air.man.six@gmail.com

目录

图目录

表目录

第 I 部分

Gnu/Linux 基础知识篇

第 1 章　关于本书

GNU/Linux(简称 GNUnux) 是 GNU 计划的支持者与开发者，特别是其创立者 Richard Stallman[1]对于一个以 Linux 闻名的类 Unix 操作系统的称呼。

由林纳斯 · 托瓦兹及其他人士开发的 Linux 并不是一个完整的操作系统，而仅仅是一个类 Unix 内核。事实上，Linux 一开始是以完成 Minix 内核的功能为目标，Linus 想做一个”比 Minix 更好的 Minix “。而 GNU 计划始于 1984 年，终极目标是完成一套基于自由软件的完整作业操作系统。到 1991 年 Linux 的第一个版本公开发行时，GNU 计划已经完成除了操作系统内核之外的大部分软件，其中包括了一个壳程序（shell），C 语言程序库以及一个 C 语言编译器。林纳斯 · 托瓦兹及其他早期 Linux 开发人员加入了这些工具，而完成了 Linux 操作系统。但是尽管 Linux 是在 GNU 通用公共许可证下发行，它却不是 GNU 计划的一部分。

正是由于 Linux 使用了许多 GNU 程序，Richard Stallman 认为应该将该操作系统称为“GNU/Linux”比较恰当。[?]

1.1　Gnu 计划

GNU 计划，有译为“革奴计划”，是由理查德 · 斯托曼在 1983 年 9 月 27 日公开发起的，它的目标是创建一套完全自由的操作系统。理查德 · 斯

[1]理查德 · 马修 · 斯托曼，美国自由软件运动的精神领袖、GNU 计划以及自由软件基金会的创立者。作为一个著名的黑客，他的主要成就包括 Emacs 及后来的 GNU Emacs，GNU C 编译器及 GDB 调试器。他所写作的 GNU 通用公共许可证是世上最广为采用的自由软件许可证，为 copyleft 观念开拓出一条崭新的道路。

1990 年代中期，斯托曼作为一个政治运动者，为自由软件辩护，对抗软件专利及版权法的扩张。他对程式设计方面的投入都放在 GNU Emacs 上。他从演讲中获得的收入，已足够维持自己的生活。

他最大的影响是为自由软件运动竖立道德、政治及法律框架。他被许多人誉为当今自由软件的斗士、伟大的理想主义者。

托曼最早是在 net.unix-wizards 新闻组上公布该消息，并附带一份《GNU 宣言》等解释为何发起该计划的文章，其中一个理由就是要“重现当年软件界合作互助的团结精神”。

UNIX 是一种广泛使用的商业操作系统的名称。由于 GNU 将要实现 UNIX 系统的接口标准，因此 GNU 计划可以分别开发不同的操作系统。GNU 计划采用了部分当时已经可自由使用的软件，例如 TEX 排版系统和 X Window 视窗系统等。不过 GNU 计划也开发了大批其他的自由软件，这些软件也被移植到其他操作系统平台上，例如 Microsoft Windows、BSD 家族、Solaris 及 MacOS。

为保证 GNU 软件可以自由地“使用、复制、修改和发布”，所有 GNU 软件都包含一份在禁止其他人添加任何限制的情况下，授权所有权利给任何人的协议条款，GNU 通用公共许可证（GNU General Public License，GPL）。这个就是被称为‘公共版权’的概念。GNU 也针对不同场合，提供 GNU 宽通用公共许可证（与 GNU 自由文档许可证这两种协议条款 [?]。

1.1.1 图片欣赏

下面给出几张美图看看。它们分别是 Gnu 的 logo，内核 Linux 的 logo，还有我最喜欢的理查德的照片，最后一张是我在网上找到的，自己稍作了处理，嘻嘻……！

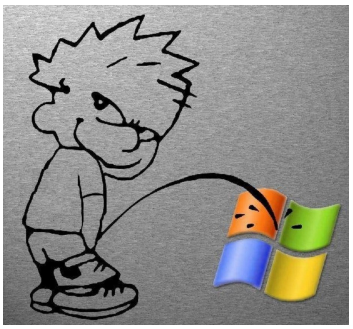
1.2 本书使用的操作系统

本书的所有操作是在作者笔记本上的虚拟机上演示，虚拟机操作系统为 RHEL5u8 32bit。

(a) Gnu Logo

(b) Linux Logo

(c) Richard



(d) 网友恶搞

图 1.1 美图欣赏

第 2 章　常用命令使用

2.1　命令行下的快捷键

经常在命令行下工作的同志们，可能用的最多的就是两个上下键，主要用来调出历史命令；使用左右箭头使光标向后或向前移动以修改上次使用过的命令。其实这样做效率并不是很高，有了快捷键可以让我们的效率有所提高，而且看起来还更专业、更加 Awesome、更加 Geek。掌握了这些快捷键，我们可以做到手不离主键盘区域，完全可以忽略掉键盘上的四个可爱的箭头。当我们熟练之后，会越发喜欢这种方式。

2.1.1　常用快捷键介绍

下面介绍一些作者在命令行下经常使用的快捷键，这些快捷键在 Emacs 下面是有同样的效果的，不信？你可以试试看。其实，Emacs 是 Gnu/Linux 系统下的命令行编辑器，通过/etc/profile 或/etc/bashrc 等文件都可以找到相关的设置。

1. Ctrl+A 快捷键

   这里的 A 可以理解为 Head。当我们按下此组合键时，光标就从当前位置移到了命令行的起始位置。别只顾着看，动手试试！

2. Ctrl+B 快捷键

   这里的 B 可以理解为 Backward，向后的意思。有时在命令行上，我们把某个命令的参数或路径写错了，一般的做法是，使用左箭头，使光标移动到指定的位置，然后修改。其实我们完全可以使用 Ctrl+B 的方式以达到同样的效果。别只顾着看，动手试试！

3. Ctrl+C 快捷键

   这个组合键是用来终止当前正在运行的前台进程。在 UNIX 环境高级编程一书上看到了一个用来终止当前运行进程的组合键，是 Ctrl+\[?]。别只顾着看，动手试试！

4. Ctrl+D 快捷键

   这个组合键的用途也很广，我主要用此组合键来退出某个程序，如 Python、MySQL 等等。在命令行下意思就不同啦，此时的 D 可以理解为 Delete。按下此组合键，会删除当前光标处的字符。别只顾着看，动手试试！

5. Ctrl+E 快捷键

   这里的 E 可以理解为 End。当在命令行按下此组合键时，我们的可爱的光标就乖乖地跑到了当前命令行的最后。

6. Ctrl+F 快捷键

   这里的 F 可以理解为 Forward，向前的意思，等同于按下右箭头。别只顾着看，动手试试！

7. Ctrl+H 快捷键

   此组合键相当于键盘上的 Backspace 键。按下此组合键，它会从当前光标处开始向后删除字符。别只顾着看，动手试试！

8. Ctrl+J 快捷键

   此组合键相当于键盘的回车键。按下此组合键，相当于按了一次回车键。在 Windows 的命令行下，Ctrl+M 好像是等同于回车键。别只顾着看，动手试试！

9. Ctrl+K 快捷键

   这里的 K 可以理解为 Kill。按下此组合键，会删除从当前光标到本命令行的结束的位置的所有字符。别只顾着看，动手试试！

10. Ctrl+L 快捷键

    这里的 L 可以理解为 Clear。按下此组合键相当于执行了 clear 这条命令，清除当前屏幕上的内容。别只顾着看，动手试试！

11. Ctrl+N 快捷键

这里的 N 可以理解为 Next。这个组合键的作用是用来调出下一条历史命令，与之对应的快捷键 Ctrl+P 是调出上一条历史命令。代替了向下的箭头。别只顾着看，动手试试！

12. Ctrl+P 快捷键

这里的 N 可以理解为 Previous。这个组合键的作用是用来调出上一条历史命令，与之对应的快捷键 Ctrl+N 是调出下一条历史命令。代替了向上的箭头。别只顾着看，动手试试！

13. Ctrl+R 快捷键

这个组合键是用来搜索之前的历史命令。这里的 R 可以理解为 Reverse，反向的意思。在 Emacs 里为向后搜索，与之对应的是 Ctrl+S 快捷键是向前搜索。不过 Ctrl+S 在命令行里却不是这个作用，而是用来锁屏的。别只顾着看，动手试试！

14. Ctrl+T 快捷键

此组合键是交换两个相邻字符的位置。交换的是当前光标处字符及其当前光标前面的字符。比如我们不小心把 clear 命令写成了 clera，此时我们也不用把 ra 两个字符删掉，然后再写上正确的。此时使我们的光标位于字符 a 上，让后按下此组合键，是不是神奇的事情发生了？当然，如果光标在行尾，按下此组合键，它会交换光标前的两个连续的字符。在 Emacs 下面，使用 Ctrl+X 与 Ctrl+T 两个组合键[1]，可以交换当前光标行与上一行的位置。别只顾着看，动手试试！

15. Ctrl+W 快捷键

此组合键在 Emacs 中的作用是剪切选中区域的文本。在命令行上使用该组合键则是往后删除一个字符组合。也就是说，删除光标左边的一个字母组合或单词。比如，我们在此命令行上使用了命令如下，“service network restart”，让我们的光标位于字符串的 restart 的后面，按下该组合键，看看有何效果？是不是变成“service network”了？确实是这样，如果我们使用 Backspace 键的话，则需要使用 7 次的按键才能达

[1]先按下 Ctrl+X，然后松开 X，继续按着 Ctrl 键，然后再按下 T 键，即可完成两个组合键的操作。别嫌麻烦，习惯就好了。

到一个 Ctrl+W 的组合键的效果。嗯，别只顾着看，动手试试？

16. Alt+. 快捷键

    此组合键是调出上一条命令的最后一个参数。如上一条命是“service network restart”，则“restart”就是最后一个参数。如果我们接下来要敲的命令需要用到上一条命令的最后一个参数，则可使用此快捷键，而不需要手工输入“restart”了，而且不会出错，节省敲击键盘的次数。如果我们接下来想重启 httpd 服务，则只需要输入“service httpd”，然后按下“Alt+.”即可补全上一条命令的“restart”。在有些终端上，按“Alt+.”组合键可能会没有效果，这时可以使用“ESC+.”组合键代替。在 Emacs 中，ESC 键与 Alt 键是等价的。可以动手试试该组合键的效果。

2.2 使用 man page 获得帮助

当我们遇到不会用的系统命令时，该怎么办呢？或许你第一个想到的是 Baidu 或 Google，这样想很正常，可以节省很多时间。如果每次遇到不会的命令，都去找网络，个人觉得这不是一件好的事情，不利于我们的提高。

如果不依靠互联网，该怎么解决呢？那就是依赖系统自带的 man page 了。通过它我们可以获取绝大部分的信息[2]，这正是锻炼我们的时候。当然我们也可以使用强大的 info 工具来查看帮助。

2.3 echo 与终端颜色

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：

1. n 不要在最后自动换行
2. 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：

   发出警告声；

[2]不过我们要能看得懂里面的英文，不然那就果断百度去吧！Google 貌似不能用了，唉！

前一个字符;
最后不加上换行符号;
换行但光标仍旧停留在原来的位置;
换行且光标移至行首;
光标移至行首，但不换行;
插入;
与相同;
插入字符;
插入（八进制）所代表的字符;
显示帮助
显示版本信息

2.3.1 终端颜色

echo 字体颜色和背景颜色

-e enable interpretation of the backslash-escaped characters listed below

字背景颜色范围:40-47

R	G	B	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

表 2-1 颜色表 [?]

黑
深红
绿

蓝色
紫色

深绿
白色

字颜色:30-37

ANSI 控制码的说明:

关闭所有属性
设置高亮度
下划线
闪烁
反显
消隐
—设置前景色
—设置背景色
光标上移行
光标下移行
光标右移行
光标左移行
设置光标位置
清屏
清除从光标到行尾的内容
保存光标位置
恢复光标位置
隐藏光标
显示光标

下面看一个例子:

输出结果为:

2.4 date 命令的使用

设日期

设时间

图 2.1 终端颜色效果

天前日期

分钟前时间

一个月前

日期格式转换

日期和时间

纳秒

换算成秒计算年至今的秒数

将时间戳换算成日期

将时间戳换算成日期

格式化系统启动时间多少秒前

2.5 yum 命令的使用

安装好系统时，在/etc/yum.repos.d 目录下回有一个 rhel-debuginfo.repo 的文件，我们这里以 redhat 系统为例进行讲解。不管这个配置文件的名字如何，但文件的扩展名须为.repo，如 redhat.repo 也是可以的。我们需要做一些准备工作。

准备系统镜像文件并挂载到本地：

复制镜像里的文件到本地目录：

修改这个配置文件：

几点说明：

1. [rhel-debuginfo] 中括号里的内容可以随意
2. name 这一行可有可无
3. baseurl 这行要指定我们的资源在哪里
4. file://说明我们使用什么协议，也可以是 ftp://等
5. /iso/Server 指明我们的源在/iso/Server 目录下

2.6 zypper 命令的使用

2.6.1 zypper 本地源的配置

SUSE 的 zypper 本地源配置起来跟 yum 的配置很相似，它们的配置文件有很多相似之处。不过，个人觉得 zypper 这个工具稍微强大些。在 SUSE 下，可以通过一条 zypper 的命令，即可完成 zypper 源的配置。

为什么会有本节内容呢？主要是 2014 年 9 月 26 日，Bash 爆出了一个漏洞，传闻比“心脏出血”漏洞还要猛，不知道是不是真的。以下是网上给出的验证方法，测试环境在我的 RHEL6 64bit 的虚拟机上，

同样，在我的 SUSE 11sp2 64bit 上运行也是同样的输出，输出内容略。以下几个包是 SUSE 的 OEM 厂商给出的 Bash 最新的升级包。

接下来的操作是把这些包放到一个目录里，然后把该目录做成系统的一个更新源。比如，把解压后的目录放到/opt 目录下，然后使用 zypper ar 添加该 zypper 源。

接下来，使用 zypper lr 验证下，

说明我们已成功添加 update 的源。另外，执行”zypper ar URI alias“后，会在 /etc/zypp/repo.d/目录下生成 alias.repo 配置文件。接下来，我们试试 zypper update 命令，看是不是可以真的可以升级？

以上说明可以进行升级的。接下来，我们使用 zypper ps 命令，可以查看有哪些终端还在使用之前没有升级过的 bash，

说明还有 bash 升级前的两个进程还在运行，我们可以退出这两个终端，再次登入系统，再次使用 zypper ps 命令来查看，就会看到”No processes using deleted files found.“的提示。

2.6.2 zypper 命令选项介绍

zypper 是 SuSE 系列下面的包管理工具，如同 2.5节的 yum 工具。

zypper 的几个重要选项：

选项	说明
repos, lr	列出库
sl	列出库（目的是与 rug 兼容）
addrepo, ar	添加库
sa	添加库（目的是与 rug 兼容）
renamerepo, nr	重命名指定的安装源
modifyrepo, mr	修改指定的安装源
refresh, ref	刷新所有安装源
clean	清除本地缓存

表 2-2 zypper 安装源操作选项

zypper 软件管理：

zypper 的查询选项：

选项	说明
install, in	安装软件包
remove, rm	删除软件包
verify, ve	检验软件包依赖关系的完整性
update, up	更新已安装的软件包到新的版本
dist-upgrade, dup	整个系统的升级
source-install, si	安装源代码软件包和它们的编译依赖

表 2-3 zypper 软件管理选项

选项	说明
search, se	安装软件包
info, if	删除软件包
patch-info	检验软件包依赖关系的完整性
product-info	更新已安装的软件包到新的版本
patch-check, pchk	整个系统的升级
list-updates, lu	安装源代码软件包和它们的编译依赖
patches, pch	列出所有可用的补丁
packages, pa	列出所有可用的软件包
patterns, pt	列出所有可用的模式
products, pd	列出所有可用的产品
what-provides, wp	列出能够提供指定功能的软件包

表 2-4 zypper 工具查选选项

2.7 parted 命令的使用

Gnu/Linux 系统的分区工具通常可以使用 fdisk 与 parted。我们用的比较多的工具就是 fdisk 了，这里不介绍它的使用了。这里简单的介绍如何使用 parted 工具，对于分区表通常有 MBR 分区表和 GPT 分区表对于磁盘大小小于 2T 的磁盘，我们可以使用 fdisk 和 parted 命令工具进行分区对于 MBR 分区表的特点（通常使用 fdisk 命令进行分区）所支持的最大磁盘大小：2T 最多支持 4 个主分区或者是 3 个主分区加上一个扩展分区对于 GPT 分区表的特点（使用 parted 命令进行分区）支持最大卷：18EB（1EB=1024TB）最多

支持 128 个分区

对于 parted 命令工具分区的介绍

最后，fdisk 与 parted 有些差异。fdisk 分区完毕后，需要使用“w”命令才能保存之前所做的一些操作；而 parted 则是实时的，每一步操作不需要保存，即时生效。

2.8 mount 命令的使用

如何挂载 iso 镜像文件呢？我们可以使用一下 mount 命令：

意思是把挂载到目录下，不过
你得事先有这个镜像文件

几个常用的命令：

命令	说明
repolist	列出我们有哪些 yum 仓库
list	列出仓库里有哪些软件包
install	安装软件包的命令
groupinstall	安装软件包组
erase	移除一个或多个软件包

