托管和专家做托管
- Fast/instant provision of new resources paid by use/time 快速/即时提供新资源，按使用/时间收费

cons= 缺点

- It's just someone else's computer!
- Lack of control over our infrastructure - who's basket are we putting all our eggs in? Can we trust X with our critical processes?
- Where's our data?
- Is it protected?

mitigation= 缓解措施

- Know your partner (who do we trust?) 所有的东西都是存在于别人手中会存在风险
- Service level agreements with strict penalties
- Independent certification of processes including security (have they gained a government seal of approval for example?)
- Many clouds make light work - spread the risk between providers

Type of Cloud Computing "Stuff"

·Architecture - layers, design, redundancy, failover, ... 构架 层 设计 冗余 故障转移

·Technology - networking(including SDNs), virtualization, load balancing, proxies, containers, orchestration, ... 网络 虚拟化 负载平衡 代理 容器 协调

·Paradigms - serverless, ephemeral, ... 范例 无服务器 短暂性

Cloud computing

/

云计算通过互联网按需提供IT资源，并且采用按使用量付费的定价方式。您可以根据需要从诸如 Amazon Web Services (AWS) 之类的云提供商那里获得技术服务，例如计算能力、存储和数据库，而无需购买、拥有和维护物理数据中心及服务器。

雲計算是一種在線訪問信息和應用程序的方式，而無需在您自己的硬盤驅動器或服務器上構建、管理和維護它們。它快速、高效且安全。

云计算的使用

你现在可能正在使用云计算，即使你没有意识到这一点。如果您使用在线服务发送电子邮件、编辑文档、观看电影或电视、听音乐、玩游戏或存储图片和其他文件，云计算可能会使这一切在幕后成为可能。第一批云计算服务才有十年的历史，但出于各种原因，从小型初创企业到全球公司、政府机构到非营利组织，已经采用了这项技术。

以下是当今云提供商云服务可能实现的几个例子：

创建云原生应用程序

快速构建、部署和扩展应用程序——Web、移动和API。利用云原生技术和方法，如容器、Kubernetes、微服务架构、API驱动通信和DevOps。

测试和构建应用程序

通过使用可以轻松扩展或缩小的云基础设施，减少应用程序开发成本和时间。

存储、备份和恢复数据

通过互联网将数据传输到可从任何位置和任何设备访问的场外云存储系统，以更具成本效益和大规模的方式保护您的数据。

分析数据

在云端统一您的跨团队、部门和位置的数据。然后使用云服务，如机器学习和人工智能，发现洞察力，做出更明智的决策。

流式传输音频和视频

随时随地在任何具有高清视频和全球发行音频的设备上与您的受众联系。

嵌入式智能

使用智能模型来帮助吸引客户，并从捕获的数据中提供有价值的见解。

按需交付软件

按需软件也称为软件即服务（SaaS），可让您随时随地为客户提供最新的软件版本和更新。

Reference from IBM
Reference from ZDnet
Reference from Wikipedia

Domain Name System: 域名系统

The Domain Name System (DNS) is a critical piece of Internet infrastructure. It most "famously" is the service that turns names into numbers (hostnames to IP addresses) which then allow the right server to be connected to.

关键部分

Hostnames:

- Easier to remember and logical
- De-coupled from an actual interface so...
- Can easily move which IP address (interface) they point at

脱钩

- Isn't a "linear" (1:1) map...
  - One hostname can point to multiple IP addresses
  - One host (with one IP address) can can have several hostnames

Hostnames and IPs

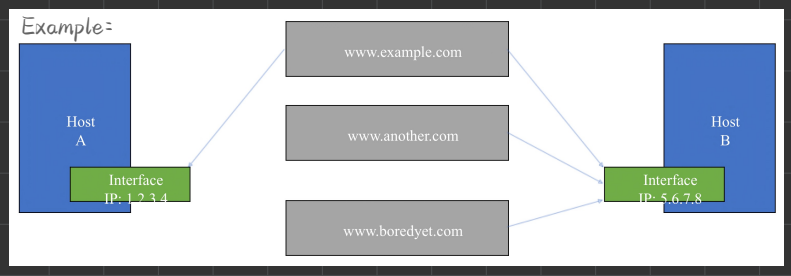
- Connected IP interfaces have an IP address
- Traditionally these are IPv4 addresses, 32 bits of data, represented commonly by the "dotted quad" format i.e. 143.117.52.246

点四

- However IP addresses are a pain to remember (good for computers, not so good for humans!)
- They also are an address tied directly to the interface
- What we would like to do is provide names which can be mapped to IP addresses, these are called hostnames
- We then commonly use these to connect to systems as part of a URL (Uniform Resource Locators)

(上面这个显然更易于人类记忆)

- http://www.qub.ac.uk/ (using a hostname)
- http://143.117.1.58 (using an IP address)



Hostname	IP Address
ecs135551.eng.qub.ac.uk	143.117.52.246
www.qub.ac.uk	143.117.16.63

Resolving Names 解析名称

- In order to locate a host (turn the hostname into an IP address) we need to find the IP address for that hostname
- This process is called resolving the hostname
- The software the resolves a name is called a resolver 解析器

Domain Name System: 域名系统

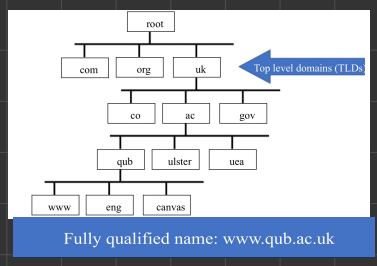
Distributed DNS 分布式域名系统:

The problem with HOSTS.TXT

- Does not scale 不具备规模
- Even if we replaced with a database whoever held that database would be flooded with requests (both resolutions and changes) 请求（决议和更改之类的）太多
- Highly centralised (高度集中) and prone to error (易出错) (single point of failure (单点故障) = bad)
- Solution: a distributed hierarchical system 分布式的分层系统
- Distributed = no single points of failure, load distributed, quicker lookups
  分布式 = 没有单点故障，负荷分散，查询速度快
- Hierarchical = changes made at the scope they effect, local control
  分层式 = 在其影响的范围内进行更改，本地控制

Hierarchical DNS 分层的DNS

- The namespace (命名空间) is divided into zones (区域)
- These form a hierarchy (层次结构) (tree)



Distributed Hierarchical DNS 分布式分层DNS

- Each zone (domain or sub-domain) is responsible for organising the names within that zone
- Each zone has at least two domain name servers
- The zone provides information about that zone including IP address(es) for each host within the zone (and other information) as well as information about sub-zones (such as which server has zone information about them)
- This gives local control and distributed load (分布式负载)

这里的13是因为上面的图中的13个服务器

The Root of the Tree (Root Servers) 根服务器

- At the top of the hierarchy is the root - this consists of 13 root servers located worldwide managed by 12 large Internet organisations 上图的13个服务器中除去qub这个
- Note this is actually 13 URLs but many more than 13 sites and actual servers!
- a.root-servers.net to m.root-servers.net
- Each leads to a network of servers that may be geographically diverse using a types of addressing called anycast (any valid server can answer)
  每一个都通向一个服务器网络，这些服务器可能在地理上是不同的，使用一种称为人意广播的寻址类型（任何有效的服务器都可以回答）

