



第十二章、正规表示法与文件格式化处理

最近更新日期: 2009/02/10

正规表示法 (Regular Expression, RE, 或称为常规表示法)是透过一些特殊字符的排列, 用以『搜寻/取代/删除』一列或多列文字字符串, 简单的说, 正规表示法就是用在字符串的处理上面的一项『表示式』。正规表示法并不是一个工具程序, 而是一个字符串处理的标准依据, 如果您想要以正规表示法的方式处理字符串, 就得要使用支持正规表示法的工具程序才行, 这类的工具程序很多, 例如 `vi`, `sed`, `awk` 等等。

正规表示法对于系统管理员来说实在是很重要! 因为系统会产生很多的讯息, 这些讯息有的重要有的仅是告知, 此时, 管理员可以透过正规表示法的功能来将重要讯息撷取出来, 并产生便于查阅的报表来简化管理流程。此外, 很多的软件包也都支持正规表示法的分析, 例如邮件服务器的过滤机制(过滤垃圾信件)就是很重要的一个例子。所以, 您最好要了解正规表示法的相关技能, 在未来管理主机时, 才能够更精简处理您的日常事务!

注: 本章节使用者需要多加练习, 因为目前很多的套件都是使用正规表示法来达成其『过滤、分析』的目的, 为了未来主机管理的便利性, 使用者至少要能看的懂正规表示法的意义!

1. 前言: 什么是正规表示法
2. 基础正规表示法
 - 2.1 语系对正规表示法的影响
 - 2.2 `grep` 的一些进阶选项
 - 2.3 基础正规表示法练习
 - 2.4 基础正规表示法字符汇整(characters)
 - 2.5 `sed` 工具: 行的新增/删除, 行的取代/显示, 搜寻并取代, 直接改档
3. 延伸正规表示法
4. 文件的格式化与相关处理
 - 4.1 `printf`: 格式化打印
 - 4.2 `awk`: 好用的数据处理工具
 - 4.3 档案比对工具: `diff`, `cmp`, `patch`
 - 4.4 档案打印准备工具: `pr`
5. 重点回顾
6. 本章习题
7. 参考数据与延伸阅读
8. 针对本文的建议: <http://phorum.vbird.org/viewtopic.php?t=23885>



前言: 什么是正规表示法

约略了解了 Linux 的基本指令 (`BASH`) 并且熟悉了 `vim` 之后, 相信你对于敲

击键盘的打字与指令下达比较不陌生了吧？接下来，底下要开始介绍一个很重要的观念，那就是所谓的『正规表示法 (Regular Expression)』啰！

什么是正规表示法

任何一个有经验的系统管理员，都会告诉你：『正规表示法真是挺重要的！』为什么很重要呢？因为日常生活就使用的到啊！举个例子来说，在你日常使用 `vim` 作字处理或程序撰写时使用到的『搜寻/取代』等等的功能，这些举动要作的漂亮，就得要配合正规表示法来处理啰！

简单的说，正规表示法就是处理字符串的方法，他是以行为单位来进行字符串的处理行为，正规表示法透过一些特殊符号的辅助，可以让使用者轻易的达到『搜寻/删除/取代』某特定字符串的处理程序！

举例来说，我只想找到 `VBird`(前面两个大写字符) 或 `Vbird`(仅有一个大写字符) 这个字样，但是不要其他的字符串 (例如 `VBIRD`, `vbird` 等不需要)，该如何办理？如果在没有正规表示法的环境中 (例如 `MS word`)，你或许就得要使用忽略大小写的办法，或者是分别以 `VBird` 及 `Vbird` 搜寻两遍。但是，忽略大小写可能会搜寻到 `VBIRD/vbird/VbIrD` 等等的不需要的字符串而造成困扰。

再举个系统常见的例子好了，假设妳发现系统在开机的时候，老是会出现一个关于 `mail` 程序的错误，而开机过程的相关程序都是在 `/etc/init.d/` 底下，也就是说，在该目录底下的某个档案内具有 `mail` 这个关键词，你想要将该档案提出来进行查询修改的动作。此时你怎么找出来含有这个关键词的档案？你当然可以一个档案一个档案的开启，然后去搜寻 `mail` 这个关键词，只是.....该目录底下的档案可能不止 100 个说～如果了解正规表示法的相关技巧，那么只要一行指令就找出来啦：『`grep 'mail' /etc/init.d/*`』那个 `grep` 就是支持正规表示法的工具程序之一！如何～很简单吧！