表 2-5 yum 常用命令选项

配置好之后，如何使用呢？直接看操作吧：

1. 列出我们有哪些 yum 仓库

2. 列出仓库里的包

一些说明：

1. 第一列是我们的软件包名
2. 第二列是对应软件包的版本号
3. 第三列

+ installed 表明该软件包已安装
+ rhel-debuginfo 表明包未安装

2.9 grep 命令的使用

grep(global regular expression pattern 的缩写)。其实可以把它理解为过滤关键字用的一个程序。具体怎么用，还是看一个实例吧，然后结束本节内容。

去掉文件里的注释行和空白行

这里我们使用 grep 命令及 cut 命令一起把 eth0 上的 IP 给取出来，看操作：

-A NUM	打印出紧随匹配的行之后的下文 NUM 行
-B NUM	打印出匹配的行之前的上文 NUM 行
-C NUM	打印出匹配的行的上下文前后各 NUM 行
-b	在输出的每行前面同时打印出当前行在输入文件中的字节偏移量
-c	显示匹配的行数
-f file	从文件 file 中获取模式，每行一个
-H	为每个匹配的文件打印文件名
-I	不搜索二进制文件
-i	忽略大小写
-l	只显示有匹配的文件的文件名
-L	只显示未匹配的文件的文件名
-n	输出行号
-o	只显示匹配字段
-q	quiet 静默模式
-v	只显示不匹配的行

表 2-6 grep 常用选项

使用命令，匹配关键字，缩小范围

使用命令，把范围再次缩小些

是不是快出来了，再使用一次，地址就出来了
自己写出来吧，我已经写得很多了！

2.10 crontab 命令的使用

假想这样一个场景：每天的凌晨两点，领导都会要求你你重启下服务器。这时，你该怎么办？你是不是每天凌晨两点都要从温暖的被窝里爬出来，然后远程连接服务器，然后重启服务器，然后重新钻进被窝，然后失眠了...。每天都如此，我想你一定会奔溃的。

crontab 命令可以解救你！crontab 几个字段的说明：

查看当前用户的

上面语句的意思是，每分钟去执行目录下的
脚本，只要系统一直运行，它就会循环往复的执行到海枯到石烂。汗！

编辑

2.11 find 命令的使用

find 命令很强大，强大到可以写很多东西。这里就介绍如何简单的使用。直接看例子吧：

文件无创建时间
使用时间
内容修改时间
状态改变时间权限、属主
时间默认以小时为单位当前时间到向前小时为天向前小时为天
且匹配两个条件参数可以确定时间范围
或匹配任意一个条件

按文件名查找

查找某一类型文件

按照文件权限查找

按照文件属主查找

按照文件所属的组来查找文件

文件使用时间在天以内

文件使用时间在天以前

文件内容改变时间在天以内

文件内容改变时间在天以前

文件状态改变时间在天前

文件状态改变时间在天内

查找文件长度大于字节的文件

按名字查找文件传递给后命令

查找文件名不取路径

批量改名查找替换为文件

查找任意一个关键字

路径

2.12 top 命令的使用

top 命令可动态显示服务器的进程信息，用户可以通过按键来刷新当前状态。别的不多说，给个例子看看：

2.13 free 命令的使用

buffer：缓存磁盘上的数据 cached：缓存的是将要写往磁盘中的数据

buffer=used-buffer-cached cached=free+buffer+cached

2.14 xargs 命令的使用

2.15 tr 命令的使用

2.16 tar 命令的使用

tar 命令是一个打包与解包的一个工具，功能很强大。下面介绍一些常用选项及使用示例。

通用选项：

选项	说明
j	使用 bzip2 的压缩方式
t	列出压缩包里有哪些文件，并不解压
z	使用 gzip 的压缩方式
f	该选项是必选的，不管是压缩还是解压缩
v	压缩或解压缩时，查看其详细过程

表 2-7 tar 通用选项

压缩时用的选项：

选项	说明
c	打包时用的选项，选项 c 与 x 不能同时出现

表 2-8 tar 压缩选项

解压缩用的选项：

举例说明：

选项	说明
x	解包时用的选项，选项 c 与 x 不能同时出现

表 2-9 tar 解压缩选项

这里有两个文件，第一个为目录，第二个为压缩包

创建包，不压缩

上面的例子我们使用选项，使我们可以看到过程。其中是我们创建的包，是针对这个目录的。

创建包，以方式压缩

创建包，以的方式压缩

查看压缩包的内容，并不解压缩

解压缩
不管是包，还是以或压缩的方式，我们使用一下这条命令都是通用的

这里我们没有加选项，可以加上选项以看到解压过程

2.17 read 命令的使用

2.18 cut 命令的使用

cut 命令有的时候很有用，比如要获得指定的以某个分隔符分割的列时，它就可以做到。其实还有比它更强大的工具，如 sed 及 awk 等，这里并不介绍它们，后续章节有介绍。这里就不提及了，跳过它们吧。下面直接看例子吧：

指定区分列的定界，默认是
指定要显示的列
按字符来切割，如
（取文件第一行的前个字符）

实验：取地址

以冒号为分割，显示第一列

以冒号为分割，显示第二列

以空格为分割，显示第一列

我想，知道这么多，应该就可以了。

2.19　sort 命令的使用

2.20　less 命令的使用

2.21　lsof 命令的使用

lsof 是“list open files”的缩写，是一个列出当前系统打开文件的工具。在 linux 环境下，任何事物都以文件的形式存在，通过文件不仅仅可以访问常规数据，还可以访问网络连接和硬件。所以如传输控制协议 (TCP) 和用户数据报协议 (UDP) 套接字等，系统在后台都为该应用程序分配了一个文件描述符，无论这个文件的本质如何，该文件描述符为应用程序与基础操作系统之间的交互提供了通用接口。因为应用程序打开文件的描述符列表提供了大量关于这个应用程序本身的信息，因此通过 lsof 工具能够查看这个列表对系统监测以及排错将是很有帮助的。

lsof 不加任何选项，默认输出所有活动进程打开的文件。

2.21.1 恢复删除的文件

有一次做研发的一位同事，因为系统根目录空间不足，想释放磁盘空间，结果使用 find 命令找到了单个文件大于 1GB 的文件，发现/var/log/messages

和 /var/log/warn 文件每个都是 1.3GB。他的系统根目录空间只有 50GB 的容量，结果把/var/log/messages 日志文件及/var/log/warn 日志文给删除了。删除之后，却没有达到的目的，磁盘使用空间依然是 100%。

殊不知，当进程打开了某个文件时，只要该进程保持打开该文件，即使将其删除，它依然存在于磁盘中。这意味着，进程并不知道文件已经被删除，它仍然可以向打开该文件时提供给它的文件描述符进行读取和写入。除了该进程之外，这个文件是不可见的，因为已经删除了其相应的目录条目。

拿/var/log/messages 文件为例，看看如何在故意删除后如何找回来。我们先看一下/var/log/messages 文件是什么进程打开的。

该命令的输出表明，/var/log/messages 文件由 syslog-ng 进程打开，当前的进程号为 1493，用户身份是 root，打开的文件描述符为 10 且该文件处于只写模式，对应的 TYPE（类型）为 REG（常规）文件、磁盘位置、文件大小、索引节点。下面一一验证：

现在，我们可以删除该文件以模拟误删除，

文件已被删除，看怎么恢复吧！从上面的输出信息可以看出之前打开该文件的进程号及文件描述符，有了这两个信息就足够了。接下来，我们去 cat 一下/proc 目录中相应的目录中的文件描述符，

通过文件描述符查看了相应的数据，那么就可以使用 I/O 重定向将其复制到文件中，如 cat /proc/1493/fd/10 > /tmp/messages。此时，可以中止该守护进程（这将删除 FD，从而删除相应的文件），将这个临时文件复制到所需的位置，然后重新启动该守护进程。

我们可以看到，已删除的/var/log/messages 文件已回来了！对于许多应用程序，尤其是日志文件和数据库，这种恢复删除文件的方法非常有用。

2.22 netstat 命令的使用

2.23 tcpdump 命令的使用

tcpdump 是一款基于命令行的工具，可以通过不同的命令行选项来改变其状态、捕获数据的数量及捕获数据的方法。tcpdump 提供的丰富选项可以使你很容易的改变程序的运行方式。

下面列举一部分比较常用的选项，

选项	说明
i	指定侦听的网络接口
v	指定详细模式输出详细的报文信息
vv	指定非常详细的模式输出及非常详细的报文信息
vvv	指定更加详细的模式输出及更详细的报文信息
x	规定 tcpdump 以 16 进制数格式显示数据包
X	规定 tcpdump 以 hex 及 ASCII 格式显示输出
XX	同上，并显示以太网头部信息
n	在捕获过程中不需要向 DNS 查询 IP 地址（显示 IP 地址及端口号）
F	从指定的文件中读取表达式
D	显示 tcpdump 可以侦听的网络接口列表
c	指定捕获多少数据包，然后停止捕获
w	把捕获到的信息写到一个文件中
s	设置捕获数据包的长度为 length

表 2-10 tcpdump 常用选项

第一种是关于类型的关键字，主要包括 host, net, port, 例如 host 210.27.38.1，指明 210.27.38.1 是一台主机，net 202.0.0.0 指明 202.0.0.0 是一个网络地址，port 23 指明端口是 23。如果没有指定类型，缺省的类型是 host。

想要截获所有的主机收到的和发出的所有的数据包：

对本机的端口进行监视，为的服务端口

如果想要获取主机接收或发出的包，使用如下命令：

想要获取主机和主机或的通信，使用命令：
在命令行中使用括号时，一定要转义

如果想要获取主机除了和主机之外所有主机通信的包，则：

2.24 traceroute 命令的使用

2.25 wget 命令的使用

2.26 screen 命令的使用

运维人员经常需要 SSH 到远程登录 Unix/Linux 服务器，经常执行一些需要很长时间才能完成的任务，比如 PCIe SSD 卡的稳定性测试、InfiniBand 网卡测试、系统备份等等。通常，我们会为每一个任务打开一个远程终端，在执行任务期间，必须等待命令执行完毕，表明该任务正常结束。在此期间，不能关闭终端或断开链接，否则这个任务一起被终止。

2.26.1 screen 常用参数

2.26.2 使用 screen

1. 创建一个新窗口
2. 查看已创建的窗口
3. 会话分离与恢复

2.27 iptables 的使用

设定为

设定为

设定为

定制源地址访问策略

接收来自的访问

拒绝来自网段的访问

目标地址的访问给予记录并查看

定制端口访问策略

拒绝任何地址访问本机的端口

拒绝网段的的源端口访问

定制端的防火墙访问状态

清除所有已经存在的规则

设定预设策略除了设为其他为

开放本机的可以自由访问

设定有相关的封包状态可以进入本机

定制防火墙的地址访问策略

清除所以已经存的规则

将设为

将目标计算机的设为

设定包，状态为的被掉

定制防火墙的访问策略

清除所有策略

重置为

通过设定来源于网段的通过转发出去

通过观察转发的数据包

定制防火墙的访问策略

清除所有策略

重置为

通过设定来源于网段通过转发出去

用观察转发的数据包

端口转发访问策略

清除所有策略

通过设定为所有访问的端口，都访问到的端口

设定所有到的端口的数据包都通过转发

设定回应数据包即通过的设定使通讯正常

修改目标地址在路由之前

修改源地址在路由之后

2.28 qperf 命令的使用

我们在做网络服务器的时候，通常会很关心网络的带宽和延迟。因为我们的很多协议都是 request-response 协议，延迟决定了最大的 QPS，而带宽决定了最大的负荷。通常我们知道自己的网卡是什么型号，交换机什么型号，

主机之间的物理距离是多少，理论上是知道带宽和延迟是多少的。但是现实的情况是，真正的带宽和延迟情况会有很多变数的，比如说网卡驱动，交换机跳数，丢包率，协议栈配置，就实际速度而言，都很大的影响了数值的估算。所以我们需要找到工具来实际测量下。

SUSE11sp2 发行版里面自带，方便安装，专业有效，能够针对 TCP 和 RDMA 进行带宽和延迟的详细测试。

由于我们需要测试 Infiniband 的传输速率，在安装之前请先确认安装了 InfiniBand 的相关包，比如 librdmacm，libibverbs 等。另外，也可以选择使用源码包编译和安装 qperf，但是需要注意，在安装之前也需要将 infiniband 相关的包先安装上，否则 RDMA 的相关测试也将无法进行。

2.28.1 参数说明及示例

qperf 分为服务器端和客户端。客户端通过发送请求并获得响应来获得服务器端和客户端之间的网络带宽以及延迟等信息。

参数名	参数说明
<server_ip>	指定服务器的地址
time	指定网络测试时间。默认单位为秒，单位可以通过后缀为 m,h,d 指定为分钟，小时，天
conf	测试输出中显示本地和远端服务器和操作系统配置
use_bits_per_sec	使用 b(bit) 而不是 B(byte) 来显示网络速度
precision 2	设置显示小数点后几位。这里设置为显示小数点后两位
verbose_more	显示更详细的配置和状态信息
loop msg_size:1:1025k:*2	loop 表示对指定的指标值进行轮询。这里设置为对 msg_size 轮询 1，2，4，8…1024k，获得对应的测试结果，下次测试的指标值是上次测试指标值的 *2 倍
tcp_bw	对 tcp 的带宽进行测试
tcp_lat	对 tcp 的延迟进行测试
udp_bw	对 udp 的带宽进行测试
udp_lat	对 udp 的延迟进行测试
sdp_bw	对 sdp 的延迟进行测试
sdp_lat	对 sdp 的延迟进行测试

2.29 iperf 命令的使用

iperf 工具我们主要

首先到官网获取 iperf 工具，并把该工具放到合适的位置。

2.29.1 参数说明及示例

参数名	参数说明
–server	以服务端模式运行
–udp	指定测试 UDP，默认为 TCP 带宽测试
–client <host>	以客户端模式运行，并连接 <host>
–bandwidth	指定测试中所使用的带宽，单位为 [KM]，默认为 1Mbit/sec
–time	指定测试时间，单位为秒
–interval	指定多少时间间隔来报告测试结果，时间单位为秒
–format [kmKM]	指定报告的输出格式，单位分别为 Kbits，Mbits，Kbytes，Mbytes

1. 以太网 UDP 丢包率测试

2. InfiniBand 网络 UDP 丢包率测试

3. 如果不指定 –udp 选项，默认就是测试 TCP 带宽

2.30 vmstat 命令的使用

Linux 下 vmstat 输出释疑：

procs r 列表示运行和等待 cpu 时间片的进程数，如果长期大于 cpu 个数，说明 cpu 不足，需要增加 cpu。

b 列表示在等待资源的进程数，比如正在等待 I/O、或者内存交换等。

memory swpd 切换到内存交换区的内存数量 (k 表示)。如果 swpd 的值不为 0，或者比较大，比如超过了 100m，只要 si、so 的值长期为 0，系统性能还是正常

free 当前的空闲页面列表中内存数量 (k 表示)

buff 作为 buffer cache 的内存数量，一般对块设备的读写才需要缓冲。

cache: 作为 page cache 的内存数量，一般作为文件系统的 cache，如果 cache 较大，说明用到 cache 的文件较多，如果此时 IO 中 bi 比较小，说明文件系统效率比较好。

swap si 由内存进入内存交换区数量。

so 由内存交换区进入内存数量。

IO bi 从块设备读入数据的总量（读磁盘）（每秒 kb）。

bo 块设备写入数据的总量（写磁盘）（每秒 kb）

system 显示采集间隔内发生的中断数

in 列表示在某一时间间隔中观测到的每秒设备中断数。

cs 列表示每秒产生的上下文切换次数，如当 cs 比磁盘 I/O 和网络信息包速率高得多，都应进行进一步调查。

cpu 表示 cpu 的使用状态

us 列显示了用户方式下所花费 CPU 时间的百分比。us 的值比较高时，说明用户进程消耗的 cpu 时间多，但是如果长期大于 50

sy 列显示了内核进程所花费的 cpu 时间的百分比。这里 us + sy 的参考值为 8080

wa 列显示了 IO 等待所占用的 CPU 时间的百分比。这里 wa 的参考值为 30 这可能是磁盘大量随机访问造成的，也可能磁盘或者磁盘访问控制器的带宽瓶颈造成的 (主要是块操作)。

id 列显示了 cpu 处在空闲状态的时间百分比

2.31 iostat 命令的使用

2.32 sar 命令的使用

sar 命令行的常用格式:

在命令行中，n 和 t 两个参数组合起来定义采样间隔和次数，t 为采样间隔，是必须有的参数，n 为采样次数，是可选的，默认值是 1，-o file 表示将命令结果以二进制格式存放在文件中，file 在此处不是关键字，是文件名。options 为命令行选项，sar 命令的选项很多，下面只列出常用选项:

选项	说明
-A	所有报告的总和
-u	CPU 利用率
-v	进程、I 节点、文件和锁表状态
-d	硬盘使用报告
-r	没有使用的内存页面和硬盘块
-g	串口 I/O 的情况 (centos 5 中无此选项)
-b	缓冲区使用情况
-a	文件读写情况
-c	系统调用情况
-R	进程的活动情况
-y	终端设备活动情况
-w	系统交换活动