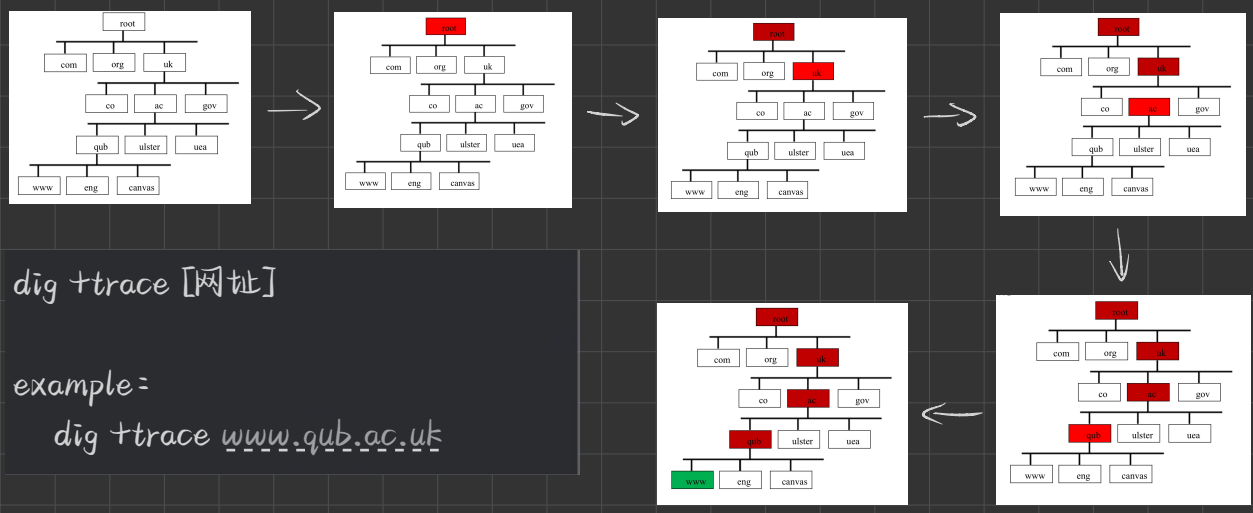
Root Servers

Domain Name System: 域名系统

/

Resolving an Address 解析一个地址

- We start at the top of the tree (root) 从顶端开始
- We ask for our address (i.e. www.qub.ac.uk) 询问地址
- Each level may not know the answer... but it knows the next level down so tells us their address(es) 每一层都可能不知道答案 但他知道下一层的地址
- Eventually we ask the right person (the authoritative) server and get our answer
  最终我们问正确的服务器询问得到答案




```
dig +trace [网址]

example:
   dig +trace www.qub.ac.uk
```

Caching 缓存

- Of course it's quite inefficient if we have to look up the address every time we need a resource (even one web page load can be 100s...)
  每次需要资源的时候都要查找地址，那是相当低效的
- The resolver and any intermediate servers will use caching to remember addresses (and intermediate answers!) so rarely have to actually go to the root server
  解析器和任何中间服务器将使用缓存来记忆地址（和中间答案） 所以很少需要实际去找根服务器
- For example; we looked up www.qub.ac.uk from root， if we looked up www.uea.ac.uk our cache would let us start at ac.uk not the root
- Of course this can be an issue - say we change the physical location of our website， but everyone has the old address cached? 改变了网站的物理地址 但每个人都有旧地址的缓存
- We use time-to-live (TTL) records， here 28800 seconds (8 hours)
- The cache can safely use this for 8 hours until it's "stale" and needs refreshed
  生存时间
  缓存可以安全的使用8小时，知道它改变需要刷新
- When choosing a TTL it's a trade off...
- Low TTL
  - A lot of queries to name server
  - Can slow down loading (as more resolutions)
  - Quick to update changes
- High TTL
  - Less queries， quicker loading
  - Data more likely to be out of date
- A common approach is to lower the TTL prior to changes 常见方法是在变化之前降低TTL
- i.e. it's 8 hours for "normal" operations but >8 hours before we're going to make a change we lower it to 5 minutes， make change， then set it back
- Warning: Some ISPs may not adhere to the TTL records to reduce load on their servers
- Cache value can be from 1 second to 68 years 缓存值可以从1s到68年

Domain Name System= 域名系统

/

实际上客户端只起到了询问作用剩下的并不是它工作得来的

Authoritative Server 权威服务器

- A server that is responsible for a domain (zone) and holds an answer is called authoritative 一个负责一个域并持有答案的服务器
- A cache isn't authoritative but repeats what it has been told by an authoritative server 缓存不是权威的，而是重复他从权威服务器哪里得到的信息

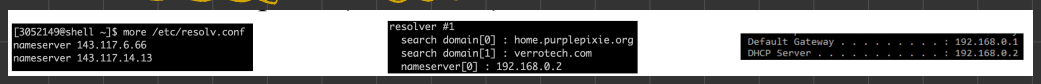
DNS Resolution

- We've seen how the process works
- DNS resolver works through cache or iterative query (one level at a time) to get the address DNS解析器通过缓存或迭代查询（一次一个级别）来获取地址
- However client computers do not actually do this (resolve an address themselves), rather the network they are connected to assigns one (or more) DNS servers for them to use
  然而，客户端计算机实际上并不会这样做（自己解决一个地址），而他们所连接的网络分配一个（或多个）DNS服务器供他们使用 → 所以是使用别人的东西
- These are the actual resolvers - the client asks them for an address and they go off and get the answer 这些是实际的解析器 -- 客户端问他们询问地址，他们就去获得答案

DHCP
DNS地址 — 路由器的地址 / ISP的DNS服务器

SoHo DNS Resolution

- In a small office/home office (SoHo) network usually the DNS address given to the clients (normally via DHCP) is either that of the router or the ISP DNS servers.
  小型办公/家庭办公
  通过DHCP提供给客户的DNS地址是路由器的地址或者ISP的DNS服务器
- If it's the router then the router may actually do resolution or (more likely) it in turn just forwards the request to the ISP DNS servers.
  如果是路由器，那么路由器可能真的会进行解析，或者（更有可能）它只是转发，（更有可能）它只是将请求转发给ISP的DNS服务器
- The local computer (and server) will cache the results. 本地计算机（和服务器）将对结果进行缓存



只对自己负责的区域更行查询答案的操作

Recursive Resolution 递归解析

- These servers, which offer to resolve any address for a client (they go off and find the answer if they don't know it) are known as recursive resolvers
  提供为客户解决任何地址的服务器（如果他呢不知道答案他们就去找答案）
- Most DNS servers will tell you the answers they know (the zones they are responsible for) but no more - they're not going to lookup other DNS for you
  大多数DNS服务器会告诉你他们知道的答案（他们负责的区域） 当没有更多-他们不会为你查询其他的DNS
- Your "friendly local DNS server" i.e. ISP or local network server will have a setting which allows recursive lookups from a given range (your network for example) and there are public services such as Google Public DNS which also offer this
  你的"友好的本地DNS服务器"，即ISP或本地网络服务器将有一个设置，允许从一个给定的范围（例如你的网络）进行递归查询，还有一些公共服务，如谷歌公共DNS，也提供这种服务.

Domain Name System: 域名系统

/

DNS Data

- DNS isn't "just" about hostnames
- It stores (and provides) other critical information about the zone that it is serving
  它储存（并提供）有关该区域的其他关键信息
- Over the years this has grown and now can include pertinent security information used for domain verification 现在可以包括相关的安全信息用于域名验证的信息

Key DNS Record Types

- A Hostname = IP Address (IPv4)
- AAAA Hostname = IP Address (IPv6)
- MX Mail Server 邮箱服务器
- NS Name Server 名称服务器
- CNAME Canonical Name (an alias) i.e. map to another record
- TXT Text Information (Generic Info)
- SOA Start of Authority (information about zone)
- PTR IP Address = Hostname (reverse lookups) 反向查询

en.wikipedia.ord Types

- SRV Service Record (specific port for service) 服务记录（服务的具体端口）
- DNSKEY Public key used to verify DNSSEC (DNS security) 用于验证DNSSEC的公钥
- RRSIG Resource record signature (used with DNSSEC) 资源记录签名 与DNSSES一起使用
- And there are MANY more...