谈到这里就得要进一步说明了，正规表示法基本上是一种『表示法』，只要工具程序支持这种表示法，那么该工具程序就可以用来作为正规表示法的字符串处理之用。也就是说，例如 `vi`, `grep`, `awk`, `sed` 等等工具，因为她们有支持正规表示法，所以，这些工具就可以使用正规表示法的特殊字符来进行字符串的处理。但例如 `cp`, `ls` 等指令并未支持正规表示法，所以就只能使用 `bash` 自己本身的通配符而已。

正规表示法对于系统管理员的用途

那么为何我需要学习正规表示法呢？对于一般使用者来说，由于使用到正规表示法的机会可能不怎么多，因此感受不到他的魅力，不过，对于身为系统管理员的你来说，正规表示法则是一个『不可不学的好东西！』怎么说呢？由于系

统如果在繁忙的情况之下，每天产生的讯息信息会多到你无法想象的地步，而我们也都知道，系统的『[错误讯息登录档案](#)』的内容(这部份我们在第五篇会详谈)记载了系统产生的所有讯息，当然，这包含你的系统是否被『入侵』的记录数据。

但是系统的数据量太大了，要身为系统管理员的你每天去看这么多的讯息数据，从千百行的资料里面找出一行有问题的讯息，呵呵～光是用肉眼去看，想不疯掉都很难！这个时候，我们就可以透过『正规表示法』的功能，将这些登录的信息进行处理，仅取出『有问题』的信息来进行分析，哈哈！如此一来，你的系统管理工作将会『快乐得不得了』啊！当然，正规表示法的优点还不止于此，等你有一定程度的了解之后，你会爱上他喔！

正规表示法的广泛用途

正规表示法除了可以让系统管理员管理主机更为便利之外，事实上，由于正规表示法强大的字符串处理能力，目前一堆软件都支持正规表示法呢！最常见的就是『邮件服务器』啦！

如果你留意因特网上的消息，那么应该不能发现，目前造成网络大塞车的主因之一就是『垃圾/广告信件』了，而如果我们可以在主机端，就将这些问题邮件剔除的话，客户端就会减少很多不必要的带宽耗损了。那么如何剔除广告信件呢？由于广告信件几乎都有一定的标题或者是内容，因此，只要每次有来信时，都先将来信的标题与内容进行特殊字符串的比对，发现有不良信件就予以剔除！嘿！这个工作怎么达到啊？就使用正规表示法啊！目前两大邮件服务器软件 sendmail 与 postfix 以及支持邮件服务器的相关分析套件，都支持正规表示法的比对功能！

当然还不止于此啦，很多的服务器软件、以及套件都支持正规表示法呢！当然，虽然各家软件都支持他，不过，这些『字符串』的比对还是需要系统管理员来加入比对规则的，所以啦！身为系统管理员的你，为了自身的工作以及客户端的需求，正规表示法实在是需要也很值得学习的一项工具呢！

正规表示法与 Shell 在 Linux 当中的角色定位

说实在的，我们在学数学的时候，一个很重要、但是粉难的东西是一定要『背』的，那就是九九表，背成功了之后，未来在数学应用的路途上，真是一帆风顺啊！这个九九表我们在小学的时候几乎背了一整年才背下来，并不是这么好背的呢！但他却是基础当中的基础！你现在一定受惠相当的多呢 ^_^！

而我们谈到的这个正规表示法，与前一章的 [BASH](#) 就有点像是数学的九九表一样，是 Linux 基础当中的基础，虽然也是最难的部分，不过，如果学成了之

后，一定是『大大的有帮助』的！这就好像是金庸小说里面的学武难关：任督二脉！打通任督二脉之后，武功立刻成倍成长！所以啦，不论是对于系统的认识与系统的管理部分，他都有很棒的辅助啊！请好好的学习这个基础吧！^_^

延伸的正规表示法

唔！正规表示法还有分喔？没错喔！正规表示法的字符串表示方式依照不同的严谨度而分为：基础正规表示法与延伸正规表示法。延伸型正规表示法除了简单的一组字符串处理之外，还可以作群组的字符串处理，例如进行搜寻 VBird 或 netman 或 lman 的搜寻，注意，是『或(or)』而不是『和(and)』的处理，此时就需要延伸正规表示法的帮助啦！藉由特殊的『()』与『|』等字符的协助，就能够达到这样的目的！不过，我们在这里主力仅是介绍最基础的基础正规表示法而已啦！好啦！清清脑门，咱们用功去啰！

Tips:

有一点要向大家报告的，那就是：『**正规表示法与通配符是完全不一样的东西！**』这很重要喔！因为『通配符(wildcard)代表的是 bash 操作接口的一个功能』，但正规表示法则是一种字符串处理的表示方式！这两者要分的很清楚才行喔！所以，学习本章，请将前一章 bash 的通配符意义先忘掉吧！