表 2-11 sar 常用选项

例一：使用命令行 sar -u t n

例如，每 60 秒采样一次，连续采样 5 次，观察 CPU 的使用情况，并将采样结果以二进制形式存入当前目录下的文件 zhou 中，需键入如下命令:

在显示内容包括：

：处在用户模式下的时间百分比。
：处在系统模式下的时间百分比。
：等待输入输出完成时间的百分比。
：空闲时间百分比。

在所有的显示中，我们应主要注意%wio 和%idle，%wio 的值过高，表示硬盘存在 I/O 瓶颈，%idle 值高，表示 CPU 较空闲，如果%idle 值高但系统响应慢时，有可能是 CPU 等待分配内存，此时应加大内存容量。%idle 值如果持续低于 10，那么系统的 CPU 处理能力相对较低，表明系统中最需要解决的资源是 CPU。

如果要查看二进制文件 zhou 中的内容，则需键入如下 sar 命令：

可见，sar 命令即可以实时采样，又可以对以往的采样结果进行查询。

例二：使用命行 sar -v t n

例如，每 30 秒采样一次，连续采样 5 次，观察核心表的状态，需键入如下命令：

显示内容包括:

proc-sz：目前核心中正在使用或分配的进程表的表项数，由核心参数 MAX-PROC 控制。inod-sz：目前核心中正在使用或分配的 i 节点表的表项数，由核心参数 MAX-INODE 控制 file-sz：目前核心中正在使用或分配的文件表的表项数，由核心参数 MAX-FILE 控制。ov：溢出出现的次数。Lock-sz：目前核心中正在使用或分配的记录加锁的表项数，由核心参数 MAX-FLCKRE 控制。

显示格式为: 实际使用表项/可以使用的表项数

显示内容表示，核心使用完全正常，三个表没有出现溢出现象，核心参数不需调整，如果出现溢出时，要调整相应的核心参数，将对应的表项数加大。

例三：使用命行 sar -d t n

例如，每 30 秒采样一次，连续采样 5 次，报告设备使用情况，需键入如下命令:

显示内容包括:

: 命令正在监视的块设备的名字。
: 设备忙时，传送请求所占时间的百分比。
: 队列站满时，未完成请求数量的平均值。
: 每秒传送到设备或从设备传出的数据量。
: 每秒传送的块数，每块字节。
: 队列占满时传送请求等待队列空闲的平均时间。
: 完成传送请求所需平均时间（毫秒）。

在显示的内容中，wd-0 是硬盘的名字，%busy 的值比较小，说明用于处理传送请求的有效时间太少，文件系统效率不高，一般来讲，%busy 值高些，avque 值低些，文件系统的效率比较高，如果%busy 和 avque 值相对比较高，说明硬盘传输速度太慢，需调整。

例四：使用命行 sar -b t n

例如，每 30 秒采样一次，连续采样 5 次，报告缓冲区的使用情况，需键入如下命令：

显示内容包括：

：每秒从硬盘读入系统缓冲区的物理块数。
：平均每秒从系统读出的逻辑块数。
：在中进行逻辑读的百分比。
：平均每秒从系统向磁盘所写的物理块数。
：平均每秒写到系统逻辑块数。
：在中进行逻辑读的百分比。
：平均每秒请求物理读的次数。
：平均每秒请求物理写的次数。

在显示的内容中，最重要的是%cache 和%wcache 两列，它们的值体现着 buffer 的使用效率，%rcache 的值小于 90 或者%wcache 的值低于 65，应适当增加系统 buffer 的数量，buffer 数量由核心参数 NBUF 控制，使%rcache 达到 90 左右，%wcache 达到 80 左右。但 buffer 参数值的多少影响 I/O 效率，增加 buffer，应在较大内存的情况下，否则系统效率反而得不到提高。

例五：使用命行 sar -g t n 例如，每 30 秒采样一次，连续采样 5 次，报告串口 I/O 的操作情况，需键入如下命令：

显示内容包括:

: 每秒在串口硬件出现的溢出。
: 每秒在串口的直接输入输出通道高速缓存出现的溢出。
: 每秒字符队列出现的溢出。

在显示的内容中，每一列的值都是零，表明在采样时间内，系统中没有发生串口 I/O 溢出现象。

sar 命令的用法很多，有时判断一个问题，需要几个 sar 命令结合起来使用，比如，怀疑 CPU 存在瓶颈，可用 sar -u 和 sar -q 来看，怀疑 I/O 存在瓶颈，可用 sar -b、sar -u 和 sar-d 来看。

第 3 章　vim 编辑器

当我们基本熟悉了操作系统的使用，是不是还少了什么件事？恭喜你猜对了，那就是在系统里面编辑一些东西，我们总该留点什么东西吧，你说是不是呢？

传说中，蔚蓝的星球上流传着两大神器，一个是 vi[1]，一个是 emacs。据说 Vim 是编辑器之神，而 Emacs 是神的编辑器。追求独步天下的高手和低手们争着一睹它们的风采，可看到它们朴素单薄的界面后，不禁心下怀疑：这就是神器吗？至有人生了轻视之心。

本文就向你展示如何使用 vim，掌握它最基本的使用即可。作者刚开始使用 vim 的时候，总是不由自主地想动动旁边的鼠标，大概用了两个晚上的时间才熟练的掌握。Emacs 可以根据自己兴趣看一看，作者使用了一年多的时间，也没觉得自己比那些使用 vim 的家伙强[2]，我的 vi 用的比 Emacs 要熟练。嘿嘿！

3.1　vim 的几种模式

vim 有两种模式，一个为输入模式，一个为命令模式。在系统提示字符 (如 $、#) 下敲入 vim filename，如果没有 filename 这个文件则会创建，若有则打开。vim 可以自动帮你载入所要编辑的文件或是开启一个新文件（如果该文件不存在或缺少文件名）。进入 vim 后屏幕左方会出现波浪符号，凡是列首有该符号就代表此列目前是空的。

如上所述，vim 存在两种模式：指令模式和输入模式。在指令模式下输入的按键将做为指令来处理：如输入 i，vim 即认为是在当前位置插入字符。而在输入模式下，vim 则把输入的按键当作插入的字符来处理。指令模式切

[1]vim 是 vi 的升级版，一般认为 vi 既 vim，vim 既 vi。也可理解为 vi 为 vim 的一个别名。

[2]这里作者并不想卷入编辑器之战，不想得罪任何一方阵营，两种编辑器我都会使用，上班时用 vi，不上班时用 Emacs；如果真的得罪了某方阵营，请原谅我的无知。

换到输入模式只需键入相应的输入命令即可（如 a，A），而要从输入模式切换到指令模式，则需在输入模式下键入 ESC 键，如果不晓得现在是处于什么模式，可以多按几次 ESC，系统如发出哔哔声就表示已处于指令模式下了。

3.1.1 输入模式

如何进入输入模式？下面列出几个进入插入模式的指令：

指令	说明
i	在光标所在处进入插入模式
I	在当前行的行首进入插入模式
a	在光标的后面进入插入模式
A	在当前行的行尾进行插入模式
o	在光标所在行的下面新开一行
O	在光标所在行的上面新开一行
s	删除光标所在字符并进入插入模式
S	删除光标所在行并进入输入模式

表 3-1 vim 进入插入模式指令

大家注意到了没有，大写与小写分别控制反向与正向。如何退出呢？在插入模式下，需要按下键盘的 ESC 键，进入命令模式，再从命令模式下，输入”:wq“即可退出。wq 的意思是保存并退出，如果不想保存，只需输入”:q“即可。如果修改后，发现改错了文件，并不想保存了，在命令模式下输入”:q!“即可，加上”!“作用是强制退出不保存，其实也可以强制保存并退出的，该怎么输入？想想看。

3.1.2 命令模式

在刚打开 vim 编辑器时，这时所处的模式是命令模式；从输入模式返回到命令模式，只需要按下键盘上的 ESC 键即可返回到命令模式。下面介绍几组很实用的指令，主要涉及删除、复制及粘贴的指令，还有几个控制光标移动的指令。

光标方向控制：

撤销指令：

指令	说明
h	光标向前移动
j	光标向下移动
k	光标向上移动
l	光标想后移动
G	按下 shift 与 G 组合，则跳到文件末尾
gg	连续按两次 G，跳到文件的首行

表 3-2 vim 方向控制键

指令	说明
u	按一次，撤销上一次的操作，按两次呢？哈哈
:e!	直接回到了刚打开时的状态

删除指令：

指令	说明
dd	删除当前光标所在行
ndd	删除当前光标所在行及以下行共 n 行，其中 n 为数字

3.1.3 vim 的其他一些指令

3.2 vim 的一些小技巧

打开文件定位到指定行

打开多个文件

垂直分屏

水平分屏

上下分割打开新文件

左右分割打开新文件

操作多个文件间操作大写操作关闭当前窗口屏幕高度一样增加高度移动光标所在屏右左上下中下一个
编辑下一个文件
编辑下二个文件
编辑前一个文件
回到首文件
打开行号
取消行号
跳转到
取消高亮
设置自动缩进
查看文本格式
改为格式
向前翻页
向后翻页
字符字符全部替换
文档加密

3.3 vim 复制粘贴及剪切板

复制粘贴指令:

vim 有 12 个粘贴板，分别是 0、1、2、...、9、 a 、 “、+; 用:reg 命令可以查看各个粘贴板里的内容。效果如图所示:

在 vim 中简单用 y 只是复制到”(双引号) 粘贴板里，同样用 p 粘贴的内容也是这个粘贴板里的内容; 要将 vim 的内容复制到某个粘贴板，需要退出编辑模式，进入正常模式后，选择要将复制的内容，然后按”Ny(注意带引号)

指令	说明
yy	复制当前光标所在行
nyy	复制当前行及以下行共 n 行，n 为数字
p	将复制的内容粘贴到当前光标的下面一行，可以与数字一起使用，如 np
P	将复制的内容粘贴到当前光标的上面一行，可以与数字一起使用，如 nP

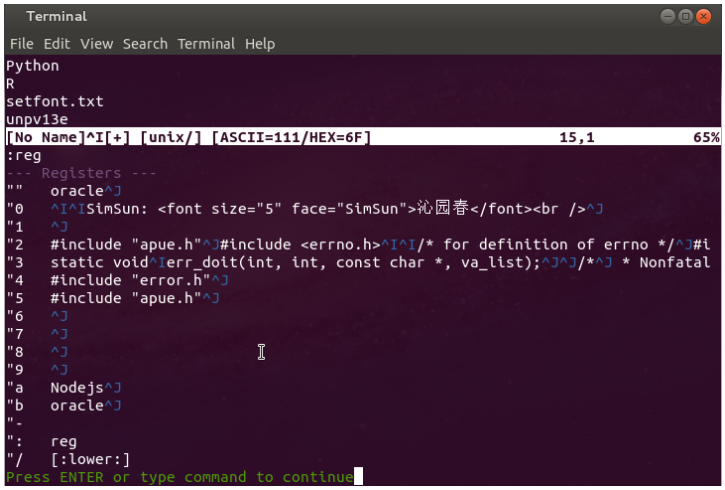


图 3.1 vim 粘贴板一览

完成复制，其中N为粘贴板号 (注意是按一下双引号后按粘贴板号再按 y)，比如要把内容复制到粘贴板 a ，选中内容后按”ay 就可以了。

有两点需要说明一下：” 号粘贴板 (临时粘贴板) 比较特殊，直接按 y 就复制到这个粘贴板中了，直接按 p 就粘贴这个粘贴板中的内容；+ 号粘贴板是系统粘贴板，用”+ y 将内容复制到该粘贴板后可以使用 Ctrl+V 将其粘贴到其他文档 (如 firefox、gedit) 中，同理，要把在其他地方用 Ctrl+C 或右键复制的内容复制到 vim 中，需要在正常模式下按”+p；

3.4 vim 配置文件

我相信，每个有想法的家伙肯定不会满足默认的配置，他们总喜欢捣鼓。这个时候我们就可以定制自己的编辑器了，以满足我们日常编辑的需求。Emacs 的定制配置更是令人蛋疼，诸多选项、诸多值……。作者曾经也是为此事而蛋疼不已，只是学会了在 Emacs 里配置 LaTeX 的一些配置项，其

它神马的，作者是一窍不通。

令人蛋疼的事就不多说，我比较喜欢简洁的配置。本人也不会开发，一些简单的配置就能满足日常需要了，再说这个东东用的也不多。其它高大上的配置东东对我来说也是浮云。下面是我常用的 vim 的配置文件，配置如下：

效果如下：

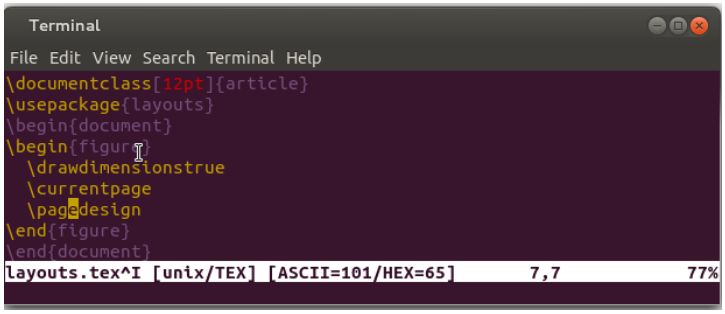


图 3.2 vim 配置文件效果

哦！天呢。根本停不下来了。就说这么多吧，如果我要再写下去，就不止这么多。呵呵！只不过是两年没用认真的使用了。呵呵！vim 的使用就介绍这么多，我想仅仅学会这些就能让初学者的效率有很大的提高，但还不够高，这里并没有介绍正则表达式等。

第 4 章　正则表达式

Regular expression (abbreviated regex or regexp) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. ”find and replace”like operations.

正则表达式就是由一系列特殊字符组成的字符串, 其中每个特殊字符都被称为元字符, 这些元字符并不表示为它们字面上的含义, 而会被解释为一些特定的含义. 具个例子, 比如引用符号, 可能就是表示某人的演讲内容, 同上, 也可能表示为我们下面将要讲到的符号的元 -含义. 正则表达式其实是由普通字符和元字符共同组成的集合, 这个集合用来匹配 (或指定) 模式.

一个正则表达式会包含下列一项或多项:

1. 一个字符集. 这里所指的字符集只包含普通字符, 这些字符只表示它们的字面含义. 正则表达式的最简单形式就是只包含字符集, 而不包含元字符.
2. 锚. 锚指定了正则表达式所要匹配的文本在文本行中所处的位置. 比如, ^ 和 $ 就是锚.
3. 修饰符. 它们扩大或缩小 (修改) 了正则表达式匹配文本的范围. 修饰符包含星号, 括号, 和反斜杠.

4.1　正则表达式语法

4.2　一些实例

第 5 章　awk

如果要格式化报文或从一个大的文本文件中抽取数据包，那么 awk 可以完成这些任务。它在文本浏览和数据的熟练使用上性能优异。与其它大多数 Unix/Linux 命令不同的是，从名字上看，我们一眼看不出 awk 的功能：它既不是具有独立意义的英文单词，也不是几个相关单词的缩写。事实上，awk 是三个人名的缩写，他们是：Aho、(Peter) Weinberg 和 (Brain)Kernighan。正是这三个家伙创造了 awk，一个优秀的样式扫描与处理工具。本章只介绍 Gnu 的 awk 版本 gawk，其他 awk 版本不做介绍。

awk 的工作流程：

5.1　初步使用

假如我们有文件 employee.txt，内容如下：

第一列表示员工姓名，第二列表示每小时加班所得的人民币，第三列为加班多少小时。我们现在就来打印出那些加班大于 0 个小时的员工及其加班费，在命令行只需执行这样一行命令即可：

我们会得到这样的输出：

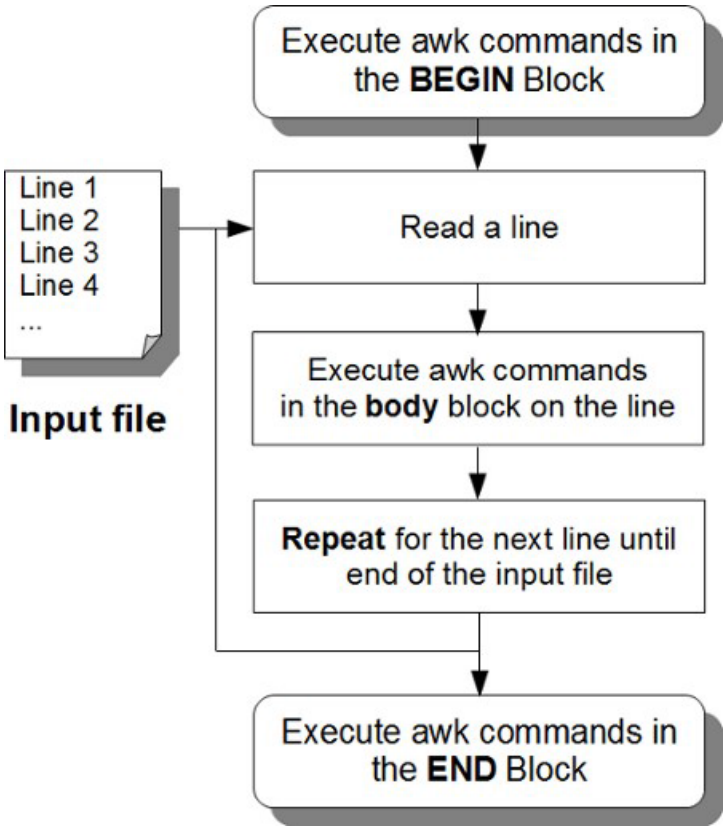


图 5.1　awk 工作流程

上面的命令告诉系统运行 awk，使用单引号里的程序，并从 employee.txt 文件里获得自己的数据。单引号里的部分完全就是 awk 程序。它包含了单个的 pattern-action（模式 -动作）语句。动作 $3 > 0，匹配输入的每一行的第三列或域，如果该列大于 0，然后就执行下面的动作