Domain Name System= 域名系统

/

Domain Name System= 域名系统

/

Domain Name System: 域名系统

/

因为DNS发出请求然后得到结果解析出我的要的结果。但是如果请求被发送到恶意网站，那得到的IP地址也是恶意网站的这时候我们使用它访问的就会是这个恶意网站

DNS Security: (DNSSEC)

- As we've seen DNS is pretty important - in fact it's a critical piece of infrastructure
- The design of the hierarchical distributed network removes a lot of issues (similarly to the design of the Internet itself - much redundancy built in)
  分层分布式网络 冗余内建
- But... it's still vulnerable. This is especially true at the local level where DNS caches can be poisoned or man-in-the-middle attacks performed
  它仍然是脆弱的 在本地层面尤其脆弱 缓存 毒害 中间人攻击 DNS缓存可以被毒害或中间人攻击

Why do we need DNSSEC?

- Remember that DNS tells you where to actually send your requests?
- This means if a malicious actor can modify/control the results you get... they can send your requests anywhere they want
  恶意行为者 如果一个恶意的行为者可以修改/控制你得到的结果...他们可以把你的请求发送到任何他们想要的地方
- Commonly this is achieved via DNS cache poisoning (malicious control of a middle resolver, giving wrong results) or on a local network via MITM through a rogue DNS server
  通常是通过DNS缓存中毒（恶意控制中间解析器，提供错误的结果）或在本地网络上通过流氓DNS服务器的MITM实现的
- This means when we ask for www.mybank.com we get an IP address on their system and will then login etc to a malicious site
  我们会得到他们系统上的一个IP地址，然后会登录到一个恶意网站等

What does DNSSEC do?

- A lot of clever things but primarily it cryptographically signs responses
  加密方式签署了回复
- The resolver can then validate these signatures using known authorities and confirm (1) the remote server is valid and (2) the results haven't been interfered with in transit/the cache
  解析器可以使用已知的权威机构验证这些签名，并确认并确认(1)远程服务器是有效的，(2)结果没有在传输中/缓存中被干扰。结果没有在传输/缓存中被干扰
- Here we see the DS (delegation signer) for the UK TLD as well as the RRSIG (Resource Record Signature) which has specifically signed this request from their server and can be used to validate it
  这里我们看到了英国顶级域名的DS（委托签名者）以及 RRSIG（资源记录签名），它特别签署了来自其服务器的这一请求，并可用于验证该请求。

DNS and Privacy

- DNS is a potential privacy problem
  潜在的隐私问题
- As we need it to visit any host-based URL (so... basically all of them) we need to do a DNS lookup which can be logged
  由于我们需要它来访问任何基于主机的URL（所以.....基本上所有的URL），我们需要做一个可以被记录的DNS查询。
- Though not able to determine a full URL the DNS request will allow whoever runs our DNS resolver server (our ISP, Google DNS, ...) to see what addresses we are looking up/visiting
  虽然不能确定一个完整的URL，但DNS请求将允许运行我们的DNS解析器服务器的人（我们的ISP，谷歌DNS，...）看到我们正在寻找/访问的地址
- Now this may be fine... or it may not. It's a privacy concern but also more of an issue if a government requests these logs or has access perhaps?
  现在，这可能是好的.....也可能不是。这是一个隐私问题，但也是一个问题，如果政府要求这些日志或有访问权限，也许?

在某种情况下我们可以认为我的无法完全作到保密

Domain Name System: 域名系统

/

许多VPN只对某些流量进行隧道化处理（例如网络），因此DNS请求仍然在本地进行，可能在浏览器扩展中运行（因此又只有浏览器流量），或者可能排除本地地址（你的DNS服务器通常是这样的！）——这被称为DNS泄漏，并导致了一些问题。

VPNs and DNS Leaks

- Another solution may be to use a VPN - pass your traffic through it to a remote trusted source 另一个解决方案可能是使用VPN——将你的流量通过它传递到一个远程受信任的来源。
- Perfect solution in theory but when is anything simple?
- Many VPNs only tunnel certain traffic (say web) so DNS requests still go locally, may run in a browser extension (so only browser traffic again) or may exclude local addresses (which your DNS server often is!) - this is called DNS Leaking and has led to issues for some
- To avoid this we can tunnel all traffic at an OS level including DNS
  为了避免这种情况，我们可以在操作系统层面上对所有的流量进行隧道化处理，包括DNS请求，甚至在本地DNS服务器正常运行的情况下。

Control of DNS

- Although DNS is a highly distributed hierarchical system... there is still a controlling body
  尽管DNS是一个高度分布式的分层系统但仍有一个控制机构
- This is ICANN (the Internet Corporation for Assigned Names and Numbers) which manage the Internet Assigned Numbers Authority (IANA) 这就是ICANN（互联网名与数字地址分配机构）
- Until 1st October 2014 ICANN existed under the authority of the US Department of Commerce through the US National Telecommunications & Information Administration (NTIA) 在2014年10月1日之前，ICANN一直在美国商务部的授权下存在。商务部通过美国国家电信和信息管理局(NTIA)
- So now in theory the Internet is "free" and no one can turn it off (well at least DNS)
  因此，现在在理论上，互联网是"免费"的没有人可以把它关掉（好吧至少是DNS）
- However many still have issue with the widely US-centric nature of the root servers and this isn't helped by US corporations such as Google offering public DNS servers
- The reality is that were the US to legislate and require it's technology companies to comply a significant dent could be put into DNS 现实情况是，如果美国立法并要求它的技术迫使它的技术公司遵守，就会对DNS产生很大的影响。

DNS Protocol

- The DNS protocol is an early standard so it's entirely binary (none of this human-readable HTTP nonsense!) DNS协议是一个早期的标准，所以它完全是二进制的（没有这种人类可的HTTP废话）。
- Can operate in UDP (unreliable and size limited but QUICK) or TCP
  可以在UDP（不可靠，大小有限，但速度快）或TCP中运行

Query ID	Flags
Question Count (no of queries)	Answer Count (no of answers)
Authority Count (no of authoritative NS)	Additional record count (no of additional records)
Query and answer data	

At ac name server.
What is IP address of www.qub.ac.uk?
Don't know try juno.qub.ac.uk

What is IP address of juno.qub.ac.uk?

Try name server for qub.ac.uk

What is name server for qub.ac.uk?

juno.qub.ac.uk

Glue Records 胶水记录

- Sometimes we need to store additional information to help resolve (especially recursive resolution) 有时我们需要存储额外的信息来帮助解决（尤其是 递归解决）。
- These are called glue records (primarily where the DNS server for a domain is in the domain it's serving) 这些被称为胶水记录（主要是一个域的DNS服务器在哪里它所服务的域）

Domain Name System: 域名系统

/

Local Servers

- Of course in most use cases we want addresses that are globally resolvable i.e. someone anywhere (on another network) can resolve the address， which is done through the hierarchies we’ ve seen 当然，在大多数用例中，我们希望地址是全球可解析的，即任何地方的人（在另一个网络上）都可以解析该地址。这是通过我们所看到的层次结构完成的。
- But we can also run private internal DNS servers so we can resolve addresses like www.mytest.local etc 但我们也可以运行私有的内部DNS服务器，这样我们就可以解决像www.mytest.local 等地址。
- These will only work for people on our network who use our local DNS servers to resolve 这些仅对我们网络上的人有效，他们使用我们的本地DNS服务器来解析
- Used a lot in cloud services to build internal networks etc 在云服务中大量使用，以建立内部网络等

Enterprise Deployment= 企业部署

/

we start to consider some of the issues we face when trying to deploy a service live. These concepts include how to secure the system (allow the right users the right access), internal risks, attack surfaces, denial of service, content delivery networks, and more.