老实说，鸟哥以前刚接触正规表示法时，老想着要将这两者归纳在一起，结果就是... 错误认知一大堆～ 所以才会建议您学习本章先忘记通配符再来学习吧！

基础正规表示法

既然正规表示法是处理字符串的一种表示方式，那么对字符排序有影响的语系数据就会对正规表示法的结果有影响！此外，正规表示法也需要支持工具程序来辅助才行！所以，我们这里就先介绍一个最简单的字符串撷取功能的工具程序，那就是 grep 啰！前一章已经介绍过 grep 的相关选项与参数，本章着重在较进阶的 grep 选项说明啰！介绍完 grep 的功能之后，就进入正规表示法的特殊字符的处理能力了。

语系对正规表示法的影响

为什么语系的数据会影响到正规表示法的输出结果呢？我们在[第零章计算器概论的文字编码系统](#)里面谈到，档案其实记录的仅有 0 与 1，我们看到的字符文字与数字都是透过编码表转换来的。由于不同语系的编码数据并不相同，所以

就会造成数据撷取结果的差异了。举例来说，在英文大小写的编码顺序中，zh_TW.big5 及 C 这两种语系的输出结果分别如下：

- LANG=C 时：0 1 2 3 4 ... A B C D ... Z a b c d ...z
- LANG=zh_TW 时：0 1 2 3 4 ... a A b B c C d D ... z Z

上面的顺序是编码的顺序，我们可以很清楚的发现这两种语系明显就是不一样！如果你想要撷取大写字符而使用 [A-Z] 时，会发现 LANG=C 确实可以仅捉到大写字符（因为是连续的），但是如果 LANG=zh_TW.big5 时，就会发现到，连同小写的 b-z 也会被撷取出来！因为就编码的顺序来看，big5 语系可以撷取到『 A b B c C ... z Z 』这一堆字符哩！所以，使用正规表示法时，需要特别留意当时环境的语系为何，否则可能会发现与别人不相同的撷取结果喔！

由于一般我们在练习正规表示法时，使用的是兼容于 POSIX 的标准，因此就使用『 C 』这个语系(注 1)！因此，底下的很多练习都是使用『 LANG=C 』这个语系数据来进行的喔！另外，为了避免这样编码所造成的英文与数字的撷取问题，因此有些特殊的符号我们得要了解一下的！这些符号主要有底下这些意义：(注 1)

| 特殊符号 | 代表意义 |
|------------|---|
| [:alnum:] | 代表英文大小写字符及数字，亦即 0-9, A-Z, a-z |
| [:alpha:] | 代表任何英文大小写字符，亦即 A-Z, a-z |
| [:blank:] | 代表空格键与 [Tab] 按键两者 |
| [:cntrl:] | 代表键盘上面的控制按键，亦即包括 CR, LF, Tab, Del.. 等等 |
| [:digit:] | 代表数字而已，亦即 0-9 |
| [:graph:] | 除了空格符（空格键与 [Tab] 按键）外的其他所有按键 |
| [:lower:] | 代表小写字符，亦即 a-z |
| [:print:] | 代表任何可以被打印出来的字符 |
| [:punct:] | 代表标点符号 (punctuation symbol)，亦即： " ' ? ! ; : # \$... |
| [:upper:] | 代表大写字符，亦即 A-Z |
| [:space:] | 任何会产生空白的字符，包括空格键，[Tab]，CR 等等 |
| [:xdigit:] | 代表 16 进位的数字类型，因此包括： 0-9, A-F, a-f 的数字与字符 |

尤其上表中的[:alnum:], [:alpha:], [:upper:], [:lower:], [:digit:] 这几个一定要知道代表什么意思，因为他要比 a-z 或 A-Z 的用途要确定的很！

好了，底下就让我们开始来玩玩进阶版的 grep 吧！

grep 的一些进阶选项

我们在第十一章 BASH 里面的 grep 谈论过一些基础用法，但其实 grep 还有不少的进阶用法喔！底下我们仅列出较进阶的 grep 选项与参数给大家参考，基础的 grep 用法请参考前一章的说明啰！

```
[root@www ~]# grep [-A] [-B] [--color=auto] '搜寻字符串' filename
```