对于每匹配到的行，就会打印第一个域并同时计算第二个域与第三个域的乘积。

如果我们想看看哪个家伙这个月没有加班，可以使用下面的命令：

这里的模式，$3 == 0，将会匹配每一输入行的第三个域，如果等于 0，就会执行下面的动作

打印第一个域。

5.2 awk 程序结构

awk 的基本操作是对输入的行做逐一扫描，搜索这些行中匹配到的任意模式，然后执行相应的动作。然后下一行被读进来并从头开始匹配到行末，直到输入文件的所有行被读进来。就 $3 > 0 来说，当满足时，说明该条件为真。

在上面的例子中，只有一个模式和动作，可以称为单 pattern-action 语句；对于每一行，匹配到第三个字段等于 0 时，第一个字段将会被打印。

另外，模式和动作并不总是成对出现的，就是说，在 pattern-action 语句中，可能只有模式或者只有动作。如果一个 pattern 没有相应的 action，例如，

然后，每一被匹配的行都将会被打印，

如果一个 action 没有 pattern 时，例如，

此时，将会打印每一行的第一个域。由上可知，patterns 和 actions 都是可选的。actions 使用大括号括起来以与 patterns 区分。

5.2.1 BEGIN 与 END

特殊的 BEGIN 模式在第一个输入文件的第一行被匹配，而 END 则是在最后一个文件的最后被处理后匹配。下面的程序将会打印一个头部：

我们将会得到这样的输出：

我们可以在一行同时输入多条语句，这些语句之间需要用分号分开。注意到，*print ""* 意思是打印一个空行，它与 print 不一样，仅仅一个 print 语句将打印整行。

5.3 域和记录

awk 执行时，其浏览域标记为 $1，$2,...$n。这种方法称为域标识。使用这些域标识将更容易对域进一步处理。

5.4 内置变量

awk 有一些内置的变量，变量如下：

5.4.1 One-liners awk 程序

这一节介绍几个很有用的、短小精悍的 awk 程序。

1. 打印输入文件的总行数
2. 打印指定的行
3. 打印输入行的最后一个字段
4. 打印最后一行的最后一个字段
5. 打印每一输入行并擦除第二个字段

变量	说明
ARGC	命令行中参数的个数
ARGV	包含命令行参数的数组
CONVFMT	用于数字的字符串转换格式（默认格式为%.6g）
ENVIRON	环境变量的关联数组
FILENAME	当前文件名
NR	当前记录的个数（当前的行号）
FNR	当前记录的个数（仅为当前文件，而非全部输入文件）
FS	字段分割符（默认为空格）
NF	当前记录中的字段个数
OFMT	数字的输出格式（默认格式为%.6g）
OFS	输出字符的分割符（默认为空格）
ORS	输出记录分割符（默认为换行）

5.5 条件和循环

这一节包含了一些基本的编程结构。它覆盖了 awk 程序设计语言中的所有控制结构。在 awk 中，这些条件和循环的语法用起来比较简单。实际上，awk 中的条件和循环结构的语法借鉴 C 语言。因此，通过学习 awk，我们也同样在学习 C 语言。

5.5.1 if-else 语句

awk 提供一个 *if-else* 语句，这些语句只能用在 actions 中。下面的这个程序使用的还是5.1节的例子。在这个例子中，我们会计算那些每小时加班费大于 6 块的家伙，他们的总加班费及平均加班费。

我们将会得到下面的输出结果：

5.5.2 while 循环

while 语句有一个 condition 和一个 body。当 condition 为真时，body 中的语句将被重复执行。我们使用下面的程序计算这样的一个公式 $value = amount(1 + rate)^{years}$

如何执行该程序？看操作，

```
回车
输入三组数字后，回车
```

5.5.3 for 循环

我们使用 for 语句改写第5.5.2节的 while 程序。如下：

5.5.4 do 循环

5.5.5 影响流控制的语句

5.6 数组

数组是可以用来存储一组数据的变量。通常这些数据之间具有某种关系。数组中的每一个元素通过它们在数组中的下标来访问。每个下标用方括号括起来。下面的语句表示为数组中的一个元素赋值。

在 awk 中不必指明数组的大小，只需要为数组指定标示符。向数组元素赋值比较容易。例如，下面的例子中为数组 hr 的一个元素指定了一个字符串“laven”。

这个数组元素的下标是“1”，下面的语句将打印字符串“laven”。

当然，我们可以用循环向数组中写入或取出元素。例如，如果数组 hr 有 10 个元素，可以使用下面的循环来打印每个元素：

我们来看一个例子，依然使用第5.1节中的输入文件，让 employee.txt 文件反序输出，

5.7 表达式

5.8 函数

5.9 总结

第 6 章　sed

Sed 是一个超级流式编辑器。picture a stream flowing through a pipe. Okay, you can't see a stream if it's inside a pipe. That's what I get for attempting a flowing analogy. You want literature, read James Joyce.

Anyhow, sed is a marvelous utility. Unfortunately, most people never learn its real power. The language is very simple, but the documentation is terrible. The Solaris on-line manual pages for sed are five pages long, and two of those pages describe the 34 different errors you can get. A program that spends as much space documenting the errors than it does documenting the language has a serious learning curve.

Do not fret! It is not your fault you don't understand sed. I will cover sed completely. But I will describe the features in the order that I learned them. I didn't learn everything at once. You don't need to either.

sed 的工作流程：

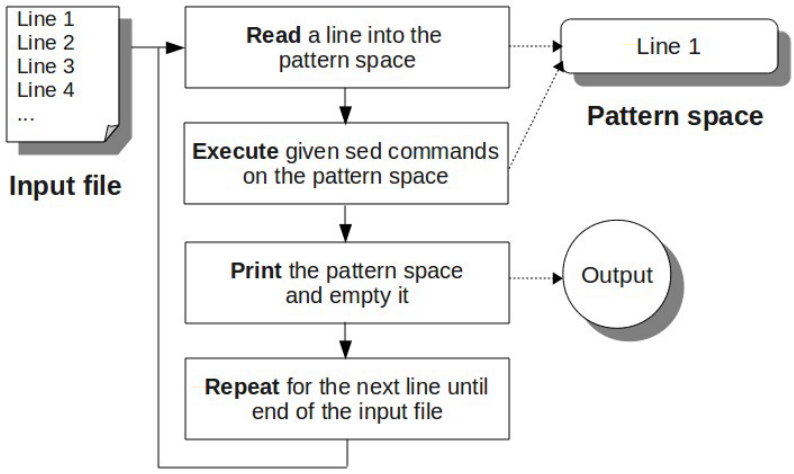


图 6.1　sed 工作流程

命令列表：

命令	说明
a	在当前行后面加入一行或者文本
b label	分支到脚本中带有标号的地方, 如果标号不存在就分支到脚本的末尾
c	用新的文本改变或者替代本行的文本
d	从模板块 (Pattern space) 位置删除行
D	删除模板块的第一行
i	在当前行上面插入文本
h	拷贝模板块的内容到内存中的缓冲区
H	追加模板块的内容到内存中的缓冲区
g	获得内存缓冲区的内容, 并替代当前模板块中的文本
G	获得内存缓冲区的内容, 并追加到当前模板块文本的后面
l	列表不能打印字符的清单
n	读取下一个输入行, 用下一个命令处理新的行而不是用第一个命令
N	追加下一个输入行到模板块后面并在二者之间嵌入一个新的行, 改变当前行的号码
p	打印模板块的行
P	打印模板块的第一行
q	退出 sed
r file	从 file 中读行
t label	if 分支, 从最后一行开始, 条件一旦被满足或者 T 命令或者 t 命令, 将导致分支到带有标号的命令处, 或者到脚本的末尾
T label	错误分支, 从最后一行开始, 一旦发生错误或者 T 命令或者 t 命令, 将导致分支到带有标号的命令处, 或者到脚本的末尾
w file	写并追模板块到 file 末尾
W file	写并追模板块的第一行到 file 末尾
!	表示后面的命令对所有没有被选定的行发生作用
s/re/string/	用 string 替换正则表达式 re
=	打印当前行号码
注释 command	把注释扩展到下一个换行符以前替换标记
g	行内全面替换
p	打印行
w	把行写入一个文件
x	互换模板块中的文本和缓冲区中的文本
y	把一个字符翻译为另外的字符 (但是不能用于正则表达式)

一些选项：

-e command	允许多点编辑
–expression=command	同上
-h,–help	打印命令行选项摘要, 并显示 bug 列表的地址
-n,–quiet,–silent	取消默认输出
-f,	引导 sed 脚本文件名
–filr=script-file	同上
-V,–version	打印版本和版权信息

6.1 匹配

6.1.1 变量定义

6.2 特殊变量

6.3 数组

6.4 删除

6.5 替换

Sed 有众多命令，但经常被用到的就是最基本的替换命令“s”了，替换命令有四部分组成，如下表所示：

s	所使用的命令是替换命令
/../../	所使用的分隔符，可以自行改变，不过三个符号要一致
old	这是将要被替换的字符串或正则表达式
new	这是替换后的字符串

sed s/day/night/ <old >new

eg: sed 's/local/host/[g]' /etc/hosts

Or another way (for Unix beginners),

sed s/day/night/ old >new

我们也可以如下这样写，

将会输出“night”。

我们在这里并没有使用引号把这些参数给引用起来，也能得出想要的结果，因为这个例子是不需要它们的，嘿嘿！然而，在使用过程中出现了原字符，这时候引号是必须要的。如果我们不确定何时要使用引号，那么，最好的习惯就是“每用必带”引号即可，不用纠结那么多。

sed 's/day/night/' <old >new

I must emphasize that the sed editor changes exactly what you tell it to. So if you executed

This would output the word "Sunnight" bacause sed found the string "day" in the input.

The search pattern is on the left hand side and the replacement string is on the right hand side.

We've covered quoting and regular expressions. That's 90% of the effort needed to learn the substitute command. To put it another way, you already know how to handle 90% of the most frequent uses of sed. There are a ... few fine points that an future sed expert should know about. (You just finished section 1. There's only 63 more sections to cover. :-) Oh. And you may want to bookmark this page, ... just in case you don't finish.

6.6 追加、插入和更改

6.7 模式空间和保留空间

6.8 流控制

6.9 地址范围

6.10 调用外部变量

6.11 总结

第 7 章　Bash 脚本

无聊地重复一件事确实惹人厌！当然了，数次重复一些命令是不是也是很不爽的一件事呢？如果是请跟随我往下看。当我们熟悉了操作系统后，以期望有更好的提高，那就是写脚本了。

7.1　语法介绍

7.1.1　变量定义

7.1.2　特殊变量

在脚本里有以下几个特殊的变量，如下

$0	代表该脚本的名字
$n	代表该程序的第 n 个参数值，n=1..9
$*	代表该脚本的所有参数
$#	代表该脚本的参数个数
$$	代表该脚本的 PID
$!	代表上一个指令的 PID
$?	代表上一个指令的返回值

7.1.3　变量赋值和替换

7.1.4　本地变量与全局变量

7.1.5　引用变量

7.1.6　数组

Bash 只支持一维数组，但参数个数没有限制。

数组赋值：

数组的元素的长度：
指的是数组的第个元素的长度，与
等价
是第个元素，

数组的元素个数：

一个例子，随机生成个整型元素的的数组，并找出该数组中的最大值。

7.1.7 特殊字符

7.2 基本流程

Bash 与其他编程语言一样，也有自己的程序处理逻辑。接下来的这个章节主要介绍 Bash 的脚本几种基本流程。

7.2.1 if 结构

先看一下 bash 的 man page 是如何定义 if 结构的，

当 *if list* 执行成功并且返回状态是 0 时，相应的 *then list* 就会被执行；否则，*elif list* 执行，且返回状态为 0 时，相应的 *then list* 就会被执行；之后命令执行结束。如果前面的 *if list* 及 *elif list* 都不能成功执行，那么将执行最后一个 *else list* 语句。返回状态就是上一条命令执行成功与否，执行成功就返回 0，不成功返回非 0。不成功的原因有很多，成功返回就一个。

一个示例：

```
#!/bin/bash
#: Title          : xxxx
#: Date           : 2012-10-12
#: Author         : "Laven Liu" ldczz2008@sina.com
#: Version        : 1.0
#: Description    : get the maximum number
```

```
#: Options                    : None

NUM=1

if [ $NUM -gt 0 ]; then
echo "NUM is great than 0"
else
echo "NUM is less than 0"
fi
```

7.2.2 for 结构

```
#!/bin/sh

# numberlines - A simple alternative to cat -n, etc.

for filename
do
    linecount="1"
    while read line
    do
        echo "${linecount}: $line"
        linecount="$(($linecount + 1))"
    done < $filename
done
exit 0
```

7.2.3 while 结构

```
#!/bin/bash

# Written By: LavenLiu
# Date: 2014-02-13
# Mail:  air.man.six@gmail.com
```

```
# server_list array saved all the servers' ip addresses
server_list=(90 110 225 231 233 235 249 251 252)

# how_many is the quantity of the servers
how_many=$(( ${#server_list[@]} - 1 ))

myloop=0

while [ ${myloop} -lt ${how_many} ] ;
do
ping -c1 192.168.18.${server_list[${myloop}]} &> /dev/null
if [ "$?" -eq 0 ] ; then
echo "Server 192.168.18.${server_list[${myloop}]} is alive"
else
echo "Server 18.${server_list[${myloop}]} is down! Contact the adminstrator
fi
let "myloop = ${myloop} + 1"
done
```

7.3 操作字符串

7.4 函数

7.5 信号捕捉

可以捕捉信号，根据捕捉到的信号，执行响应的操作。

语法：

7.6 开机脚本启动顺序

7.7 一个实例

```
#!/bin/bash
# Written By: LavenLiu
# Date: 2013-01-08
def_colors () {
  # Attributes
  normal='\033[0m'; bold='\033[1m'; dim='\033[2m'; under='\033[4m'
  italic='033[3m'; notalic='\033[23m'; blink='\033[5m';
  reverse='\033[7m'; conceal='\033[8m'; nobold='\033[22m';
  nounder='\033[24m'; noblink='\033[25m'

  # Foreground
  black='\033[30m'; red='\033[31m'; green='\033[32m'; yellow='\033[3
  blue='\033[34m'; magenta='\033[35m'; cyan='\033[36m'; white='\033

  # Background
  bblack='\033[40m'; bred='\033[41m'
  bgreen='\033[42m'; byellow='\033[43m'
  bblue='\033[44m'; bmagenta='\033[45m'
  bcyan='\033[46m'; bwhite='\033[47m'
}

def_colors
clear
hostname=`cat /proc/sys/kernel/hostname`
echo
echo -e "System Report for $white$hostname$normal on `date`"
echo
processor=`grep 'model name' /proc/cpuinfo | cut -d: -f2 | cut -c2-
nisdomain=`cat /proc/sys/kernel/domainname`
cache=`grep 'cache size' /proc/cpuinfo | awk '{print $4,$5}'`
bogomips=`grep 'bogomips' /proc/cpuinfo | awk '{print $3}'`
vendor=`grep 'vendor_id' /proc/cpuinfo`
echo -e "Hostname: $white$hostname$normal NIS Domain: $white$nisdom
```

```
if [ "`echo $vedner | grep -i intel`" ]
then
    cpu_color=$blue
elif [ "`echo $vender | grep -i amd`" ]
then
    cpu_color=$green
fi

echo -e "Processor: $cpu_color$processor$normal"
echo -e "Running at $white$bogomips$normal bogomips with \
$white$cache$normal cache"
echo
osrelease=`cat /proc/sys/kernel/osrelease`
rev=`cat /proc/sys/kernel/version | awk '{print $1}'`
da_date=`cat /proc/sys/kernel/version | cut -d\  -f2-`
upsec=`awk '{print $1}' /proc/uptime`
uptime=`echo "scale=2;$upsec/86400" | bc`
echo -e "OS Type: $white$ostype$normal Kernel: \
$white$osrelease$normal"
echo -e "Kernel Compile $white$rev$normal on \
$white$da_date$normal"
echo -e "Uptime: $magenta$uptime$normal days"
set `grep MemTotal /proc/meminfo`
tot_mem=$2; tot_mem_unit=$3
set 'grep MemFree /proc/meminfo'
free_mem=$2; fre_mem_unit=$3
perc_mem_used=$((100-(100*free_mem/tot_mem)))
set `grep SwapTotal /proc/meminfo`
tot_swap=$2; tot_swap_unit=$3
perc_swap_used=$((100-(100*free_swap/tot_swap)))
if [ $perc_mem_used -lt 80 ]
then
    mem_color=$green
```

```
elif [ $perc_mem_used -ge 80 -a $perc_mem_used -lt 90 ]
then
    mem_color=$yellow
else
    mem_color=$red
fi
if [ $perc_swap_used -lt 80 ]
then
    swap_color=$green
elif [ $perc_swap_used -ge 80 -a $perc_swap_used -lt 90 ]
then
    swap_color=$yellow
else
    swap_color=$red
fi

echo -e "Memory: $white$tot_mem$normal $tot_mem_unit Free: $white$fr
$fre_mem_unit $Used: $mem_color$perc_mem_used$normal"
echo -e "Swap: $white$tot_swap$normal $tot_swap_unit Free: $white$fr
$fre_swap_unit $Used: $swap_color$perc_swap_used$normal"
echo
set `cat /proc/loadavg`
one_min=$1
five_min=$2
fifteen_min=$3
echo -n "Load Avderage:"
for ave in $one_min $five_min $fifteen_min
do
    int_ave=`echo $ave | cut -d. -f1`
    if [ $int_ave -lt 1 ] ; then
        echo -en "$green$ave$normal "
    elif [ $int_ave -ge 1 -a $int_ave -lt 5 ] ; then
        echo -en "$yellow$ave$normal "
```

```
    else
        echo -en "$red$ave$normal"
    fi
done
echo
running=0; sleeping=0; stopped=0; zombie=0
for pid in /proc/[1-9]*
do
    procs=$((procs+1))
    stat=`awk '{print $3}' $pid/stat`
    case $stat in
    R) running=$((running+1));;
    S) sleeping=$((sleeping+1));;
    T) stopped=$((stopped+1));;
    Z) zombie=$((zombie+1));;
    esac
done
echo -n "Process Count: "
echo -e "$white$process$normal total $white$running$normal running \
$white$sleep$normal sleeping $white$stopped$normal stopped \
$white$zombie$normal zombie"
echo
```