尝试实时部署服务时面临的一些问题
如何保護系統
內部風險
攻擊面
拒絕服務
內容交付網絡

Security安全问题

- Authenticatilon + Authorisation 认证+授权

  Authentication 验证
  - Who are you?
  - "We know you are user BOB because you entered the right password"
  - "We know you are APP X because you have the right API key"
  - "We know you are a QUB user because your IP address is in a range"

  Authorisation 授权
  - What can you do?
  - "You are allowed to save files to this location"
  - "You are allowed to print to this printer" → 工具的使用权
  - "You are not allowed to read files on this drive"
    数据读取的权限

- Accounts
  - Passwords
  - Keys
  - TFA
- API Keys
  - Google Maps etc
- Source
  - Domain
  - IP address

Enterprise Deployment: 企业部署

The Internet's a Dangerous Place

- In "the good old days" we could just stick a computer directly onto the Internet
- Public IP address
- Anyone could connect from anywhere
- Nowadays doing that is a recipe for near-instant calamitous death
- Internet service providers see endless probes trawling all the time for vulnerable services open

Vectors

Attack 攻击

- Attack Surface
  - The amount of yourself you expose to the outside world for attack 暴露在外部世界的攻击量
  - Wants to be as little as possible
  - For example use a firewall 防火墙 to block services we don't want exposed
  - Or we turn these services off
- Attack Vector 攻击媒介
  - Mode by which attack can be delivered
  - Internet direct connection is very common (and automated)
  - Also dodgy emails with attachments, rogue USB sticks, ...
    可疑 流氓

Attack Surface

- Anything that's accessible offers a surface for attack 可以访问的东西就意味着有可以攻击的接口
- Example MySQL database server...
  - Runs on TCP port 3306
  - Has its own access control (usernames and passwords)
  - However - relying on its own access control means it's still exposed to the world and vulnerabilities with it can be exploited
    - We only need to be able to connect to it from our local (web) server 漏洞
    - So we can use a firewall to simply stop all traffic 流量 on tcp/3306 from getting through

Attack Mitigations 缓解措施

- External:
  - Firewall
  - Air gap 气隙
  - Reduce surface
- Internal:
  - Anti-virus (anti-malware etc, pro-active scanning) 反病毒 (反恶意软件等, 主动扫描)
  - Device level blocking
  - Access Control
- Organisational:
  - Security controls (only access to who should have it)
  - Training (social engineering)

Enterprise Deployment: 企业部署

/

Successful Access Exploits: 成功的访问漏洞

Chelsea Manning 切尔西曼宁

- Contacted Wikileaks in Jan 2010 维基解密
- Jan 5 2010 downloaded 40，000 "Iraq War Logs" 伊拉克战争日志
- Jan 8 2010 downloaded 91，000 "Afghan War Logs" 阿富汗战争日志
- Saved to CD-RW and labelled the CD "Lady Gaga" to get through security
- Various other material inc. video

Edward Snowden 爱德华·斯诺登

- CIA employee and NSA- subcontractor
- Estimates: 15，000 Australian， 58，000 British， 50，000-200，000 NSA

(later US estimates 1.7 million documents)

- Leaks still ongoing
  泄漏

Security Failure: 安全故障

- Access to the information itself
  - Why did a Private in Iraq need access to diplomatic cables from Finland?
  - Overreaction to the 9/11 criticisms about a lack of information sharing?
    过度反应
- Ability to extract and export the information
  - Can save onto a CD-RW? Or other removable media?
- Ability to physically bring information out
  物理安全是否应该允许电子套件进出安全区域
  - Should physical security allow electronic kit in and out of secure areas?
- "Reliability test" (akin to Personnel Reliability Program in DoD)
  国防部
  - Manning had many concerns raised about suitability (though some of this could have been prejudice)
    担忧
  - Snowden raised concerns over the legality of collection

Enterprise Deployment: 企业部署

/

DDoS: 分散性阻断服务

Denial of Service (DoS) 拒绝服务

- Goal: we want to knock out a service
- Solution: flood that service with requests (perfectly valid requests probably)
- Result: service gets overloaded and either falls over or certainly slows down/ rejects genuine requests from real users
  跌倒 拒绝来自真实用户的真实请求
- Back in the day could flood a link from a single connection

Distributed Denial of Service (DDoS) 分布式拒绝服务

- A single connection...
  - Is easier to block
  - Is easier to deal with today at scale (your link capacity is much greater than broadband uplink capacity for example)
    规模化
- New threat is a Distributed Denial of Service (DDoS)
- Often seen used by Anonymous and lulzsec for example
  威胁

Is DDoS hacking?

- DDoS doesn't gain you access to anything
- But it can take a service completely offline (and cause longer term damage to reputation or equipment i.e. overloading hard drives) 超载硬盘.
  也可能使服务完全脱机 长期声誉或设备受损
- The entry requirements are very low so anyone can do it (that is not an encouragement)

Sources of DDoS
僵尸网络 商业指导. 赎金 / 威胁

- Botnets (often commercially directed for ransom/threat)
- Individuals (LOIC) in coordinated action 协调行动
- Maybe a tool like LOIC but also what about everyone just going to a website at 3pm together?
  个人
- Illegal and there have been prosecutions worldwide against DDoS organisers and perpetrators (with no sense of humour from the authorities)

DDoS Protection

- Adaptive firewalls 自适应的防火墙
- Ability to weather the storm 抵御风暴的能力
- Use of edge suppliers such as Content Delivery Networks (CDNs)
- Elastic computing in the cloud
  - Cost implications though
  - But... someone else's problem (but our reputation still!)

Enterprise Deployment：企业部署

Scaling Up 扩大规模

- We have a new service
- We rent a server to run it on
- Works great for dev and 1，000 users
- Next thing we're on the front page of WIRED... 100，000 new subscribers in a day... can the systems cope?

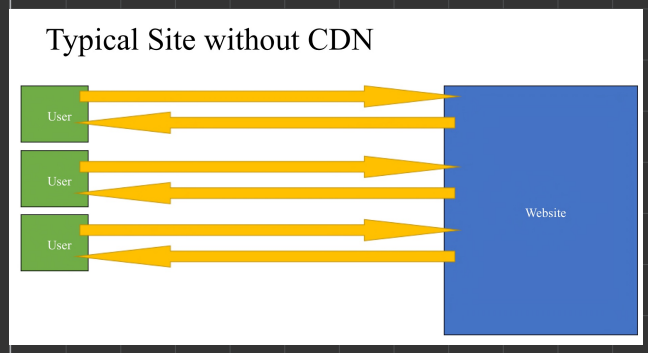
Fluctuating Load 负荷波动

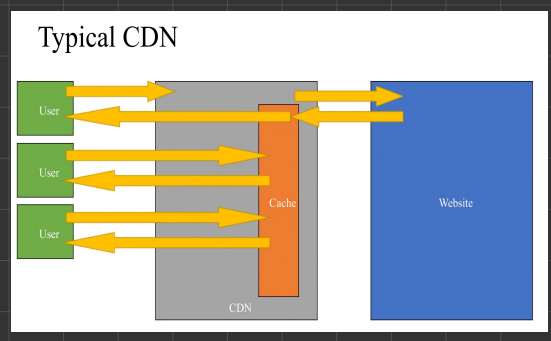
- Load fluctuation is a very real problem
- This might be in response to publicity of a review
- It might be cyclical (electrical load after Coronation Street finishes)
- It might be predictable... it might not be

Two Sides to the Problem

- Capacity
  - Ability of the servers/disks/network links to handle the load
  - How many connections can we take
- Ability
  - Can our service work on multiple servers?
  - Is it able to reconcile multiple different data streams
    协调 数据流

Content Delivery Networks (CDNs)内容交付网络





CDNs

- Allow widespread distribution of resources 允许资源的广泛分布
- High availability (uptime is their problem)
- Great for static resources (CSS， images， downloads， etc) - but not as efficient for dynamic content (stuff that changes every time)
  静态资源 动态资源
- May then offer a layer of protection but no speed up/load reduction 没有加速/减少负载

Enterprise Deployment: 企业部署

/

High Availability 高可用性

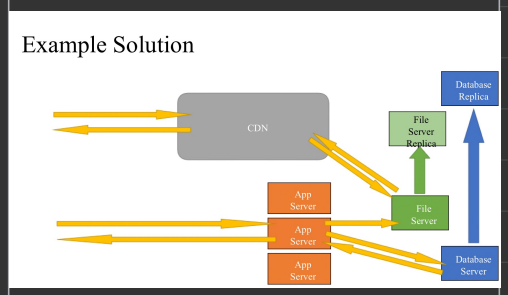
High Availability Services
- We want our service to be available
- We want to be able to cope with things like...
  - Increased volume of traffic 流量的增加
  - Failure
  - Saturation 饱和
  - Fire
  - Power Loss
  - Network Issues
  - ...