选项与参数：

-A：后面可加数字，为 after 的意思，除了列出该行外，后续的 n 行也列出来；

-B：后面可加数字，为 befer 的意思，除了列出该行外，前面的 n 行也列出来；

--color=auto 可将正确的那个撷取数据列出颜色

范例一：用 dmesg 列出核心讯息，再以 grep 找出内含 eth 那行

```
[root@www ~]# dmesg | grep 'eth'
eth0: RealTek RTL8139 at 0xee846000,
00:90:cc:a6:34:84, IRQ 10
eth0: Identified 8139 chip type 'RTL-8139C'
eth0: link up, 100Mbps, full-duplex, lpa 0xC5E1
eth0: no IPv6 routers present
# dmesg 可列出核心产生的讯息！透过 grep 来撷取网络卡
相关信息 (eth)，
# 就可发现如上信息。不过没有行号与特殊颜色显示！看看
下个范例吧！
```

范例二：承上题，要将捉到的关键词显色，且加上行号来表示：

```
[root@www ~]# dmesg | grep -n --color=auto 'eth'
247:eth0: RealTek RTL8139 at 0xee846000,
00:90:cc:a6:34:84, IRQ 10
248:eth0: Identified 8139 chip type 'RTL-8139C'
294:eth0: link up, 100Mbps, full-duplex, lpa 0xC5E1
305:eth0: no IPv6 routers present
# 你会发现除了 eth 会有特殊颜色来表示之外，最前面还
有行号喔！
```

范例三：承上题，在关键词所在行的前两行与后三行也一起

捉出来显示

```
[root@www ~]# dmesg | grep -n -A3 -B2 --color=auto 'eth'
245-PCI: setting IRQ 10 as level-triggered
246-ACPI: PCI Interrupt 0000:00:0e.0[A] -> Link
[LNKB] ...
247:eth0: RealTek RTL8139 at 0xee846000,
00:90:cc:a6:34:84, IRQ 10
248:eth0: Identified 8139 chip type 'RTL-8139C'
249-input: PC Speaker as /class/input/input2
250-ACPI: PCI Interrupt 0000:00:01.4[B] -> Link
[LNKB] ...
251-hdb: ATAPI 48X DVD-ROM DVD-R-RAM CD-R/RW drive,
2048kB Cache, UDMA(66)
# 如上所示，你会发现关键词 247 所在的前两行及 248 后
三行也都被显示出来！
# 这样可以让你将关键词前后数据捉出来进行分析啦！
```

grep 是一个很常见也很常用的指令，他最重要的功能就是进行字符串数据的比对，然后将符合用户需求的字符串打印出来。需要说明的是『grep 在数据中查寻一个字符串时，是以“整行”为单位来进行数据的撷取的！』也就是说，假如一个档案内有 10 行，其中有两行具有你所搜寻的字符串，则将那两行显示在屏幕上，其他的就丢弃了！

在关键词的显示方面，grep 可以使用 `--color=auto` 来将关键词部分使用颜色显示。这可是个很不错的功能啊！但是如果每次使用 grep 都要自行加上 `--color=auto` 又显的很麻烦～此时那个好用的 alias 就得来处理一下啦！你可以在 `~/.bashrc` 内加上这行：『`alias grep='grep --color=auto'`』再以『`source ~/.bashrc`』来立即生效即可喔！这样每次执行 grep 他都会自动帮你加上颜色显示啦！

基础正规表示法练习

要了解正规表示法最简单的方法就是由实际练习去感受啦！所以在汇整正规表示法特殊符号前，我们先以底下这个档案的内容来进行正规表示法的理解吧！先说明一下，底下的练习大前提是：

- 语系已经使用『`export LANG=C`』的设定值；
- grep 已经使用 alias 设定成为『`grep --color=auto`』

至于本章的练习用档案请由底下的连结来下载。需要特别注意的是，底下这个档案是鸟哥在 MS Windows 系统下编辑的，并且已经特殊处理过，因此，他虽然是纯文本档，但是内含一些 Windows 系统下的软件常常自行加入的一些特殊

字符,例如断行字符 (^M) 就是一例! 所以,你可以直接将底下的文字以 vi 储存成 regular_express.txt 这个档案, 不过, 还是比较建议直接点底下的连结:

http://linux.vbird.org/linux_basic/0330regex/regular_express.txt

如果你的 Linux 可以直接连上 Internet 的话,那么使用如下的指令来提取即可:

```
wget
http://linux.vbird.org/linux_basic/0330regex/regular_express.txt
```

至于这个档案的内容如下:

```
[root@www ~]# vi regular_express.txt
"Open Source" is a good mechanism to develop programs.
apple is my favorite food.
Football game is not use feet only.
this dress doesn't fit me.
However, this dress is about $ 3183 dollars.^M
GNU is free air not free beer.^M
Her hair is very beauty.^M
I can't finish the test.^M
Oh! The soup taste good.^M
motorcycle is cheap than car.
This window is clear.
the symbol '*' is represented as start.
Oh!      My god!
The gd software is a library for drafting programs.^M
You are the best is mean you are the no. 1.
The world <Happy> is the same with "glad".
I like dog.
google is the best tools for search keyword.
goooooogle yes!
go! go! Let's go.
# I am VBird
```

这档案共有 22 行, 最底下一行为空白行! 现在开始我们一个案例一个案例的来介绍吧!