下面是该脚本的输出结果:

第 8 章　Makefile 文件

一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，makefile 定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为 makefile 就像一个 Shell 脚本一样，其中也可以执行操作系统的命令。

8.1　定义和概述

Linux 环境下的程序员如果不会使用 GNU make 来构建和管理自己的工程，应该不能算是一个合格的专业程序员，至少不能称得上是 Unix 程序员。在 Linux（unix）环境下使用 GNU 的 make 工具能够比较容易的构建一个属于你自己的工程，整个工程的编译只需要一个命令就可以完成编译、连接以至于最后的执行。不过这需要我们投入一些时间去完成一个或者多个称之为 Makefile 文件的编写。

所要完成的 Makefile 文件描述了整个工程的编译、连接等规则。其中包括：工程中的哪些源文件需要编译以及如何编译、需要创建那些库文件以及如何创建这些库文件、如何最后产生我们想要的可执行文件。尽管看起来可能是很复杂的事情，但是为工程编写 Makefile 的好处是能够使用一行命令来完成“自动化编译”，一旦提供一个（通常对于一个工程来说会是多个）正确的 Makefile。编译整个工程你所要做的唯一的一件事就是在 shell 提示符下输入 make 命令。整个工程完全自动编译，极大提高了效率。

make 是一个命令工具，它解释 Makefile 中的指令（应该说是规则）。在 Makefile 文件中描述了整个工程所有文件的编译顺序、编译规则。Makefile 有自己的书写格式、关键字、函数。像 C 语言有自己的格式、关键字和函数一样。而且在 Makefile 中可以使用系统 shell 所提供的任何命令来完成想要的工作。Makefile（在其它的系统上可能是另外的文件名）在绝大多数的 IDE 开发环境中都在使用，已经成为一种工程的编译方法。

8.2 自动化编译

makefile 带来的好处就是-“自动化编译”，一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大的提高了软件开发的效率。make 是一个命令工具，是一个解释 makefile 中指令的命令工具，一般来说，大多数的 IDE 都有这个命令，比如：Delphi 的 make，Visual C++ 的 nmake，Linux 下 GNU 的 make。可见，makefile 都成为了一种在工程方面的编译方法。

8.3 主要功能

Make 工具最主要也是最基本的功能就是通过 makefile 文件来描述源程序之间的相互关系并自动维护编译工作。而 makefile 文件需要按照某种语法进行编写，文件中需要说明如何编译各个源文件并连接生成可执行文件，并要求定义源文件之间的依赖关系。makefile 文件是许多编译器 –包括 Windows NT 下的编译器 –维护编译信息的常用方法，只是在集成开发环境中，用户通过友好的界面修改 makefile 文件而已。

在 UNIX 系统中，习惯使用 Makefile 作为 makefile 文件。如果要使用其他文件作为 makefile，则可利用类似下面的 make 命令选项指定 makefile 文件：

8.3.1 make 命令

8.3.2 Makefile 的规则

8.3.3 文件定义与命令

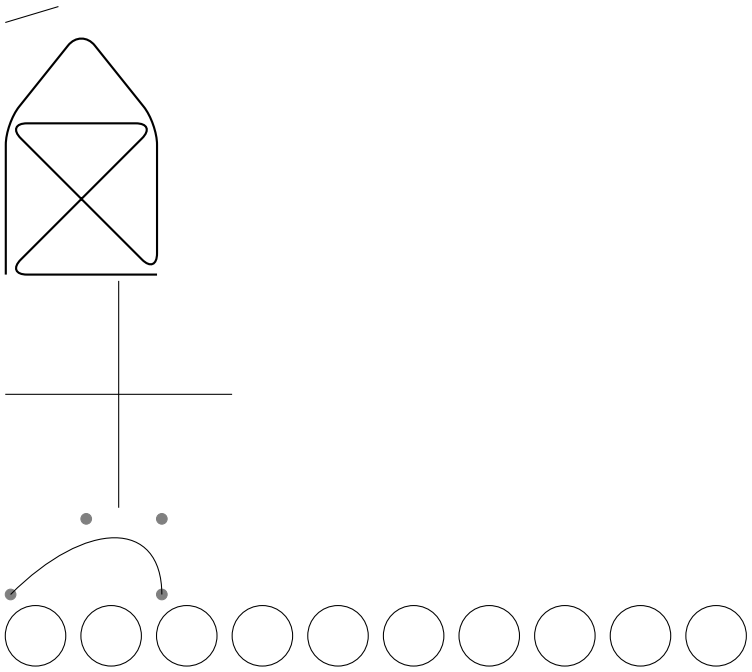
8.3.4 有效的宏引用

8.3.5 预定义变量

8.4 给个实例看看

第 9 章　git

9.1　git 简介



hello

unsigned integer

hello

hello

9.2　安装 git

9.3 创建版本库

9.4 本地仓库

什么是版本库呢？版本库又名仓库，英文名 repository，你可以简单理解成一个目录，这个目录里面的所有文件都可以被 Git 管理起来，每个文件的修改、删除，Git 都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

所以，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：

pwd 命令用于显示当前目录。在我的 Mac 上，这个仓库位于/Users/michael/learngit。

如果你使用 Windows 系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

第二步，通过 git init 命令把这个目录变成 Git 可以管理的仓库：

瞬间 Git 就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository），细心的读者可以发现当前目录下多了一个.git 的目录，这个目录是 Git 来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把 Git 仓库给破坏了。

首先这里再明确一下，所有的版本控制系统，其实只能跟踪文本文件的改动，比如 TXT 文件，网页，所有的程序代码等等，Git 也不例外。版本控制系统可以告诉你每次的改动，比如在第 5 行加了一个单词“Linux”，在第 8 行删了一个单词“Windows”。而图片、视频这些二进制文件，虽然也

能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从 100KB 改成了 120KB，但到底改了啥，版本控制系统不知道，也没法知道。

不幸的是，Microsoft 的 Word 格式是二进制格式，因此，版本控制系统是没法跟踪 Word 文件的改动的，前面我们举的例子只是为了演示，如果要真正使用版本控制系统，就要以纯文本方式编写文件。

因为文本是有编码的，比如中文有常用的 GBK 编码，日文有 Shift_JIS 编码，如果没有历史遗留问题，强烈建议使用标准的 UTF-8 编码，所有语言使用同一种编码，既没有冲突，又被所有平台所支持。

言归正传，现在我们编写一个 readme.txt 文件，内容如下：

一定要放到 learngit 目录下（子目录也行），因为这是一个 Git 仓库，放到其他地方 Git 再厉害也找不到这个文件。

和把大象放到冰箱需要 3 步相比，把一个文件放到 Git 仓库只需要两步。

第一步，用命令 git add 告诉 Git，把文件添加到仓库：

执行上面的命令，没有任何显示，这就对了，Unix 的哲学是“没有消息就是好消息”，说明添加成功。

第二步，用命令 git commit 告诉 Git，把文件提交到仓库：

简单解释一下 git commit 命令，-m 后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

嫌麻烦不想输入 -m ”xxx” 行不行？确实有办法可以这么干，但是强烈不建议你这么干，因为输入说明对自己对别人阅读都很重要。实在不想输入说明的童鞋请自行 Google，我不告诉你这个参数。

git commit 命令执行成功后会告诉你，1 个文件被改动（我们新添加的 readme.txt 文件），插入了两行内容（readme.txt 有两行内容）。

为什么 Git 添加文件需要 add，commit 一共两步呢？因为 commit 可以一次提交很多文件，所以你可以多次 add 不同的文件，比如：

第 II 部分

数据库篇

第 10 章　MySQL 数据库基本知识

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 公司。MySQL 是一个命途多舛的产品，2008 年 2 月 26 日被 Sun 完全收购，2009 年 4 月 20 日 Oracle 并购了 Sun 公司。业界一片哗然，一时众说纷纭，褒贬不一。在此过程中衍生出了两个版本的数据库，一个是 Percona Server 及 MariaDB，这些版本的衍生，考虑到 Oracle 将其闭源的潜在可能。

关系型数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策（本词条“授权政策”），它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。由于其社区版的性能卓越，搭配 PHP 和 Apache 可组成良好的开发环境。

10.1　存储设备配置

10.1.1　构建文件系统

使用 xfs 文件系统来提供 MySQL 数据的存储。使用 xfs 的话，需要添加管理配置工具包，如下：

使用 mkfs.xfs 创建文件系统如下：

注：这里的 sdb 是我们使用的一个样例。不同 SSD 厂商的设备在系统里被识别的标识不一定是/dev/sdb，根据实际情况选择合适的标示符（如，Fusion-io 的设备在本次测试中被识别为/dev/fioa；LSI 的设备在本次测试中被识别为/dev/sdb；Intel 的设备在本次测试中被识别为/dev/nvme0n1；宝存的设备在本次测试中被识别为/def/dfa）。

10.1.2 挂载文件系统

在/etc/fstab 中添加自动挂载参数如下：

10.2 安装 MySQL

10.2.1 创建 MySQL 用户

我们需要添加专门的 MySQL 用户，并设置它的 uid 及 gid 为 601。

10.2.2 安装 MySQL

采用二进制方案安装，步骤如下：

1. 首先获取与生产环境数据库相同的二进制安装文件

2. 解压

3. 创建数据目录

4. 初始化 MySQL 元数据

5. 链接到/usr/local/mysql

10.3 操作系统配置

10.3.1 进程文件数配置

由于 MySQL 采用多线程形式，每个进程打开的文件数要求比较高。所以我们需要增加 ulimit 进程文件数的限制。在/etc/rc.local 中添加:

把上述配置写到配置文件/etc/security/limits.conf，找到“# End of file”，在其上添加 3 行

10.3.2 sysctl 配置

修改 sysctl，添加 swapness 并设置 TIME_WAIT 的相关参数。避免内存被交换并且减少 TIME_WAIT 状态时间。

10.3.3 cgroup 配置

我们只对 cpu 和内存做限制，所以只需要挂载两个子系统。另外，我们在本服务器上计划启动 4 个实例，这四个实例分别属于四个组，mysql_g1, mysql_g2, mysql_g3, mysql_g4。目前我们限制它们都在 4 个超线程并且每个实例的内存不超过 24G。如下：

10.3.4 cgrule 配置

我们设置使用 mysqld_safe1 启动的命令为 mysql_g1 组，这样就可以限制它的 CPU 和内存使用。其他的四个实例同样进行设置如下：

10.4 MySQL 配置

10.4.1 程序文件和目录配置

10.4.2 多实例配置

1. my.cnf 文件配置
   我们在/home/mysql/conf 下保存着 my1.cnf，my2.cnf，my3.cnf，my4.cnf。针对每一个数据库实例都有一个对应的配置文件。各个配置文件中都保存的是每个实例独立的配置。通过“实例切换配置”来切换到各个不同的实例。
2. 实例切换配置
   我们在/home/mysql/bin 下存放有四个文件 my1，my2，my3，my4。用于切换到各个实例的配置环境中。并且我们在 profile 中添加了对应的别名：

   如，my1 文件内容如下：

直接输入命令 my1 就可以切换到 mysql 实例 1 的环境下。此时可以直接输入 my1 命令登录第一个实例的 mysql 环境，输入 stopmysql 就可以停掉 MySQL 数据库，输入 startmysql 就可以启动 MySQL 数据库。

另外，还做了一些简单方便的命令，以便管理：

cdd 用于切换到实例的数据目录 cdm 用于切换到 MySQL 根目录
cdb 用于切换到 MySQL 的程序文件目录 psm 用于输出 MySQL 数据库的进程和 cgroup 信息等

10.5 数据库日常管理

10.5.1 实例切换

10.5.2 MySQL 启动

10.5.3 MySQL 关闭

10.6 如何进入 MySQL 数据库

进入 MySQL 数据库很简单，首先确保 mysqld 进程已经启动。如果没有启动，请参照前面的命令启动之。只需要在命令行输入命令 mysql 回车即可，因为我们的 IPTV 系统里的 mysql 的 root 账户是没有密码的。

查看我们有哪些库

稍后我们将导出这个库

选择要使用的库
提示数据库已改变

查看库中有哪些表

退出数据库

10.7 如何建库建表

当我们的数据很少时，把这些信息可以手工的记录在纸片上或其他介质上，管理起来也是很方便的。现如今是信息的时代，我们总不能把数据还是记录在纸上或龟壳上吧，管理起来非把人给累死。把数据存在易于管理的数据库系统中是明智之举。

10.8 简单 sql 语句的使用

有了上面的介绍是不是想动手操练一番。首先确保你的系统中应该安装有 mysql 数据软件；其次，要确保 mysqld 服务已经在运行；最后，要有数据库的用户名及密码。这里我们使用 Clear 的用户名及密码为 123456 为演示环境。

10.8.1 create 语句的使用

create 可以创建库和表，首先我们以一个具体的例子来演示该命令如何使用。接下来我们要创建这样一个库，库名为 cc（clear company），表的名字为 hr（human resource）

id	name	deptname	salary
1	james	development	1000
2	wade	engineer	800
3	bosh	finance	600
4	richard	sale	600
5	laven	it	1010

上面的具体含义，我就不详细讲解了，自己去 Google 上 Baidu 一下吧。上面只是把一个框架给搭建了起来，里面并没有任何数据，我们可以查看一下：

你将会看到这样的提示

10.8.2 insert 语句的使用

insert 就是往表里面添加数据的，废话不多说，直接看例子：

10.8.3 select 语句的使用

select 应该是数据库中使用最多的命令了，接下来赶紧介绍如何写 select 语句。

* 是通配符，表示所有。上面的语句意思是查看 maincontrol 表的所有字段。如果只查看某个字段或某几个字段，该怎么办呢？各字段之间用逗号分开即可。每个表有哪些字段呢？使用 desc 查看即可。下面看演示：

10.8.4 update 语句的使用

当有更新的操作时，update 命令就派上用场了。当我们忘记信息发布或 IPTV 后台的密码时，也是可用用到 update 语句的，接下来我们看看如何使

用。

找到
查看有哪些字段

其他列我省略了

这是用户名
这是密码

此处省略若干行

这时，我们看到了后台用户及其密码。密码是使用 md5 方式加密的，不安全。我们可以在数据库里查看某个字符串的 md5 值，如：

发现没有，123456 的 md5 值跟上面 master 用户密码的 md5 值一样，那说明 maser 用户的密码为 123456。即使我们不知道 master 用户的密码，我们照样可以在数据库里更新它的密码。这里我们把它的密码改为“654321”，看操作：

用鼠标左键选择“c33367701511b4f6020ec61ded352059”，别把双引号也给选上了。之后按下鼠标中键即可实现粘贴功能

或者一步到位：

再次查询，验证是不是已经改变，这时可以使用新的密码登陆后台了。

10.9 数据库文件的备份

做完一个项目后的第一件事，就是做好现场的备份，包括前台、后台及数据库。首先，这是一个很好的习惯。如果不这样，万一出现问题，又要重新部署，岂不是 x 都碎了。

10.9.1 数据库文件的导出

数据库文件的导出，我们这里使用 mysqldump 命令。别的不多说，直接介绍该命令如何使用

上面这是标准的数据库备份命令，root 为数据的用户名，password 为 root 用户的密码，需要导出的数据库名字为 db_name，然后把数据保存到当前目录下的 db_name.sql 文件。在我们的 IPTV 系统里，mysql 的 root 账户是没有密码的，如果我们要备份 clear_360fy 的库文件，则输入一下命令即可：

这条命令意思是把的库到目录下的文件

10.9.2 数据库文件的导入

数据库文件导入的前提是准备要导入的 sql 文件，这里我们准备好 clear_360fy.sql 文件，然后创建相应的库，再然后把 sql 文件导入到刚创建的库中，这里只介绍一种方法，仅供参考。