Techniques
- Virtualisation 虚拟化
- We can move servers around abstracted from hardware
- Cloud Computing (virtual or not)
- Containers and App Engines 容器和应用程序引擎
- Elastic Computing 弹性计算
- Read-only Replicas
- Hot-Swap / Warm-Swap Facilities 热交换 / 暖交换
- Multiple Data Centres
- ... many more!

Example.

Scenario
- We operate a web-based service: gooddoggo.com
- People visit this site to see pictures of good doggos
- People may also register and upload their own pictures of good doggos

Example Solution



Analysis
- 99%+ of requests are READ
- Most of our traffic (volume) is images of doggos we store
- Some uploading
- We need server scalability in case we get featured on dogs.com
  可拓展性的

Enterprise Deployment：企业部署

/

Replicas 复制品

- Can be synchronised (two-way); changes can be made to either and get copied to the other 同步
- Can be read-only (common); changes can only be made to one and the other can only be read from
- Note: read-only replicas can fail over to become the read-write masters as well
  只读副本可以故障转移成为读写主站

Summary

- The Internet is a dangerous place
- Reduce your attack surface and vulnerability to vectors 减少攻击面和易受攻击的载体
- Modern services can be both scalable and resilient but need careful planning
  可拓展的 弹性的

Enterprise Deployment: 企业部署

Virtualisation and Containers 虚拟化和容器

在过去我们需要考虑物理服务器（physical servers），除此之外每台服务器上都有操作系统（operating system）在系统上我们可以安装应用程序

Traditional Server Issues 传统的服务器问题

- Expensive，slow to provision（配置慢），demanding（要求高），heavy，…
- Single application very inefficient way of working
- Multi-tenancy (multiple applications on one box) created more headaches than it solved (applications like their own stacks)（租户）
- "Don't touch that" or "X doesn't play nicely with Y" culture

Virtualisation 虚拟化

- In effect an OS virtualises the hardware (provides a virtual interface to physical hardware) 实际上，操作系统将硬件虚拟化（为物理硬件提供一个虚拟接口）。
- What about virtualising the hardware before (at a lower level than) the OS so we could provide the same hardware to multiple operating systems? 在操作系统之前（比其更低的级别）对硬件进行虚拟化怎么样？所以我们可以为多个操作系统提供相同的硬件。系统提供相同的硬件呢？

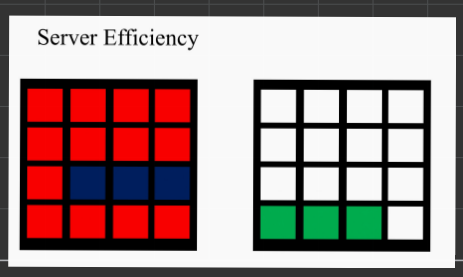
Step back: Emulation 仿真

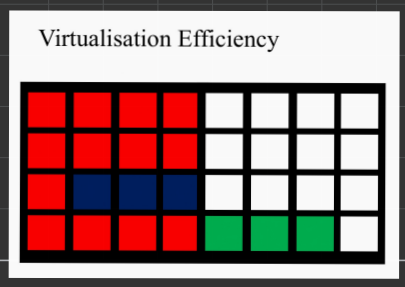
- Emulation is software pretending to be hardware 仿真是软件假装成硬件
- Computer resources are so quick now that a Raspberry Pi can "pretend" to be an old gaming console 现在的计算机资源是如此之快，以至于Raspberry Pi可以"假装"成一个旧的游戏机
- My desktop can "pretend" to be many many BBC Micro computers 我的台式机可以"假装"成许多BBC Micro电脑
- Software "emulates" the hardware (processor etc) and the guest software runs none the wiser; clever but not efficient (only really useful when we can't use the actual hardware)

软件"模拟"硬件（处理器等），而客户软件在运行的时候毫不知情；聪明但效率不高（只有在我们不能使用时才真正有用 只有在我们不能使用实际硬件时才真正有用）

Virtualisation

- Hardware virtualisation is the virtualisation of complete hardware platforms including certain logical abstractions of components 硬件虚拟化是完整硬件的虚拟化 平台的虚拟化，包括某些组件的逻辑抽象。
- Instead of providing actual hardware to an operating system (or other program… but let's not complicate matters now!) we provide "virtual hardware" (pretend hardware if you like) but this relates to actual components often (i.e. uses an actual CPU through an abstraction) 而不是向操作系统（或其他程序）提供实际的硬件。程序……但我们现在不要把问题复杂化！），我们提供"虚拟的 我们提供"虚拟硬件"（如果你愿意的话，假装的硬件），但这与实际的组件有关。组件（例如，通过抽象使用实际的CPU）。





History of Virtualisation

- 1960s mainframes were under-utilised 没有得到充分利用
- Not suitable for multiple simultaneous users (security and data protection)
- Virtualisation sought to "partition" the mainframe HW into distinct and apparently separate executions of the OS 虚拟化试图将大型机的硬件"分割"成不同的 的、明显独立的操作系统的执行
- Modern servers have the same challenges

Enterprise Deployment: 企业部署

Virtualisation and Containers:

Hypervisors: 管理程序

How does Virtualisation work? 如何工作 安装一个中间件——虚拟机监控器或"管理程序"。

- A piece of middleware – the virtual machine monitor or "hypervisor" is installed
- This virtualises the hardware and can provide it to multiple operating systems handling the timesharing of resources 这使硬件虚拟化，并可以提供给多个操作系统，处理资源的分时共享问题
- Each OS thinks it's installed and running on it's own server

每个操作系统都认为它是安装在自己的服务器上并运行的。

Types of Hypervisor

- Type 1 – runs on the physical hardware (no underlying operating system) with direct access to hardware 在物理硬件上运行（没有底层操作系统），可直接访问硬件
  - Common in enterprise
  - Most efficient when we don't want the machine to do anything else
  - Examples: VMWare ESX, KVM
- Type 2 – runs on top of an existing operating system (hosted) 在现有操作系统之上运行（托管）。
  - Great when you want to use guest OS "in addition" to main OS
  - Examples: VirtualBox, VMWare Player
- Hybrid – a mixed bag where there are some OS elements but the OS kernel directly supports virtualisation (so sort of type 1) 一种混合型，其中有一些操作系统元素，但操作系统内核直接支持虚拟化（所以有点像类型1）
  - Examples: Xen, Hyper-V

Advantages

- Make best use of available hardware (many individual virtual machines on one physical server) – can move them around between physical servers for best use. Can also set resource usage limits.
- Not the same as running all services on one operating system – logically separate and secure from each other (mostly!)
- Each service (software) can have it's own exact requirements
- Provisioning a "new virtual server" is a software copy job not a physical task and can easily be done completely remotely