- 例题一、搜寻特定字符串

搜寻特定字符串很简单吧？假设我们要从刚刚的档案当中取得 the 这个特定字符串，最简单的方式就是这样：

```
[root@www ~]# grep -n 'the' regular_express.txt
8:I can't finish the test.
12:the symbol '*' is represented as start.
15:You are the best is mean you are the no. 1.
16:The world <Happy> is the same with "glad".
18:google is the best tools for search keyword.
```

那如果想要『反向选择』呢？也就是说，当该行没有 'the' 这个字符串时才显示在屏幕上，那就直接使用：

```
[root@www ~]# grep -vn 'the' regular_express.txt
```

你会发现，屏幕上出现的行列为除了 8, 12, 15, 16, 18 五行之外的其他行列！接下来，如果你想要取得不论大小写的 the 这个字符串，则：

```
[root@www ~]# grep -in 'the' regular_express.txt
8:I can't finish the test.
9:Oh! The soup taste good.
12:the symbol '*' is represented as start.
14:The gd software is a library for drafting programs.
15:You are the best is mean you are the no. 1.
16:The world <Happy> is the same with "glad".
18:google is the best tools for search keyword.
```

除了多两行（9, 14 行）之外，第 16 行也多了一个 The 的关键词被撷取到喔！

- 例题二、利用中括号 [] 来搜寻集合字符

如果我想要搜寻 test 或 taste 这两个单字时，可以发现到，其实她们有共通的 't?st' 存在~这个时候，我可以这样来搜寻：

```
[root@www ~]# grep -n 't[ae]st' regular_express.txt
8:I can't finish the test.
9:Oh! The soup taste good.
```

了解了吧？其实 `[]` 里面不论有几个字符，他都谨代表某『一个』字符，所以，上面的例子说明了，我需要的字符串是『tast』或『test』两个字符串而已！而如果想要搜寻到有 `oo` 的字符时，则使用：

```
[root@www ~]# grep -n 'oo' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
2:apple is my favorite food.
3:Football game is not use feet only.
9:Oh! The soup taste good.
18:google is the best tools for search keyword.
19:gooooooogle yes!
```

但是，如果我不想要 `oo` 前面有 `g` 的话呢？此时，可以利用在集合字符的反向选择 `[^]` 来达成：

```
[root@www ~]# grep -n '[^g]oo' regular_express.txt
2:apple is my favorite food.
3:Football game is not use feet only.
18:google is the best tools for search keyword.
19:gooooooogle yes!
```

意思就是说，我需要的是 `oo`，但是 `oo` 前面不能是 `g` 就是了！仔细比较上面两个表格，你会发现，第 1,9 行不见了，因为 `oo` 前面出现了 `g` 所致！第 2,3 行没有疑问，因为 `foo` 与 `Foo` 均可被接受！但是第 18 行明明有 `google` 的 `goo` 啊～别忘记了，因为该行后面出现了 `tool` 的 `too` 啊！所以该行也被列出来～也就是说，18 行里面虽然出现了我们所不要的项目（`goo`）但是由于有需要的项目（`too`），因此，是符合字符串搜寻的喔！

至于第 19 行，同样的，因为 `gooooooogle` 里面的 `oo` 前面可能是 `o`，例如：`go(ooo)oogle`，所以，这一行也是符合需求的！

再来，假设我 `oo` 前面不想要有小写字符，所以，我可以这样写 `[^abcd...z]oo`，但是这样似乎不怎么方便，由于小写字符的 ASCII 上编码的顺序是连续的，因此，我们可以将之简化为底下这样：

```
[root@www ~]# grep -n '[^a-z]oo' regular_express.txt
3:Football game is not use feet only.
```

也就是说，当我们在一组集合字符中，如果该字符组是连续的，例如大写英文/小写英文/数字等等，就可以使用 `[a-z]`, `[A-Z]`, `[0-9]` 等方式来书写，那么如果我们的要求字符串是数字与英文呢？呵呵！就将他全部写在一起，变成：