这条命令意思是先创建 clear_360fy 库，然后把/home 目录下的 clear_360fy.sql 文件导入到该库中

10.10 常用命令总结

1. 刷新

2. 显示所有数据库

3. 打开数据库

4. 显示当前数据中所有的表

5. 查看表结构

6. 删除数据库

7. 删除表

8. 创建数据库

9. 查询
10. 查看用户权限

查询
列名称表名称

查看用户权限

查看进程

查看所有用户

查看主从状态

查看所有参数变量

查看表的引擎状态

表存在就删除

表不存在就创建

查询用户权限先

创建表

查看系统的字符集和排序方式的设定

查看超时

删除空用户

删除用户

改变现有的表使用的存储引擎

查询表引擎
库名表名

创建表指定存储引擎的类型或

创建主从复制用户
用户密码

添加索引

插入字段
列名字段不为空

更新数据

清除二进制文件

第 11 章　MySQL 复制

主服务器/从服务器设置增加了健壮性。主服务器出现问题时，可以切换到从服务器。

通过在主服务器和从服务器之间切分处理客户查询的负荷，可以得到更好的客户响应时间。SELECT 查询可以发送到从服务器以降低主服务器的查询处理负荷。但修改数据的语句仍然应发送到主服务器，以便主服务器和从服务器保持同步。

MySQL 提供了数据库的同步功能，这对实现数据库的冗灾、备份、恢复、负载均衡等都是有极大帮助的。

11.1　复制如何工作

MySQL 复制数据，总的来说，有三个步骤：

1. 在主库上把数据更改记录到二进制日志（Binary Log）中 (这些记录被称为二进制日志事件)。
2. 备库将主库上的日志复制到自己的中继日志（Relay Log）中。
3. 备库读取中继日志中的事件，将其重放到备库数据之上。

复制如何工作：

第一步是在主库上记录二进制日志。在每次准备提交事务完成数据更新前，主库将数据更新的事件记录到二进制日志中。MySQL 会按事务提交的顺序而非每条语句的执行顺序来记录二进制日志。在记录二进制日志后，主库会告诉存储引擎可以提交事务了。

下一步，备库将主库的二进制日志复制到其本地的中继日志中。首先，备库会启动一个工作线程，称为 I/O 线程，I/O 线程跟主库建立一个普通的客户端连接，然后在主库上启动一个特殊的二进制转储（binlog dump）线程 (该线程没有对应的 SQL 命令），这个二进制转储线程会读取主库上二进制

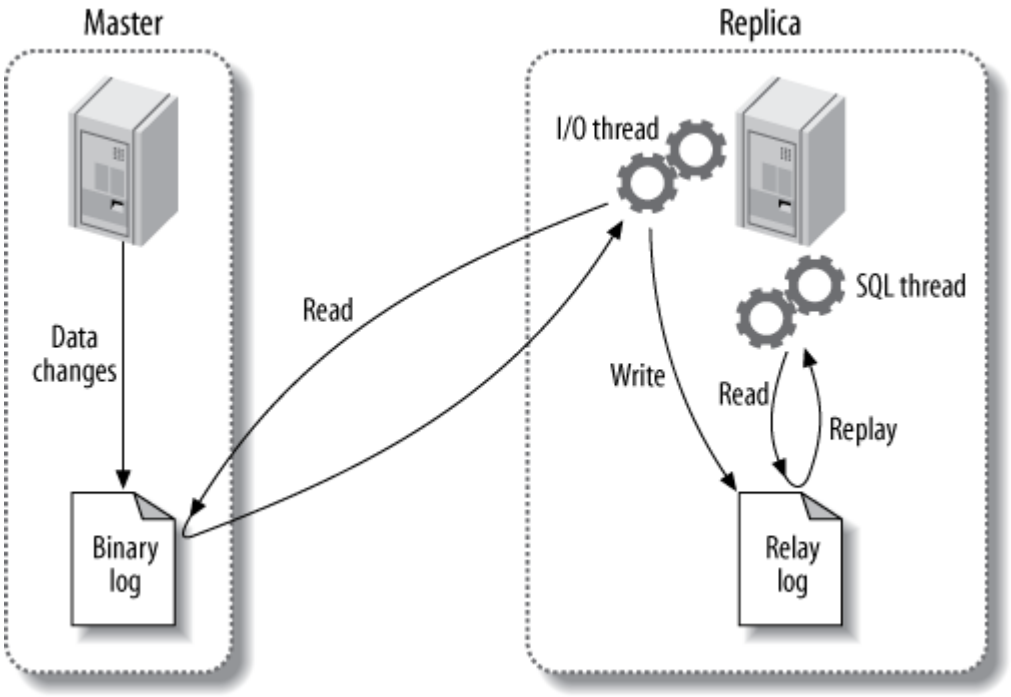


图 11.1　MySQL 复制的工作原理

日志中的事件。它不会对事件进行轮询。如果该线程追赶上了主库，它将进入睡眠状态，直到主库发送信号量通知其有新的事件产生时才会被唤醒，备库 I/O 线程会将接收到的事件记录到中继日志中。

备库的 SQL 线程执行最后一步，该线程从中继日志中读取事件并在备库执行，从而实现备库数据的更新。当 SQL 线程追赶上 I/O 线程时，中继日志通常已经在系统缓存中了，所以中继日志的开销很低。SQL 线程执行的事件也可以通过配置选项来决定是否写入其自己的二进制日志中。

这种复制架构实现了获取事件的重放事件的解耦，允许这两个过程异步进行。也就是说 I/O 线程能够独立于 SQL 线程之外工作。但这种架构也限制了复制的过程，其中最重要的一点是在主库上并发运行的查询在备库上只能串化执行，因为只有一个 SQL 线程来重放中继日志的事件。

11.2　MySQL 复制原理

MySQL 支持单向、双向复制、异步复制，复制过程中一个服务器充当主服务器，而一个或多个其他服务器充当从服务器。主服务器将更新写入一

个二进制日志文件中，并创建一个索引文件以跟踪日志循环。这些日志文件可以将记录发送到从服务器以让从服务器保持与主服务器的数据一致。当一个从服务器连接主服务器时，日志文件会通知主服务器，从服务器在上次成功更新的位置处开始进入更新操作。更新完成后从服务器开始进入等待状态，等待主服务器后续的更新。

11.3 MySQL 同步细节

MySQL 同步功能由 3 个线程（Master 上 1 个 binlog dump，Slave 上 2 个，分别是 SQL 进程和 IO 进程）来实现。执行“START SLAVE”语句后，Slave 就创建

11.4 MySQL 半同步配置

1. 检查是否支持动态加载
   登录 MySQL 并查询 have_dynamic_loading 变量

2. 确认 lib/plugin 目录下有半同步插件 so
   确认主库 MySQL 程序目录下有 lib/plugin/semisync_master.so 文件，确认备库 MySQL 程序目录下有 lib/plugin/semisync_slave.so 文件

3. 主库动态加载半同步插件
   登录 MySQL 并加载主库半同步插件，并设置主库半同步有效

4. 备库动态加载半同步插件

5. 主库 MySQL 配置文件修改
   为了保证主库 MySQL 再次重启服务器后自动打开半同步，需要增加主库 MySQL 配置如下：

6. 备库 MySQL 配置文件修改
   为了保证备库 MySQL 再次重启服务器后自动打开半同步，需要增加备库 MySQL 配置如下：

7. 备库 MySQL 重新连接
   为了保证主备库是以半同步方式复制的，建议备库上重启复制：

第 12 章　MySQL+KeepAlived

12.1　KeepAlived 介绍

Keepalived is a routing software written in C. The main goal of this project is to provide simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures. Loadbalancing framework relies on well-known and widely used Linux Virtual Server (IPVS) kernel module providing Layer4 loadbalancing. Keepalived implements a set of checkers to dynamically and adaptively maintain and manage loadbalanced server pool according their health. On the other hand high-availability is achieved by VRRP protocol. VRRP is a fundamental brick for router failover. In addition, Keepalived implements a set of hooks to the VRRP finite state machine providing low-level and high-speed protocol interactions. Keepalived frameworks can be used independently or all together to provide resilient infrastructures.

使用 MySQL 双 master+keepalived 是一种非常好的解决方案，在 MySQL-HA 环境中，MySQL 互为主从关系，这样就保证了两台 MySQL 数据的一致性，然后用 KeepAlived 实现虚拟 IP，通过 KeepAlived 自带的服务监控功能来实现 MySQL 故障时自动切换。

环境：

各个主机授权：

查看各主机的日志文件及日志位置：

各主机：

12.2 测试同步

12.3 安装配置 KeepAlived

安装配置：

12.4 启动 KeepAlived

启动服务，验证：

在其他机器上，测试的连通性：

在其他机器上登陆，进行验证：

停掉的：

故障切换：
为客户端，连接来进行查询，
首先，停掉的服务，验证上的是不是被停掉了。

然后，查看客户端的连接情况。

最后，查看的状态及连接情况。

第 13 章　MySQL+LVS

第 III 部分

基本服务篇

本章介绍几种常见的服务。这里不注重概念的介绍，概念只做简短的描述，主要介绍如何实施及实现这些服务。

第 14 章　DHCP

动态主机配置协议 (DHCP)，

1. 服务端端口 67/UDP
2. 客户端端口 68/UDP
3. 客户端发送 DHCPDISCOVER 在网络中寻求地址分配
4. 服务端回应 DHCPDISCOVER 请求
5. 客户端发送 DHCPREQUEST
6. 服务端发送 DHCPACK
7. 客户端得到地址

主配 cp /usr/share/doc/dhcp-<version>/dhcpd.conf.sample /etc/dhcpd.conf 包含以下 7 项服务器即可搭建

第 15 章　FTP

第 16 章　NFS

NFS（Network File System）网络文件系统，目前依然非常流行。NFS 的一个最大优点是具有广泛的支持：大部分的类 UNIX 都可以支持 NFS。NFS 通常比其他的网络文件系统更容易配置和使用。与其他网络文件系统一样，NFS 可以在服务器或任何一个客户端上修改文件，然后在其他所有系统上可以立即使用修改后的文件。

16.1　配置 NFS 服务器

16.2　配置 NFS 客户端

第 17 章　Kickstart

Red Hat Linux 提供了一种非常方便的自动化系统安装方式，即为 kickstart。这种工具的出现，极大的方便了众多的系统管理员或装机攻城狮。有了它，我们就不用拿着光盘或 U 盘在机房来回乱窜地装机了，我们可以在办公室通过 IPMI 的方式来操作。不管这个工具有多优秀，相比之前单台的安装方式，效率提升了很多。当然，也有另外几种较优越的自动化系统安装工具，这里就不介绍了。

kickstart 文件包含了安装程序所使用的指令，在安装的过程中可以用来减少或者消除用户输入的麻烦。

在系统数目巨大且完全相同的时候，kickstart 文件非常有用。

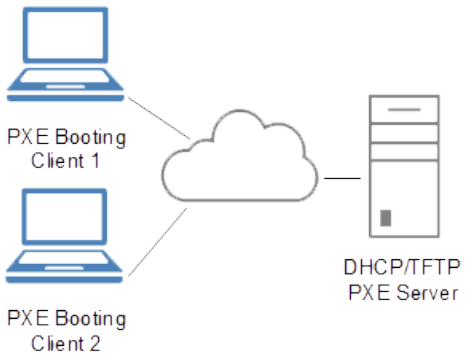


图 17.1　PXE WorkFlow

PXE 工作流程图：

17.1　安装相关软件包

图 17.2 PXE 工作流程图

3. Stop some services, such as selinux, iptables setenforce 0 or vi /etc/sysconfig/selinux disabled service iptables stop

4. Copy the related files to the related path

5. Modify the related configure files

6. Start services, and later the client can install OS

NOTE: In the ks.cfg, the keyword clearpart –none is default change this into: clearpart –all

第 18 章　Samba

第 19 章　LAMP

19.1　安装依赖包

Needless to say, you should install required system packages. So you cancontinue the next steps. :-)

Be sure these packages are installed:

19.2　安装额外的包

In this section, we need to compile and install four packages:

1. GD2

2. LibXML2

3. LibMcrypt

4. OpenSSL

19.3 编译安装 MySQL

19.4 编译安装 Apache

19.5 编译安装 PHP

19.6 安装 Zend 加速器

19.7 整合 Apache 与 PHP

We should modify configure file httpd.conf:

Find this line,

Add on line under this line,

Find these lines,

Change them like this,

Find these lines and uncomment them,

When finished, save it! Then restart apache service,

第 20 章　多网卡绑定 bonding

Linux bonding 提供将多个网络接口设备捆绑为单个网络接口设置来使用，用于网络负载均衡及网络冗余。

以太网平衡设备即使用两个或两个以上的网络接口模拟一个虚拟的网络接口用以外部网络连接。这多个真实网络接口可以连接在同一交换设备或多个交换设备上，以达到多通路高可用性。在所有真实网络接口中可以指定其中某些真实网络接口活跃，其他真实网络接口在活跃的真实网络接口故障时接管网络传输；也可以指定所有真实网络接口活跃，分担全部网络传输带宽。

通道绑定 (Channel bonding) 需要服务器至少拥有 2 个以太网卡，当使用绑定功能的时候，

bonding 模块会使用第一个实际网卡的 MAC 地址来通信，在侦测到这个网卡失败以后，它会把这个 MAC 地址指定到另一块网卡上。

20.1　bonding 的几种模式

bonding 只能提供链路监测，即从主机到交换机的链路是否接通。如果只是交换机对外的链路 down 掉了，而交换机本身并没有故障，那么 bonding 会认为链路没有问题而继续使用。

miimon 是用来进行链路监测的。比如:miimon=100，那么系统每 100ms 监测一次链路连接状态，如果有一条线路不通就转入另一条线路；mode 的值表示工作模式，他共有 0，1，2，3，4，5，6 七种模式，作者遇到的场景是使用的 1，0，6 三种，其他场景适合哪些模式作者也不清楚。

mode=1，表示 fault-tolerance (active-backup) 提供网络冗余功能，工作方式是主备模式，也就是说默认情况下只有一块网卡工作，另一块做备份。

mode=0，表示 load balancing (round-robin) 为负载均衡模式，两块网卡都工作，但是与网卡相连的交换机必须做特殊配置（这两个端口应该采取聚合

方式），因为做 bonding 的这两块网卡是使用同一个 MAC 地址。

mode=6，表示 load balancing (round-robin) 为负载均衡方式，两块网卡都工作，但是该模式下无需配置交换机，因为做 bonding 的这两块网卡是使用不同的 MAC 地址。

20.2 RHEL 下配置 bonding

20.3 SuSE 下配置 bonding

bond0 配置：

br0 配置：

管理网

为网卡打

上联核心交换机端口的号

验证是否设置成功

20.4 单网卡多 IP 配置

第 21 章　LVM 硬盘管理及扩容

LVM 是 Logical Volume Manager(逻辑卷管理) 的简写，它由 Heinz Mauelshagen 在 Linux 2.4 内核上实现。LVM 将一个或多个硬盘的分区在逻辑上集合，相当于一个大硬盘来使用，当硬盘的空间不够使用的时候，可以继续将其它的硬盘的分区加入其中，这样可以实现磁盘空间的动态管理，相对于普通的磁盘分区有很大的灵活性。

与传统的磁盘与分区相比，LVM 为计算机提供了更高层次的磁盘存储。它使系统管理员可以更方便的为应用与用户分配存储空间。在 LVM 管理下的存储卷可以按需要随时改变大小与移除 (可能需对文件系统工具进行升级)。LVM 也允许按用户组对存储卷进行管理，允许管理员用更直观的名称 (如”sales’、’development’) 代替物理磁盘名 (如’sda’、’sdb’) 来标识存储卷。

如图所示 LVM 模型：

由四个磁盘分区可以组成一个很大的空间，然后在这些空间上划分一些逻辑分区，当一个逻辑分区的空间不够用的时候，可以从剩余空间上划分一些空间给空间不够用的分区使用。

第 22 章　使用 U 盘安装 Gnu 系统

使用 U 盘安装系统在某些时候还是很方便的，其安装速度也是蛮给力的。

22.1　准备工作

1. 4G 以上的优盘，格式化成 FAT32
2. 镜像编辑软件 UltraISO
3. Gnu/Linux 镜像文件

22.2　制作启动盘

1. 启动软件后打开 Linux ISO 文件
2. 点击“工具”，“加载到虚拟光驱”，点击“加载”
3. 打开“我的电脑”找到虚拟光驱的盘符
4. 找到“boot.ISO”文件，双击打开
5. 点击“启动”，“写入硬盘镜像”
6. 选择你的 U 盘，写入方式：USB-HDD+，点击“写入”，开始写入引导文件到 U 盘
7. 等待一会儿，消息栏提示“刻录成功”，引导文件写入完毕
8. 确认是否写入成功和盘符是否选对：打开“我的电脑”，可移动存储栏里出现以：“Red Hat Ent”命名的盘符，代表写入成功

22.3　开始安装 Gnu/Linux

在安装过程中，需要选择 Grub 安装位置时，需要注意一下，我们要把 Grub 安装到系统盘中，默认是安装在 U 盘里的，这时可以修改之以继续安装。如果这一步没有修改，也没有关系，等系统安装完毕重启时，不要拔掉