优势

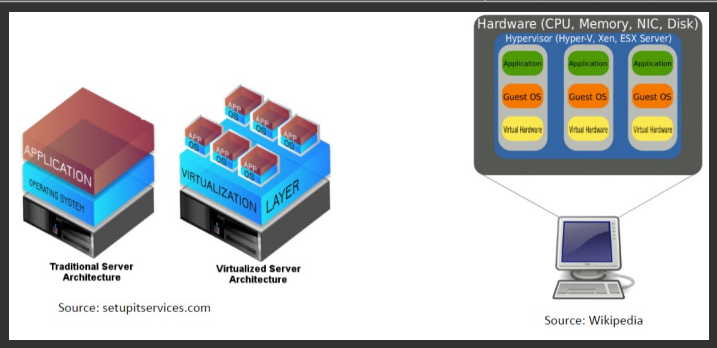
- 充分利用可用的硬件（一台物理服务器上有许多单独的虚拟机）- 可以在物理服务器之间移动它们以达到最佳使用效果。还可以设置资源使用限制。
- 与在一个操作系统上运行所有的服务不一样 - 在逻辑上相互分离和安全（主要是！）。
- 每个服务（软件）都可以有它自己的确切要求
- 提供一个"新的虚拟服务器"是一个软件复制工作，而不是一个物理任务，可以很容易地完全远程完成

Disadvantages

- Many eggs now in a single hardware basket
- Can be complex to understand
- Security through hypervisor vulnerabilities can be lax
- Shared loading needs careful consideration
- Temptation to just overload available capacity

缺点

- 现在许多鸡蛋都放在一个硬件篮子里
- 理解起来可能很复杂
- 通过管理程序漏洞实现的安全可能会很松散
- 共享负载需要仔细考虑
- 有可能使可用容量超载



Enterprise Deployment：企业部署

/

Virtualisation and Containers：

Routing and load balancing路由和负载平衡

The Virtualisation Revolution

- Virtualisation has revolutionised traditional server provisioning
- It's pretty much the default mode of operation now for servers
- But as mentioned it has it's disadvantages - mainly around size of the virtual machines (they're full blown OS stacks)

虚拟化革命

- 虚拟化已经彻底改变了传统的服务器配置方式
- 它几乎是现在服务器的默认操作模式
- 但如前所述，它也有缺点——主要是围绕虚拟机的大小（它们是完整的操作系统栈）。

Enterprise Deployment: 企业部署

Virtualisation and Containers:

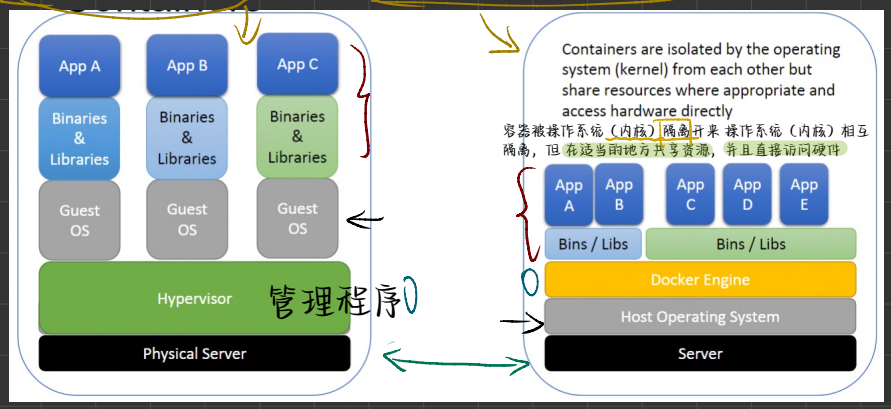
Containers: 容器

- The "new hotness" led in large part by Docker adoption
- Virtual servers are great but... they are a full operating system stack installed with all the virtualised IO hardware necessary
- Often we just want to run an application on the bare minimum but want it contained within it's own environment - containers are "operating-system-level virtualisation"
- Containers try and be a "middle ground" - all the advantages or virtualisation without the disadvantages

容器

- 在很大程度上，Docker的采用引领了"新的热度"。
- 虚拟服务器很好，但是.....它们是一个完整的操作系统栈，安装了所有必要的虚拟IO硬件
- 通常，我们只想在最低限度内运行一个应用程序，但希望它包含在自己的环境中 - 容器是"操作系统级的虚拟化"
- 容器试图成为一个"中间地带"--具有虚拟化的所有优点，但没有缺点。

Virtual Machines versus Docker Containers



Install size comparison:

- "Clean" CentOS VM
  - ~1GB
- Docker
  - 71MB

经过的更少要的内存更少

Process count comparison:

"Clean" CentOS VM

- ps aux | wc -l
- 185

Docker CentOS Container

- ps aux | 完成-l
- 4

Why?

- Docker by default ships just the most minimal of OS in a container
- Uses the kernel etc of the host machine (partitioned for security)
- Everything else uses the host machine OS (I/O etc) - all the stuff that takes up loads of processes, RAM and disk space

为什么?

- Docker默认在容器中只提供最简单的操作系统
- 使用主机的内核等（为安全起见进行了分区）。
- 其他一切都使用主机操作系统（I/O等）--所有的东西都是占用了大量的进程、内存和磁盘空间

Why Care?

- Because containers are so small and flexible they can be easily deployed and moved around
- A typical full stack VM is many GB
- A typical full stack container (even once running with data!) is a few hundred MB (and the difference between base and running is small)
- Easily orchestrated (e.g. kubernetes)

为什么关心?

- 因为容器是如此之小和灵活，他们可以很容易地 部署和移动
- 一个典型的全堆栈虚拟机是许多GB
- 一个典型的全堆栈容器（即使是在有数据的情况下运行！）只有几 几百MB（而且基础和运行之间的差异很小）
- 容易协调（例如，kubernetes）。

Enterprise Deployment：企业部署

Virtualisation and Containers：

Container：容器

The Developer’s Environment

- Even on a VM we (the upstream dev) don’t have great control over the actual environment
- We could tell people what to install or we could even ship an entire VM image (like we use in labs)
- But this isn’t scaleable
- Containers let us build the exact image every single time

开发者的环境

- 即使在虚拟机上，我们（上游开发者）也不能很好地控制实际环境
- 我们可以告诉人们要安装什么，或者我们甚至可以发送一个完整的虚拟机镜像（就像我们在实验室里使用的那样）。
- 但这并不具有可扩展性
- 容器让我们每次都能建立精确的镜像

Enterprise Deployment: 企业部署

Virtualisation and Containers:

Container

Docker:

What is Docker?

- Other container environments exist but Docker is the most popular and largely led the revolution
- Docker Engine is the core component that runs containers
- Docker Compose and Docker Swarm offer coordination services
- Other container engines often support Dockerfile settings

什么是Docker?