[a-zA-Z0-9]。例如，我们要取得有数字的那一行，就这样：

```
[root@www ~]# grep -n '[0-9]' regular_express.txt
5:However, this dress is about $ 3183 dollars.
15:You are the best is mean you are the no. 1.
```

但由于考虑到语系对于编码顺序的影响，因此除了连续编码使用减号『 - 』之外，你也可以使用如下的方法来取得前面两个测试的结果：

```
[root@www ~]# grep -n '[[:lower:]]oo'
regular_express.txt
# 那个 [[:lower:]] 代表的就是 a-z 的意思！请参考前两小节的说明表格

[root@www ~]# grep -n '[[:digit:]]'
regular_express.txt
```

这样对于 [] 以及 [^] 以及 [] 当中的 -，还有关于前面表格提到的特殊关键词有了解了吗？^_^！

- 例题三、行首与行尾字符 ^ \$

我们在例题一当中，可以查询到一行字符串里面有 the 的，那如果我要让 the 只在行首列出呢？这个时候就得要使用制表符了！我们可以这样做：

```
[root@www ~]# grep -n '^the' regular_express.txt
12:the symbol '*' is represented as start.
```

此时，就只剩下第 12 行，因为只有第 12 行的行首是 the 开头啊～此外，如果我要开头是小写字母的那一行就列出呢？可以这样：

```
[root@www ~]# grep -n '^[a-z]' regular_express.txt
2:apple is my favorite food.
4:this dress doesn't fit me.
10:motorcycle is cheap than car.
12:the symbol '*' is represented as start.
18:google is the best tools for search keyword.
19:gooooooogle yes!
20:go! go! Let's go.
```

你可以发现我们可以捉到第一个字符都不是大写的！只不过 grep 列出的关键词部分不只有第一个字符，grep 是列出一整个字（word）说！同样的，上面的指令也可以用如下的方式来取代的：

```
[root@www ~]# grep -n '^[[:lower:]]'
regular_express.txt
```

好！那如果我不想要开头是英文字母，则可以是这样：

```
[root@www ~]# grep -n '^[^a-zA-Z]' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
21:# I am VBird
# 指令也可以是： grep -n '^[^[:alpha:]]'
regular_express.txt
```

注意到了吧？那个 ^ 符号，在字符集合符号(括号[])之内与之外是不同的！在 [] 内代表『反向选择』，在 [] 之外则代表定位在行首的意义！要分清楚喔！反过来思考，那如果我想要找出来，行尾结束为小数点 (.) 的那一行，该如何处理：

```
[root@www ~]# grep -n '\.$' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
2:apple is my favorite food.
3:Football game is not use feet only.
4:this dress doesn't fit me.
10:motorcycle is cheap than car.
11:This window is clear.
12:the symbol '*' is represented as start.
15:You are the best is mean you are the no. 1.
16:The world <Happy> is the same with "glad".
17:I like dog.
18:google is the best tools for search keyword.
20:go! go! Let's go.
```

特别注意到，因为小数点具有其他意义(底下会介绍)，所以必须要使用跳脱字符(\)来加以解除其特殊意义！不过，你或许会觉得奇怪，但是第 5~9 行最后面也是 . 啊～怎么无法打印出来？这里就牵涉到 Windows 平台的软件对于断行字符的判断问题了！我们使用 cat -A 将第五行拿出来看，你会发现：

```
[root@www ~]# cat -An regular_express.txt | head -n 10
| tail -n 6
    5  However, this dress is about $ 3183 dollars. ^M$
    6  GNU is free air not free beer. ^M$
    7  Her hair is very beauty. ^M$
    8  I can't finish the test. ^M$
    9  Oh! The soup taste good. ^M$
   10  motorcycle is cheap than car. $
```

我们在第十章内谈到过断行字符在 Linux 与 Windows 上的差异，在上面的表格中我们可以发现 5~9 行为 Windows 的断行字符（^M\$），而正常的 Linux 应该仅有第 10 行显示的那样（\$）。所以啰，那个 . 自然就不是紧接在 \$ 之前喔！也就捉不到 5~9 行了！这样可以了解 ^ 与 \$ 的意义吗？好了，先不要看底下的解答，自己想一想，那么如果我想要找出来，哪一行是『空白行』，也就是说，该行并没有输入任何数据，该如何搜寻？

```
[root@www ~]# grep -n '^$' regular_express.txt
22:
```