U 盘。一切就绪进入系统后，可以使用 grub-install 命令重新安装 Grub 到指定的盘中。

22.4 安装结束应注意的地方

第 23 章　RAID 技术

RAID 技术有各种级别之分，包括 RAID0、RAID1、RAID2、RAID3、RAID4、RAID5、RAID5E、RAID5EE、RAID6、RAID10 等。作者接触最多的是 RAID0、RAID1、RAID5 及 RAID10 这些 RAID 级别，其他级别在实际工作当中并没有见到，这里就不在介绍。

23.1　RAID 基础知识

RAID 最初是 1987 年在加利福尼亚大学进行的一个科研项目，后来由伯克利分校的 D.A. Patterson 教授在 1988 年正式提出。

RAID 是 Redundant Array of Inexpensive Disks 的缩写，直译为“廉价冗余磁盘阵列”，最初是为了组合多块小容量的廉价磁盘来代替大容量的昂贵磁盘，同时希望在磁盘失效时不会对数据造成影响而开发出的一种磁盘存储技术。

后来随着磁盘研发技术的不断提升，硬盘容量越来越大，成本却不断下降，所以 RAID 中“Inexpensive（廉价）”一词已经失去意义，于是将这个词用“Independent（独立）”来替代，RAID 就成了”独立冗余磁盘阵列“，也简称为”磁盘阵列“，但这只是名称的变化，实质性的内容并没有改变。

23.1.1　RAID 解决了什么问题

通俗地说，RAID 就是通过将多个磁盘按照一定的形式和方案组织起来，通过这样的形式能够获取比单个磁盘更高的速度、更好的稳定性、更大的存储能力的存储解决方案，我们不必关心磁盘阵列究竟由多少块磁盘组成，使用中整个整列就如同一块磁盘一样。所以，RAID 技术能够为计算机系统提供以下三个方面的优异性能：

1. 提供更大的存储空间

使用 RAID 技术，就可以把多块磁盘组成一个更大的存储空间供我们使用。比如，利用 RAID0 技术把 5 块 1TB 的硬盘组织起来，能够提供 5TB 的存储空间。

2. 提供更快的传输速度

著名的摩尔定律告诉我们，CPU 的性能每隔 18 个月就会提高一倍，可见其速度增长之快。然而，机械硬盘作为计算机中最重要的存储设备，在容量飞速增长的同时，速度却提高缓慢，已经成为计算机速度发展的瓶颈。

如果采用 RAID 技术，可以让很多机械硬盘同时传输数据，而这些硬盘在逻辑上又表现为一块硬盘，所以使用 RAID 可以达到单个磁盘几倍、甚至 N 多倍的速率。也就是说，RAID 技术可以通过在多个磁盘上同时存储和读取数据的方式来大幅提高存储系统的数据吞吐量。

3. 提高更高的安全性

RAID 可以通过数据校验提供容错功能，在很多 RAID 模式中都有较为完备的冗余措施，甚至是直接相互的镜像备份，从而大大提高来 RAID 系统的容错性，让系统的稳定性更好、安全性更高。

23.2 RAID 实现方式

23.2.1 RAID0 数据组织原理

RAID0 是无冗余、无校验的磁盘阵列，实现 RAID0，至少需要两个以上硬盘，它将两个以上的硬盘合并成一块，数据同时分散在每块磁盘中，因为带宽加倍，所以读写速度加倍，RAID0 的理论速度是单块硬盘的 N 倍，但是由于数据并不是保存在一个硬盘上，而是分成数据块保存在不同的硬盘上，所以安全性也下降 N 倍，只要任何一块磁盘损坏就会丢失所有数据。

RAID0 是最简单的一种 RAID 形式，目的是把多块物理盘连接在一起形成一个容量更大的存储设备，RAID0 逻辑盘的容量等于物理盘的容量乘以成员盘的数目。

原理图

RAID0 只是单纯地提高读写性能，并没有数据的可靠性提供保证，而且其中的任何一个物理盘失效都将影响到所有数据。因此，RAID0 不能用于

数据安全性要求高的场合。

23.2.2 RAID1 数据组织原理

RAID1 通过磁盘数据镜像实现数据的冗余，在两块磁盘上产生互为备份的数据，当其中一块成员盘出现故障时，系统还可以从另外一块成员盘中读取数据。因此，RAID1 可以提供更好的冗余性。

RAID1 又称为磁盘镜像，需要在两个物理盘共同构建，使用磁盘镜像技术，方法是在工作磁盘之外再加一额外的备份磁盘，两个磁盘所存储的数据完全一样，数据写入工作磁盘的同时亦写入备份磁盘，也就是将一块物理盘的内容完全复制到另一块物理盘上。所以，两块物理盘所构成的 RAID1 阵列，其容量仅等于一块磁盘的容量，其数据分布情况如图所示。

原理图

RAID1 是磁盘阵列中单位成本最高的，但提供来很高的数据安全性和可用性。当一个物理盘失效时，系统可以自动切换到镜像盘上读写，而不需要重组失效的数据。

RAID1 是所有 RAID 等级中实现成本最高的一种，尽管如此，我们还是选择 RAID1 来保存那些关键性的重要数据。

23.2.3 RAID10 数据组织原理

23.2.4 RAID5 数据组织原理

23.3 MegaRAID Cli 工具基本使用

我们都是使用过 LSI 的 Web 界面去配置 RAID，虽听起来很高大上，但整个配置过程是那么的令人蛋疼。配置完成后，我们还要按“Ctrl+Alt+Del”组合键来重启机器，步骤虽有些繁琐，但仍能令人接受。MegaRAID Cli 工具是在命令行模式下操作 RAID 控制器的，它的优点之一就是做完 RAID 之后，可以直接使用并不需要重启操作系统，而且操作简单方便。

写这一节的目的并不是推荐使用该工具，而是熟悉了原来的配置界面，不愿意再学新的东西。若不是同事的提示，还不知有这种很 xx 的工具，使用起来确实很酷，愿意跟大家分享一下使用过程。

要想使用该工具，首先系统上要安装相应的软件包，这里省略安装过程。安装完毕之后，工具默认会安装在/opt/MegaRAID/MegaCli 的目录下。

本次使用该命令行工具的场景：新到了两块 Intel 的 SATA 接口 400GB 的 SSD 硬盘，欲测其性能。原来的服务器上自带了 4 块 600GB 的磁盘并做了 RAID10，这四块磁盘分别占据了第 0、1、2、3 这四个硬盘插槽，在第 4、第 5 个插槽放置了 SSD 盘，操作系统使用的是 SLES 11.2。下面是具体的操作过程，

23.3.1 制作 RAID

1. 查看 RAID 卡的设备号

   说明：上面的输出，表示我们有一个 RAID 卡，因为这些 ID 是一样的。这个卡下面有 6 个盘，这里的 ID 号需要记下来，后面做 RAID 时需要用到。可以把该 RAID 的 ID 号理解为主设备号。
2. 查看 Slot ID 以确认有无错序的情况

   说明：这里的 Slot Number 号需要记下来，后面做 RAID 时需要用到。可以吧该 Slot 的 ID 理解为次设备号。
3. 查看 Foreign 信息

说明：状态显示为“Foreign”的磁盘，说明是新添加进来的或者是未使用的。这两个为“Foreign”状态的正是我们新添加的 SSD 盘。接下来的操作就是清除这些“Foreign”状态的盘。

4. 清除盘的 Foreign 信息

5. 新做 RAID，在 Slot4 和 Slot5 上做 RAID0

说明：如果做 RAID1，只需要把 r0 改为 r1 即可。

23.3.2 删除 RAID

当测试完毕 RAID0 级别的 SSD 盘时，要开始测试 RAID1 级别下 Intel SATA SSD 的性能。所以，之前制作的 RAID0 要被删除了。在删除之前，我们需要知道被删除的 Target Id，然后方可删除该组 RAID。

查看有多少个 RAID 级别，找到我们想要删除的 Target Id，其中“Target Id：n”，n 即为第 n 组 RAID。

删除阵列，

附录：名词解释磁盘缓存策略：

()
()
()
()
()

第 24 章　PCIe SSD

24.1　现有硬盘厂商

第 IV 部分

集群方案篇

本章介绍几种高可用的解决方案。

第 25 章　集群基础知识

25.1　集群概述

集群是一组协同工作的服务实体，用以提供比单一服务实体更具扩展性和可用性的服务平台。

在客户端看来，一个集群就是一个完整不可细分的实体，但事实上一个集群实体是由完成不同任务的服务节点个体所组成的。

集群实体的可扩展性是指，在集群运行中新的服务节点可以动态的加入集群实体从而提升集群实体的综合性能。

集群实体的高可用性是指，集群实体通过其内部的服务节点的冗余使客户端免于 OUT OF SERVICE 错误。简单的说，在集群中同一服务可以由多个服务节点提供，当部分服务节点失效后，其他服务节点可以接管服务。

集群实体地址是指客户端访问集群实体获得服务资源的唯一入口地址。

负载均衡是指集群中的分发设备（服务）将用户的请求任务比较均衡(不是平均）分发到集群实体中的服务节点计算、存储和网络资源中。一般我们将提供负载均衡分发的设备叫做负载均衡器。负载均衡器一般具备如下三个功能：

1. 维护集群地址
2. 负责管理各个服务节点的加入和退出
3. 集群地址向内部服务节点地址的转换

错误恢复是指集群中某个节点或某些服务节点（设备）不能正常工作(或提供服务），其他类似服务节点（设备）可以资源透明和持续的完成原有任务。具备错误恢复能力是集群实体高可用性的必要条件。

负载均衡和错误恢复都需要集群实体中各个服务节点中有执行同一任务的资源存在，而且对于同一任务的各个资源来说，执行任务所学的信息视图必须一致。

25.2 集群类型

多台同构或异构的计算机用某种方式连接起来协同完成特定的任务就构成了集群系统，目前 Linux 下的集群主要有三种类型：

1. HA（High Availability）
2. LB（Load Balancing）
3. HPC（High Performance Computing）

第 26 章　LVS+Keepalived 负载均衡集群

LVS(Linux Virtual Server) is a cluster of servers

The Linux Virtual Server can be used to build scalable network services based on a cluster of two or more nodes. The active node of the cluster redirects service requests to a collection of server hosts that will actually perform the services. Supported features include two protocols (TCP and UDP), three packet-forwarding methods (NAT, tunneling, and direct routing), and eight load balancing algorithms (round robin, weighted round robin, least-connection, weighted least-connection, locality-based least-connection, locality-based least-connection with replication, destination-hashing, and source-hashing).

LVS: ipvsadm/ipvs

When a new connection is requested from a client to a service provided by the LVS (e.g. httpd), the director will choose a realserver from the client.

From then, all packets from the client will go through the director to that particular realserver.

The association between the client and the realserver will last for only the life of the tcp connection (or udp exchange).

For the next tcp connection, the director will choose a new realserver (which may or may not be the same as the first realserver).

Thus a web browser connecting to a LVS serving a webpage consisting of several hits (images, html page), may get each hit from a separate realserver.

LVS IP Address Name Conventions

Virtual IP (VIP) address: The IP address the Director uses to offer services to client computers

Real IP (RIP) address: The IP address used on the cluster nodes

Director's IP (DIP) address: The IP address the Director uses to connect to the D/RIP network

Client computer's IP (CIP) address: The IP address assigned to a client computer that it uses as a source IP address for requests sent to the cluster

Basic Properties of LVS-NAT

1. The cluster nodes need to be on the same network (VLAN or subnet) as the Director. 集群节点必须与调度器在同一个网络中

2. The RIP addresses of the cluster nodes are normally private, non-routable IP addresses used only for intracluster communication. RIP 通常是私有地址，仅用于各集群节点间的通信

3. director 位于 client 与 real server 之间，并负责处理进出的所有通信

4. realserver 必须将网关指向 DIP

5. 支持端口映射

6. real server 可以使用任何操作系统

7. 较大规模应用场景中，director 易成为系统瓶颈

1. 集群节点跟 director 必须在同一物理网络中

2. RIP 可以使用公网地址，实现便捷的远程管理和监控

3. director 仅负责处理入站请求，响应报文则由 real server 直接发往客户端

4. real server 不能讲网关指向 DIP

5. 不支持端口映射

real server 必须能够隐藏 VIP

TUN: 1. 集群节点可以跨越 Internet 2. RIP 必须是公网地址 3. Director 仅处理入站请求，响应报文则由 real server 直接发往客户端 4. real server 网关不能指向 director 5. 只有支持隧道功能的 os 才能用于 real server 6. 不支持端口映射

26.1 LVS 调度算法

Director 在接收到来自于 Client 的请求时，会基于"schedule" 从 RealServer 中选择一个响应给 Client。ipvs 支持以下调度算法：

1. 轮询（round robin, rr), 加权轮询 (Weighted round robin, wrr)
   新的连接请求被轮流分配至各 RealServer；算法的优点是其简洁性，它无需记录当前所有连接的状态，所以它是一种无状态调度。轮叫调度算法假设所有服务器处理性能均相同，不管服务器的当前连接数和响应速度。该算法相对简单，不适用于服务器组中处理性能不一的情况，而且当请求服务时间变化比较大时，轮叫调度算法容易导致服务器间的负载不平衡。
2. 最少连接 (least connected, lc)，加权最少连接 (weighted least connection, wlc)
   新的连接请求将被分配至当前连接数最少的 RealServer；最小连接调度是一种动态调度算法，它通过服务器当前所活跃的连接数来估计服务器的负载情况。调度器需要记录各个服务器已建立连接的数目，当一个请求被调度到某台服务器，其连接数加 1；当连接中止或超时，其连接数减一。
3. 基于局部性的最少链接调度（Locality-Based Least Connections Scheduling，lblc）
   针对请求报文的目标 IP 地址的负载均衡调度，目前主要用于 Cache 集群系统，因为在 Cache 集群中客户请求报文的目标 IP 地址是变化的。这里假设任何后端服务器都可以处理任一请求，算法的设计目标是在服务器的负载基本平衡情况下，将相同目标 IP 地址的请求调度到同一台服务器，来提高各台服务器的访问局部性和主存 Cache 命中率，从而整个集群系统的处理能力。LBLC 调度算法先根据请求的目标 IP 地址找出该目标 IP 地址最近使用的服务器，若该服务器是可用的且没有超载，将请求发送到该服务器；若服务器不存在，或者该服务器超载且有服务器处于其一半的工作负载，则用“最少链接”的原则选出一个可用的服务器，将请求发送到该服务器。
4. 带复制的基于局部性最少链接调度（Locality-Based Least Connections with Replication Scheduling，lblcr）
   也是针对目标 IP 地址的负载均衡，目前主要用于 Cache 集群系统。它与 LBLC 算法的不同之处是它要维护从一个目标 IP 地址到一组服务器的映射，而 LBLC 算法维护从一个目标 IP 地址到一台服务器的映射。对于一个“热门”站点的服务请求，一台 Cache 服务器可能会忙不过来处理这些请求。这时，LBLC 调度算法会从所有的 Cache 服务器中按“最小连接”原则选出一台 Cache 服务器，映射该“热门”站点到这台

Cache 服务器，很快这台 Cache 服务器也会超载，就会重复上述过程选出新的 Cache 服务器。这样，可能会导致该“热门”站点的映像会出现在所有的 Cache 服务器上，降低了 Cache 服务器的使用效率。LBLCR 调度算法将“热门”站点映射到一组 Cache 服务器（服务器集合），当该“热门”站点的请求负载增加时，会增加集合里的 Cache 服务器，来处理不断增长的负载；当该“热门”站点的请求负载降低时，会减少集合里的 Cache 服务器数目。这样，该“热门”站点的映像不太可能出现在所有的 Cache 服务器上，从而提供 Cache 集群系统的使用效率。LBLCR 算法先根据请求的目标 IP 地址找出该目标 IP 地址对应的服务器组；按“最小连接”原则从该服务器组中选出一台服务器，若服务器没有超载，将请求发送到该服务器；若服务器超载；则按“最小连接”原则从整个集群中选出一台服务器，将该服务器加入到服务器组中，将请求发送到该服务器。同时，当该服务器组有一段时间没有被修改，将最忙的服务器从服务器组中删除，以降低复制的程度。

5. 目标地址散列调度（Destination Hashing，dh）
   算法也是针对目标 IP 地址的负载均衡，但它是一种静态映射算法，通过一个散列（Hash）函数将一个目标 IP 地址映射到一台服务器。目标地址散列调度算法先根据请求的目标 IP 地址，作为散列键（Hash Key）从静态分配的散列表找出对应的服务器，若该服务器是可用的且未超载，将请求发送到该服务器，否则返回空。
6. 源地址散列调度（Source Hashing，sh）
   算法正好与目标地址散列调度算法相反，它根据请求的源 IP 地址，作为散列键（Hash Key）从静态分配的散列表找出对应的服务器，若该服务器是可用的且未超载，将请求发送到该服务器，否则返回空。它采用的散列函数与目标地址散列调度算法的相同。除了将请求的目标 IP 地址换成请求的源 IP 地址外，它的算法流程与目标地址散列调度算法的基本相似。在实际应用中，源地址散列调度和目标地址散列调度可以结合使用在防火墙集群中，它们可以保证整个系统的唯一出入口。