- 存在其他容器环境，但Docker是最流行的。并在很大程度上引领了这场革命
- Docker引擎是运行容器的核心组件
- Docker Compose和Docker Swarm提供协调服务
- 其他容器引擎通常支持Dockerfile设置

Dockerfile

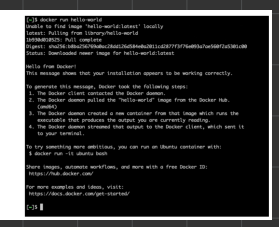
- The Dockerfile determines what actually goes into a container
- The key elements include:
- Base container (if any)，for example Linux CentOS &
- Network setup
- Storage setup
- Commands to run to build your environment (at setup)
- Commands to run to execute your service (at runtime)

- Dockerfile决定了容器中的实际内容。
- 关键元素包括
- 基础容器（如果有的话），例如Linux CentOS &
- 网络设置
- 存储设置
- 运行构建环境的命令（在设置时）。
- 运行命令来执行你的服务（在运行时）。

Example Simple Dockerfile

- FROM ubuntu:18.04 — Start from the Ubuntu 18.04 stock image
- COPY . /app — Copy everything from CWD (.) to /app on container
- RUN make /app — Build our app with make (during container build)
- CMD python /app/app.py — Run this command when we start our container

First Docker Container (hello-world)



First Docker Container (hello-world)

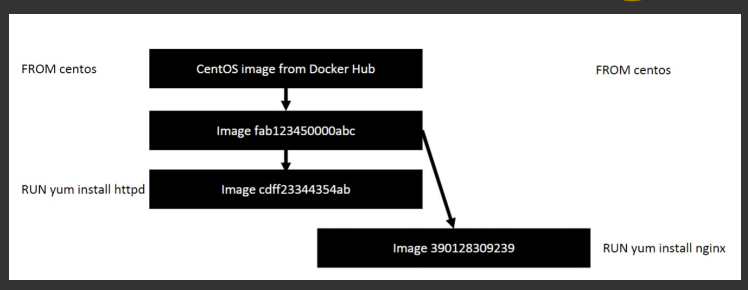
Take 2

Incremental Image Building

- Each time we change (RUN) our container during build it's just the increment that is stored
- The original image also remains (and so can be used again easily)

增量图像构建

- 在构建过程中，每次我们改变（RUN）我们的容器时，只是存储了增量。
- 原始镜像也会保留（因此可以很容易地再次使用）



Enterprise Deployment：企业部署

/

Enterprise Deployment：企业部署

/

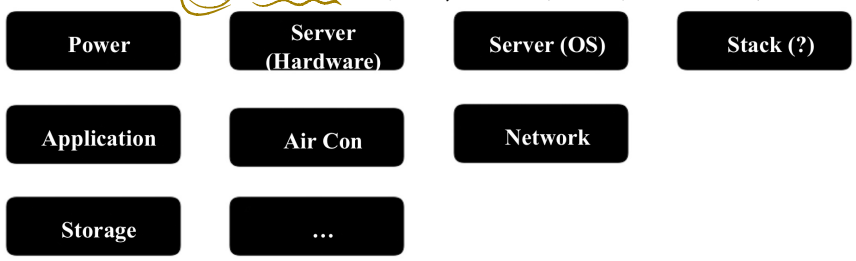
Enterprise Deployment：企业部署

/

Resilience and Measurement弹性和测量

Service Availability 服务可用性

- To keep our **service available to end users** what sort of things do we need to be working?
  为了保持我们的服务对终端用户的可用性，我们需要什么样的东西？需要工作？

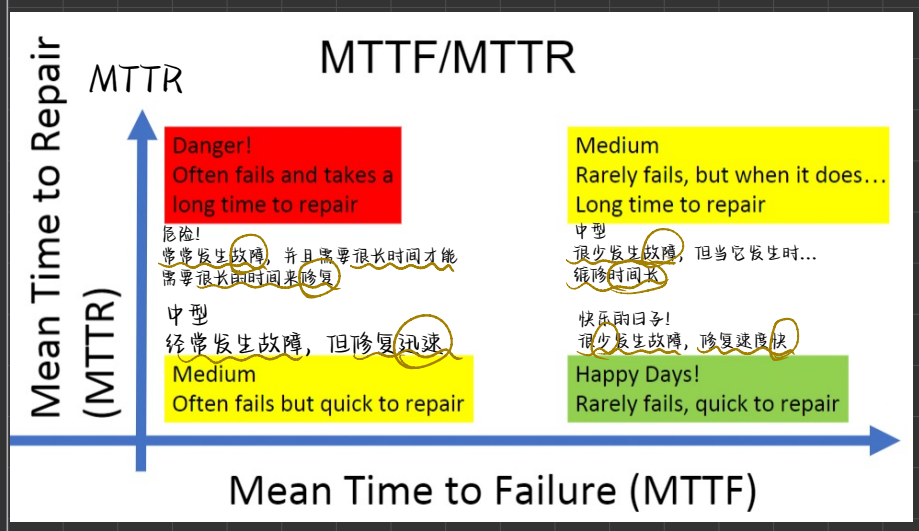


Some Terms

- SPOF – Single Point of Failure 单点故障
- MTTF – Mean Time to Failure 平均故障时间
- MTTR – Mean Time to Repair/Replace/Recover平均修复 / 替换 / 恢复时间
- Redundancy – duplication of critical components such that there are spares (can carry on if one fails) 冗余 – 关键部件的复制，这样就有了备用部件（如果一个部件发生故障，可以继续使用）。

Single Point of Failure (SPOF) 单一故障点(SPOF)

- Minimise these! – 尽量减少这些
- Redundant systems – 冗余系统
- Clever design / Fault tolerance – 巧妙设计/容错



Redundancy Terminology

- n is the number of things required to do a job
  - We need two servers, n=2
  - We need one air conditioner, n=1
  - We need three generators, n=3
- Redundancy is given in terms of n
  - n+1 – we have one more than required (warning, if we require 100 then n+1 isn't much slack)
  - 2n – we have twice the required (when n=1, n+1 = 2n also)

冗余术语

- n是完成一项工作所需的东西的数量
  - 我们需要两台服务器，n=2
  - 我们需要一台空调，n=1
  - 我们需要三台发电机，n=3
- 冗余度是以n为单位的
  - n+1 – 我们比需要的东西多一个（警告，如果我们需要100个，那么n+1并不是很宽松）
  - 2n – 我们有两倍的需求（当n=1时，n+1=2n）。

Resilience and Measurement弹性和测量

/

Redundancy Calculations

- We need one air conditioner (n=1) and we have two (n+1)
- The chances of one air conditioner failing in any given year is 1:20 (MTTF to 20 years; apologies to any actual statisticians in!)
- What is the chance of an air conditioning outage in any given year?
- A. 1:400 (i.e. 1:20x1:20) B. 1:20

冗余度的计算

- 我们需要一台空调（n=1），我们有两台（n+1）。
- 一台空调在任何一年出现故障的几率是1:20（MTTF到20年；向任何实际的统计学家表示歉意！）。
- 在任何给定的一年中，空调故障的机会是什么？

The Redundancy Fallacy、

- All things being equal and independent the calculation is simple=
  - Chances of 1 failing in a given period 1:x
  - Chances of 2 failing in the same period 2:x*x
- Actually a lot more complex than that
  - Firstly the question isn't "chances of a failure in a year" it's actually "chances of both failing at the same time" (within the repair window)
  - Also "chance of" doesn't mean won't etc - cars crash every day
- And factors are not independent at all
  - Same environmental factors
  - Same age of equipment
  - ...

冗长的谬误、

- 在所有事情都相同和独立的情况下，计算很简单。
  - 在一个给定的时期内，1个失败的几率是1:x
  - 在同一时期，2个失败的机会 2:x*x
- 实际上比这要复杂得多
  - 首先，问题不是"一年内失败的几率"，而是"两个同时失败的几率 两者同时发生故障的机会"（在维修窗口内）。
  - 另外，"几率"并不意味着不会等--汽车每天都在崩溃。
- 而且各种因素根本就不是独立的
  - 相同的环境因素
  - 相同的设备年龄
  - ...

Resilience and Measurement弹性和测量

/

NASA Thinking

- Space shuttle boosters are made of sections which slot together before launch
- To stop the hot gasses escaping from these gaps (not good!) they are sealed with rubber O-rings
- Since these are a critical component there are two (n+1 or 2n)
- Gasses are found to have escaped past one O-ring but never the second (so the redundancy worked，right?)