因为只有行首跟行尾（^\$），所以，这样就可以找出空白行啦！再来，假设你已经知道在一个程序脚本（shell script）或者是配置文件当中，空白行与开头为 # 的那一行是批注，因此如果你要将资料列出给别人参考时，可以将这些数据省略掉以节省宝贵的纸张，那么你可以怎么作呢？我们以 /etc/syslog.conf 这个档案来作范例，你可以自行参考一下输出的结果：

```
[root@www ~]# cat -n /etc/syslog.conf
# 在 CentOS 中，结果可以发现 33 行的输出，很多空白
# 行与 # 开头

[root@www ~]# grep -v '^$' /etc/syslog.conf | grep -v
'^#'
# 结果仅有 10 行，其中第一个『-v '^$'』代表『不要
# 空白行』，
# 第二个『-v '^#'』代表『不要开头是 # 的那行』喔！
```

是否节省很多版面啊？

-
- 例题四、任意一个字符 . 与重复字符 *

在 第十一章 bash 当中，我们知道通配符 * 可以用来代表任意(0 或多个)字符，但是正规表示法并不是通配符，两者之间是不相同的！至于正规表示法

当中的『.』则代表『绝对有一个任意字符』的意思！这两个符号在正规表示法的意义如下：

- . (小数点)：代表『一定有一个任意字符』的意思；
- * (星星号)：代表『重复前一个 0 到无穷多次』的意思，为组合形态

这样讲不好懂，我们直接做个练习吧！假设我需要找出 `g??d` 的字符串，亦即共有四个字符，起头是 `g` 而结束是 `d`，我可以这样做：

```
[root@www ~]# grep -n 'g..d' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
9:Oh! The soup taste good.
16:The world <Happy> is the same with "glad".
```

因为强调 `g` 与 `d` 之间一定要存在两个字符，因此，第 13 行的 `god` 与第 14 行的 `gd` 就不会被列出来啦！再来，如果我想要列出有 `oo`, `ooo`, `oooo` 等等的的数据，也就是说，至少要有两个(含) `o` 以上，该如何是好？是 `o*` 还是 `oo*` 还是 `ooo*` 呢？虽然你可以试看看结果，不过结果太占版面了 @_@，所以，我这里就直接说明。

因为 `*` 代表的是『重复 0 个或多个前面的 RE 字符』的意义，因此，『`o*`』代表的是：『拥有空字符或一个 `o` 以上的字符』，特别注意，因为允许空字符(就是有没有字符都可以的意思)，因此，『`grep -n 'o*' regular_express.txt`』将会把所有的数据都打印出来屏幕上！

那如果是『`oo*`』呢？则第一个 `o` 肯定必须要存在，第二个 `o` 则是可有可无的多个 `o`，所以，凡是含有 `o`, `oo`, `ooo`, `oooo` 等等，都可以被列出来～

同理，当我们需要『至少两个 `o` 以上的字符串』时，就需要 `ooo*`，亦即是：

```
[root@www ~]# grep -n 'ooo*' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
2:apple is my favorite food.
3:Football game is not use feet only.
9:Oh! The soup taste good.
18:google is the best tools for search keyword.
19:gooooooooooole yes!
```

这样理解 `*` 的意义了吗？好了，现在出个练习，如果我想要字符串开头与结尾

都是 g,但是两个 g 之间仅能存在至少一个 o ,亦即是 gog, goog, goooog....
等等, 那该如何?

```
[root@www ~]# grep -n 'goo*g' regular_express.txt
18:google is the best tools for search keyword.
19:gooooooogle yes!
```

如此了解了吗? 再来一题, 如果我想要找出 g 开头与 g 结尾的字符串, 当中的字符可有可无, 那该如何是好? 是『g*g』吗?

```
[root@www ~]# grep -n 'g*g' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
3:Football game is not use feet only.
9:Oh! The soup taste good.
13:Oh! My god!
14:The gd software is a library for drafting programs.
16:The world <Happy> is the same with "glad".
17:I like dog.
18:google is the best tools for search keyword.
19:gooooooogle yes!
20:go! go! Let's go.
```

但测试的结果竟然出现这么多行? 太诡异了吧? 其实一点也不诡异, 因为 g*g 里面的 g* 代表『空字符或一个以上的 g』在加上后面的 g, 因此, 整个 RE 的内容就是 g, gg, ggg, gggg, 因此, 只要该行当中拥有一个以上的 g 就符合所需了!