基于 DNS 的负载均衡方案性能可能会出现问题。DNS 的记录会缓存。

rsync 基于文件的同步，效率低。

drbd 磁盘镜像，让两个计算机的两块磁盘做镜像，基于块级别的同步，效率高。

第 27 章　Heartbeat 高可用集群

Heartbeat 提供了诸多集群基础架构服务，比如集群之间的消息传递、节点成员身份、IP 地址分配和迁移，以及服务的开启和停止。Heartbeat 可以用来为 Apache、Samba 和 Squid 等企业应用系统构建几乎任何一种高可用性的集群。此外，它可以结合负载均衡软件使用，那样入站请求就可以由所有集群节点来分担。

本文中的示例集群将由 2 台运行 Heartbeat 的服务器组成。我们测试故障切换机制的方法是，手动关闭服务器，检查它们服务的网站是不是仍然可用。下面是我们的测试拓扑结构：

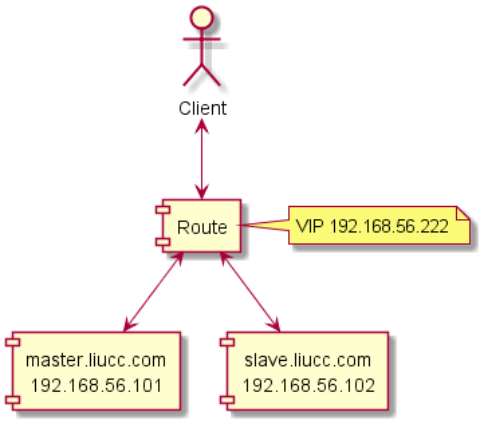


图 27.1　heartbeat 实验拓扑

映射服务所用的 IP 地址需要一直能够访问得到。通常，Heartbeat 会为你将指定的 IP 地址分配给主服务器上的虚拟网络接口卡。如果主服务器出现了故障，集群会自动将 IP 地址切换到另一台可用服务器上的虚拟网卡。如果主服务器恢复正常运行，它会再次将 IP 地址切换回到主服务器。由于具有迁移属性，这个 IP 地址被称为“浮动”地址。

27.1 安装 Heartbeat

在所有服务器上安装软件包

想组建集群，首先要使用 yum，在每一个节点上安装必要的软件包：

下一步，下载和安装官方 CentOS 软件库里面没有的两个 Heartbeat RPM 文件。

另外，你可以将 EPEL 软件库添加到源文件，并使用 yum 进行安装。

Heartbeat 会管理 Apache 的 httpd 服务的开启和停止，所以停止 Apache，并禁止它自动开启：

设置主机名称

现在设置服务器的主机名称，为此编辑每个系统上的/etc/sysconfig/network，并更改 HOSTNAME 这一行：

HOSTNAME=xxxxx.liucc.com 新的主机名称会在服务器下一次启动时激活。你可以使用 hostname 命令立即激活它，不需要重启服务器：

hostname xxxxx.liucc.com 你可以在每一台服务器上运行 uname -n，以此证实主机名称已正确设置好。

27.2 配置 Heartbeat

想配置 Heartbeat，首先要将其默认配置文件从/usr 拷贝到/etc/ha.d/：

然后，你还得改动全部集群节点上的所有三个文件，以便与你的需求相匹配。

authkeys 文件含有集群节点彼此联系时所使用的预共享密码。集群里面的每个 Heartbeat 消息都含有该密码，节点只处理拥有正确密码的那些消息。Heartbeat 支持 SHA1 密码和 MD5 密码。在 authkeys 文件中，下列指令将验证方法设置为 SHA1，并且定义了所使用的密码：

保存该文件，然后使用命令

为该文件授予只有 root 用户可以读写的权限。

下一步，在 ha.cf 文件中，定义计时器、集群节点、消息传递机制、第 4 层端口及其他设置：

日志

计时器
所有计时器设成以秒为单位。如果你需要以毫秒为单位设置时间，就使用 ‘’ 。
间隔时间

超过这个时间后，节点被认为已停滞

一些服务器花更长的时间来启动。该计时器定义了证实服务器宕机之前所等待的额外时间。该计时器的建议时间是停滞计时器的至少一倍。

消息传递参数

你还可以使用多播或单播
节点定义
确保主机名称符合

最后，文件 haresources 含有 Heartbeat 认为是主节点的那台服务器的主机名称，另外还含有浮动 IP 地址。该文件在所有服务器上都一模一样，这点很重要。只要主节点在正常运行，它就服务所有请求；Heartbeat 停止其他所

有节点上的高可用性服务。Heartbeat 检测到该主节点停机运行后，它会在集群中的下一个可用节点上自动开启服务。主节点恢复正常运行后，Heartbeat 会让它再次接手任务，服务所有请求。最后，该文件含有负责高可用性服务的脚本的名称：这里是 httpd。其他可能出现的值有 squid、smb、nmb 或 postfix，映射到通常位于 /etc/init.d/目录中的服务启动脚本的名称。

在 haresources 中，定义 master.liucc.com.com 为主服务器，定义 192.168.56.222 为浮动 IP 地址，定义 httpd 为高可用性服务。你不需要创建任何接口，也不需要为任何接口手动分配浮动 IP 地址，Heartbeat 为我们处理这项任务：

每一台服务器上的配置文件准备就绪后，开启 Heartbeat 服务，并将它添加到系统启动项：

这时，可以查看一下 master 节点上的 IP 地址信息，

你可以借助命令 tailf /var/log/ha-log，密切关注 Heartbeat 日志。

Heartbeat 可用于多项服务。比如说，haresources 中的下列指令将让 Heartbeat 同时管理 Apache 服务和 Samba 服务：

master.liucc.com 192.168.56.222 httpd smb nmb 不过，除非你还在运行 Pacemaker 之类的集群资源管理器（CRM），否则我不建议使用 Heartbeat 在单一集群中提供多项服务。要是没有 Pacemaker，Heartbeat 使用 IP 地址监测第 3 层中的集群节点。只要 IP 地址可以访问得到，Heartbeat 无视服务在服务器节点上可能遇到的任何崩溃或困难。

27.3 测试

一旦 Heartbeat 设置并运行起来，不妨对它测试一下。在所有三台服务器上创建单独的 index.html 文件，那样你就能看清哪台服务器在服务页面。浏览到 192.168.56.222，如果你设置好了 DNS，也可以浏览到相应域名。页面应该会从 master.liucc.com.com 加载，你可以查看服务器 1 中的 Apache 日志文件来核实这一点。试着刷新页面，证实该页面是否每次都从同一台服务器加载。

关闭主节点后，主节点产生的日志信息，

此时，备节点上的日志信息为：

如果这一切进展良好，测试一下故障切换机制：停止 master.liucc.com 上的 Heartbeat 服务。浮动 IP 地址应该会迁移到服务器 slave 上，页面应该会

从该服务器加载。迅速看一下 slave 的 Apache 日志，应该可以证实这一点。如果我们重启了服务器 master 上的 heartbeat 服务，浮动 IP 地址应该会按照 haresources 中的设置，从活动节点迁移到服务器 slave 上。

当我们把 master 上的 heartbeart 重新运行起来，这时，观察 slave 节点的日志信息，

正如我们所见，使用 Heartbeat，在 RHEL 下组建一个高可用性的 Apache 集群是件很容易的事。虽然我们使用了 2 台服务器，但 Heartbeat 在节点数量更多或更少的环境下应该同样没问题。Heartbeat 对节点数量没有任何限制，所以你可以根据需要扩展所设置环境的规模。

第 28 章　pacemaker+corosync 高可用集群

高可用集群，是指以减少服务中断时间为目的的服务器集群技术。高可用集群的出现是为了减少由计算机硬件和软件易错性所带来的损失。它通过保护用户的业务程序对外不间断提供的服务，把因软件/硬件/人为造成的故障对业务的影响降低到最小程度。如果某个节点失效，它的备援节点将在数秒钟的时间内接管它的职责。因此，对于用户而言，集群永远不会停机。高可用集群软件的主要作用就是实现故障检查和业务切换的自动化。

下面案例是 httpd 服务的具体实现：

28.1　准备工作

本案例的拓扑图为：

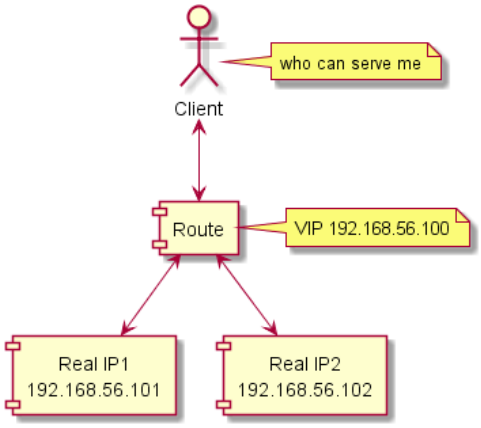


图 28.1　corosync 实验拓扑

为了配置一台主机成为 HA 的节点，通常需要做出如下的准备工作：

1. 所有节点的主机名和对应的 IP 地址解析服务可以正常工作，且每个节点的主机名称需要跟“uname -n”命令的结果保持一致；因此，需要保证两个节点上的/etc/hosts 文件一致；
2. 两个节点可以基于密钥进行 ssh 通信；

具体实现为：

为了使得重新启动系统后仍能保持如上的主机名称，还需要分别在各节点修改 /etc/sysconfig/network 文件。

各节点生成 ssh 密钥：

28.2 安装软件包

首先确保已配置本地 yum 源，然后安装如下依赖包：

安装完毕依赖包，接下来安装 corosync 及 pacemaker，这里已经准备了相应的软件包，

28.3 配置及启动 corosync

下面的配置在 master 上完成，编辑/etc/corosync/corosync.conf

在文件尾部添加如下内容:

并设定此配置文件中 bindnetaddr 后面的 IP 地址为你的网卡所在网络的网络地址，我们这里的两个节点在 192.168.56.0 网络，因此这里将其设定为 192.168.56.0。整个配置文件如下:

版本号
安全认证
线程数，根据个数和核心数确定

冗余环号，节点有多个网卡是可定义对应网卡在一个环内
绑定网络地址
心跳信息所使用的组播地址
心跳组播使用的端口

指定要打印的行
是否发送到标准错误输出
记录到文件
记录到
日志文件

是否打印时间戳，利于定位错误（会消耗资源）

定义在启动时自动启动

表示启动的功能，以哪个用户身份运行

生成节点间通信时用到的认证密钥文件：

建立日志所需要的目录

接下来就可以启动 corosync 了，

此时，可以查看一下关于 corosync 引擎是否正常启动的系统日志，

查看初始化成员节点通知是否正常发出，

检查启动过程中是否有错误产生：

查看 pacemaker 是否正常启动：

如果上面命令执行均没有问题，接着可以执行如下命令启动 node2 上的 corosync

注意：启动 node2 需要在 node1 上使用如上命令进行，不要在 node2 节点上直接启动；

使用如下命令查看集群节点的启动状态：

从上面的信息可以看出两个节点都已经正常启动，并且集群已经处于正常工作状态。

由于 corosync 默认启用了 stonith，而当前集群并没有相应的 stonith 设备，因此此默认配置目前尚不可用，这可以通过如下命令验正：

我们里可以通过如下命令先禁用 stonith：

使用如下命令查看当前的配置信息：

从中可以看出 stonith 已经被禁用。上面的 crm，crm_verify 命令是 1.0 后的版本的 pacemaker 提供的基于命令行的集群管理工具；可以在集群中的任何一个节点上执行。

28.4 为集群添加资源

corosync 支持 heartbeat，LSB 和 ocf 等类型的资源代理，目前较为常用的类型为 LSB 和 OCF 两类，stonith 类专为配置 stonith 设备而用；可以通过如下命令查看当前集群系统所支持的类型：

如果想要查看某种类别下的所用资源代理的列表，可以使用类似如下命令实现：

例如：

28.5 为 Web 创建 IP 资源

接下来要创建的 web 集群创建一个 IP 地址资源，以在通过集群提供 web 服务时使用；这可以通过如下方式实现：

语法：

例子：

28.6 高可用验证

第 29 章　drbd

distributed replicated block device

a software-based, shared-nothing, replicated storage solution mirroring the content of block devices (hard disks, partitions, logical volumes etc.) between servers.

DRBD's core functionality is implemented by way of a Linux Kernel module.

User space administration tools

drbdadm

The high-level administration tool of the DRBD program suite. It obtains all DRBD configuration parameters from the configuration file /etc/drbd.conf

drbdsetup

The program that allows users to configure the DRBD module that has been loaded into the running kernel. It is the low-level tool within the DRBD program suite.

Resource roles

In DRBD, every resource has a role, which may be Primary or Secondary.

Primary

A DRBD device in the primary role can be used unrestrictedly for read and write operations. It may be used for creating and mounting file systems, raw or direct I/O to the block device, etc.

Secondary

A DRBD device in the secondary role receives all updates from the perr node's device, but otherwise disallows access completely. It can not be used by applications, neither for read nor write access.

DRBD modes

Single-primary mode

Any resource is, at any given time, in the primary role on only one cluster

member.

Only one cluster node manipulates the data at any moment.

This mode can be used with any conventional file system (ext3, ext4, XFS, etc.)

Deploying DRBD in single-primary mode is the canonical approach for high availability (fail-over capable) clusters.

Dual-primary mode

Any resource is, at any given time, in the primary role on both cluster nodes.

Since concurrent access to the data is thus possible, this mode requires the use of a shared cluster file system that utilizes a distributed lock manager. Examples include GFS and OCFS2.

Deploying DRBD in dual-primary mode is the preferred approach for load-balancing clusters which require concurrent data access from two nodes.

Disabled by default, and must be enabled explicitly in DRBD's configuration file.

Available in DRBD 8.0 and later.

29.1 Replication Modes

Protocol A

Asynchronous replication protocol.

Local write opeations on the primary node are considered completed as soon as the local disk write has occurred, and the replication packet has been placed in the local TCP send buffer.

In the event of forced fail-over, data loss may occur.

The data on the standby node is consistent after fail-over, however, the most recent updates performed prior to the crash could be lost.

Protocol B

Memory synchronous (semi-synchronous) replication protocol.

Local write opeartions on the primary node are considered completed as soon as the local disk write has occurred, and the replication packet has reached the peer node.

Normally, no writes are lost in case of forced fail-over.

However, in the event of simultaneous power failure on both nodes and concurrent, irreversible destruction of the primary's data store, the most recent writes completed on the primary may be lost.

Protocol C

Synchronous replication protocol.

Local write opeartions on the primary node are considered completed only after both the local and the remote disk write have been confirmed.

Loss of a single node is guranteed not to lead to any data loss.

Data loss is, of course, inveitable even with this replication protocol if both nodes (or their storage subsystems) are irreversibly destroyed at the same time.

Efficient synchronization

Synchronization is necessary if the replication link has been interrupted for any reason

1. failure of the primary node 2. failure of the secondary node 3. or interruption of the replication link

(Re-)synchronization is distinct from device replication

Split brain

Split brain is a situation where, due to temporary failure of all network links between cluster nodes, and possibly due to intervension by a cluster management software or human error, both nodes switched to the primary role while disconnected.

automatic recovery

DRBD allows for automatic operator notification (by email or other means) when it detects split brain.

DRBD has several resolution algorithms available for resolving the split brain automaticly.

1. Discarding modifications made on the ”younger” primary 2. Discarding modifications made on the ”older” primary 3. Discarding modifications on the primary with fewer changes 4. Graceful recovery from split brain if one host has had no intermediate changes

29.2 安装及配置 DRBD

DRBD 包含两大主要组件，内核空间中的驱动代码与用户空间中的管理工具。自从 Linux 内核 2.6.33 版本及之后版本，DRBD 的驱动代码已整合进了内核模块。所以，在 2.6.33 版本及更新版本后，我们只需要安装用户空间的管理工具即可。否则，我们需要同时安装内核支持模块与管理工具两个包，并且两个工具的版本号要保持一致。

实验环境：

机器列表	IP 地址	操作系统	DRBD 版本
node1.laven.com	192.168.1.15	RHEL5U5 32bit	8.3.8
node2.laven.com	192.168.1.16	RHEL5U5 32bit	8.3.8

开始安装

在 node2 上进行同样的安装操作。

开始配置，

安装完成之后，来个简单的配置，使 drbd 可以简单的用起来。drbd 的主配置文件为/etc/drbd.conf，其中还有一个/etc/drbd.d 目录，里面放置了其他的配置文件，这也是便于管理，因此，我们可以分段进行配置，然后放置到该目录即可，在主配置文件中使用 include 指令把这些文件加载进来即可。

看一下全局配置文件，

现在定一个简单的资源 r0, 资源文件后缀以 res 结尾。

配置完毕，我们可以把这些文件复制到节点 node2 上去，

好了，初始化两端的 drbd，然后启动 drbd 服务。

初始化完毕之后，我们把 node1 作为主节点，node2 作为从节点。之后，我们把 /dev/drbd0 格式化然后挂载到/web 目录下，然后在/web 路下创建一些文件，然后到 node2 上检查这些文件是否可以看到。

这时，我们把 node2 作为主节点，并把/dev/drbd0 挂载到/web 目录下，验证是不是有上述文件。首先，需要把/web 目录 umount，然后把 node1 变为从节点。之后，node2 需要先把自己变为主节点，然后挂载/dev/drbd0 到/web 下即可，