美国宇航局的思考

- 航天飞机的助推器是由一些部分组成的，这些部分在发射前会被卡住。
- 为了防止热气从这些缝隙中逸出（不好！），它们被用橡胶O型圈密封
- 由于这些是一个关键的部件，所以有两个（n+1或2n）。
- 气体被发现逃过了一个O型环，但从未逃过第二个O型环。（所以冗余是有效的，对吗？）

Resilience and Measurement弹性和测量

/

Measurement测量
我们需要有效地控制我们没有测量的东西

Why Measure?
- To control – 为了控制
- To predict – 为了预测
- To respond – 响应

What is measurement?
- Measurement is the process by which numbers or symbols are assigned to attributes of entities in such a way as to describe them according to clearly defined rules
- Entity – an object (person, room, server, event i.e. journey, ...)
- Attribute – feature we're interested in (height, length, weight, area, colour, time, ...)

什么是测量?
– 测量是一个过程，通过这个过程，数字或符号被分配给实体的属性，以便描述它们。赋予实体的属性，以便根据明确规定的规则来描述它们
– 实体--一个物体（人、房间、服务器、事件，即旅程，...）。
– 属性--我们感兴趣的特征（高度、长度、重量、面积。颜色，时间，...）

Resilience and Measurement弹性和测量

/

Metrics度量衡

- Measuring something against a formal standard
- Usually implies some sort of expected standard for comparison
- 用一个正式的标准来衡量某物
- 通常意味着有某种预期的标准进行比较

Key/Meaningful Metrics

- Nowadays we can store a lot of monitoring data
- Storage is cheap and recording is easy
- Having a clue how to make use of this data is a challenge (big data anyone?)
- We need to measure meaningful (aka key) metrics

关键/有意义的指标

- 如今，我们可以存储大量的监测数据
- 存储很便宜，记录很容易
- 如何利用这些数据是一个挑战。数据是一个挑战（大数据是什么？）
- 我们需要测量有意义的（又称 关键）指标

我们记录什么

- 日志(事件)
- 指标的测量
- 测试结果（指标与比较）。

我们为什么要记录

- 提醒（某些东西已经失败或有失败的趋势）。
- 预测
- 测量和验证/分析
- 追踪潜在的故障

Logs are still needed

- Logs are often text formatted and verbose in the most
- There are automated tools but generally we don’t use logs to monitor
- We have a metric/test which fails and alerts
- Then we use the logs in a targeted fashion to find the problem

日志仍然是需要的

- 日志通常是文本格式的，而且是最冗长的
- 有一些自动化的工具，但通常我们不使用日志来监控
- 我们有一个指标/测试，失败后会发出警报
- 然后，我们有针对性地使用日志来找到问题所在。

Internal and external metrics

- We’re very good at measuring internal attributes
- But are these really meaningful to our core demographic?
- Does a customer care about CPU load per-se? • Does a CPU load care about the customer?
- Internal attributes can be useful indicators but often we want to measure from an external perspective

内部和外部的衡量标准

- 我们非常善于衡量内部属性
- 但这些对我们的核心人群真的有意义吗？
- 客户关心CPU负载本身吗？
- CPU负载关心客户吗？
- 内部属性可能是有用的指标，但往往我们希望从外部角度来衡量

Resilience and Measurement弹性和测量

/

Internal and external metrics

Internal 内部

- Average server CPU load – 平均服务器CPU负载
- Highest server CPU load – 最高的服务器CPU负载
- Memory usage (%) – 内存使用率(%)
- Storage usage (%) – 存储器使用量(%)
- Power consumption – 电力消耗
- Running containers – 运行中的容器
- Demanded containers – 需求的容器
- Available containers – 可用容器

External 外部

- Web response time – 网络响应时间
- Transaction response time – 交易响应时间
- Connection error rate – 连接错误率
- Content validity testing – 内容有效性测试

Resilience and Measurement弹性和测量

/

Approaches to Metrics 衡量标准的方法

Goal Question Metric (GQM)目标问题衡量标准

- Goals -> Questions -> Actual Metrics
- "Fast response" -> "How fast is our response?" -> Measure Response
- "Accuracy" -> "How often is the answer right?" -> Check Answer
- Goals are actually often more complex

- 目标 -> 问题 -> 实际指标
- "快速反应" -> "我们的反应有多快？" -> 测量反应
- "准确性" -> "答案正确的频率是多少？" -> 检查答案
- 目标实际上往往更复杂

Goal Definition目标定义

Purpose:

对过程进行定性以便理解它

To (characterise) the (process) in order to (understand) it

evaluate *product* *assess*
predict *model* *manage*
motivate *metric* *improve*

Perspective (as who)

Environment (what's the setup)

Goal Example

To predict the product in order to improve the effectiveness from the viewpoint of the customer when run on a live system

- Pretty open ended， but hopefully you get the idea

目标示例

对产品进行预测，以便从客户的角度提高产品的有效性。从客户的角度出发，提高产品在实际系统中运行的有效性。

- 非常开放，但希望你能明白这个意思。

Quality Factors质量因素

Quality factors as applied to a product

Resilience and Measurement弹性和测量

/

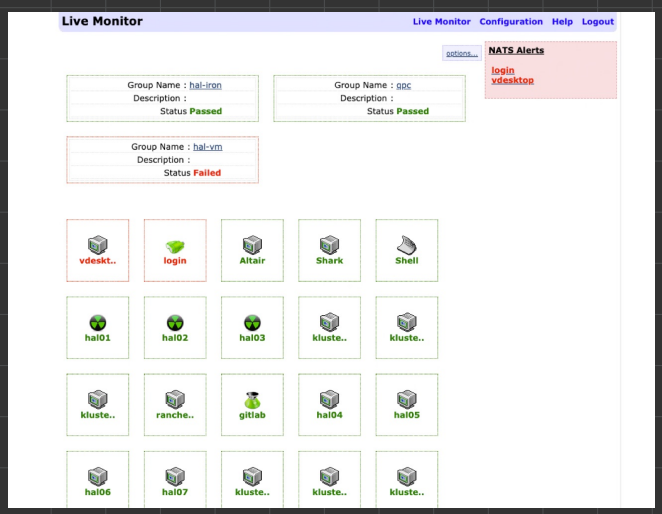
Monitoring监控

Monitoring Sources

- Monitoring tools (metrics)
- Alerts for failing tests
- Logs (useful for after-the-fact debugging)
- Remember: we ideally want measures from the bottom to the top (CPU node, CPU overall, app stack, network, ..., customer request)

监测来源

- 监测工具（度量）。
- 对失败的测试发出警报
- 日志（对事后调试有用）
- 记住：我们最好从下往上测量（CPU节点、CPU整体、应用堆栈、网络……、客户要求）



Video Service

Daveflix Global:

- What could we measure/record? — 我们可以测量/记录什么？
- What should we measure/record? — 我们应该测量/记录什么？
- What are our actual drivers (factors/criteria)? — 我们的实际驱动力是什么（因素/标准）？
- What should we alert on (key metrics)? — 我们应该警惕什么（关键指标）？

Resilience and Measurement弹性和测量

/

Setting the Bar

Setting our initial bar

- Sometimes this is easy - "how quickly does the page load? 0.5s is acceptable"
- Sometimes this is hard - "what is the CPU load? What is acceptable... ??"
- Sometimes we have to make an educated guess and go from there

设定我们的初始标准

- 有时这很简单--"页面的加载速度如何？0.5秒是 可以接受"
- 有时这很难--"CPU的负荷是多少？什么是可以接受的.....？??"
- 有时，我们必须做出有根据的猜测，然后从那里开始。

Resilience and Measurement弹性和测量

/

Estimation (Fermi Question)估计数

Fermi Question / Fermi Estimation

- Quick estimate of a quantity that seems difficult or impossible to determine precisely
- Use common sense and rough estimates to generate estimate
- Useful for establishing baselines as well as being fun

费米问题/费米估算

- 对一个似乎很难或不可能确定的数量的快速估计 准确地确定
- 使用常识和粗略估计来产生估计值
- 有助于建立基线，同时也很有趣