那该如何得到我们的 g...g 的需求呢? 呵呵! 就利用任意一个字符『.』啊! 亦即是:『g.*g』的作法, 因为 * 可以是 0 或多个重复前面的字符, 而 . 是任意字符, 所以:『.* 就代表零个或多个任意字符』的意思啦!

```
[root@www ~]# grep -n 'g.*g' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
14:The gd software is a library for drafting programs.
18:google is the best tools for search keyword.
19:gooooooogle yes!
20:go! go! Let's go.
```

因为是代表 g 开头与 g 结尾, 中间任意字符均可接受, 所以, 第 1, 14, 20 行是可接受的喔! 这个 .* 的 RE 表示任意字符是很常见的, 希望大家能够理解

并且熟悉！再出一题，如果我想要找出『任意数字』的行列呢？因为仅有数字，所以就成为：

```
[root@www ~]# grep -n '[0-9][0-9]*'
regular_express.txt
5:However, this dress is about $ 3183 dollars.
15:You are the best is mean you are the no. 1.
```

虽然使用 `grep -n '[0-9]' regular_express.txt` 也可以得到相同的结果，但鸟哥希望大家能够理解上面指令当中 RE 表示法的意义才好！

• 例题五、限定连续 RE 字符范围 {}

在上个例题当中，我们可以利用 `.` 与 RE 字符及 `*` 来设定 0 个到无限多个重复字符，那如果我想要限制一个范围区间内的重复字符数呢？举例来说，我想要找出两个到五个 `o` 的连续字符串，该如何作？这时候就得要使用到限定范围的字符 `{}` 了。但因为 `{` 与 `}` 的符号在 shell 是有特殊意义的，因此，我们必须使用跳脱字符 `\` 来让他失去特殊意义才行。至于 `{}` 的语法是这样的，假设我要找到两个 `o` 的字符串，可以是：

```
[root@www ~]# grep -n 'o\{2\}' regular_express.txt
1:"Open Source" is a good mechanism to develop
programs.
2:apple is my favorite food.
3:Football game is not use feet only.
9:Oh! The soup taste good.
18:google is the best tools for search keyword.
19:gooooooogle yes!
```

这样看似与 `ooo*` 的字符没有什么差异啊？因为第 19 行有多个 `o` 依旧也出现了！好，那么换个搜寻的字符串，假设我们要找出 `g` 后面接 2 到 5 个 `o`，然后再接一个 `g` 的字符串，他会是这样：

```
[root@www ~]# grep -n 'go\{2,5\}g' regular_express.txt
18:google is the best tools for search keyword.
```

嗯！很好！第 19 行终于没有被取用了(因为 19 行有 6 个 `o` 啊！)。那么，如果我想要的是 2 个 `o` 以上的 `goooo...g` 呢？除了可以是 `gooo*g`，也可以是：

```
[root@www ~]# grep -n 'go\{2,\}g' regular_express.txt
18:google is the best tools for search keyword.
19:gooooooogle yes!
```

呵呵！就可以找出来啦～

🔦基础正规表示法字符汇整 (characters)

经过了上面的几个简单的范例，我们可以将基础的正规表示法特殊字符汇整如下：

| RE 字符 | 意义与范例 |
|----------------------|--|
| <code>^word</code> | <u>意义：待搜寻的字符串(word)在行首！</u> 范例：搜寻行首为 # 开始的那一行，并列出行号 <code>grep -n '^#' regular_express.txt</code> |
| <code>word\$</code> | <u>意义：待搜寻的字符串(word)在行尾！</u> 范例：将行尾为 ! 的那一行打印出来，并列出行号 <code>grep -n '!\$' regular_express.txt</code> |
| <code>.</code> | <u>意义：代表『一定有一个任意字符』的字符！</u> 范例：搜寻的字符串可以是 (eve) (eae) (eee) (e e)，但不能仅有 (ee)！亦即 e 与 e 中间『一定』仅有一个字符，而空格符也是字符！ <code>grep -n 'e.e' regular_express.txt</code> |
| <code>\</code> | <u>意义：跳脱字符，将特殊符号的特殊意义去除！</u> 范例：搜寻含有单引号 ' 的那一行！ <code>grep -n \' regular_express.txt</code> |
| <code>*</code> | <u>意义：重复零个到无穷多个的前一个 RE 字符</u> 范例：找出含有 (es) (ess) (esss) 等等的字符串，注意，因为 * 可以是 0 个，所以 es 也是符合带搜寻字符串。另外，因为 * 为重复『前一个 RE 字符』的符号，因此，在 * 之前必须要紧接着一个 RE 字符喔！例如任意字符则为『.*』！ <code>grep -n 'ess*' regular_express.txt</code> |
| <code>[list]</code> | <u>意义：字符集合的 RE 字符，里面列出想要撷取的字符！</u> 范例：搜寻含有 (gl) 或 (gd) 的那一行，需要特别留意的是，在 [] 当中『谨代表一个待搜寻的字符』，例如『a[af1]y』代表搜寻的字符串可以是 aay, afy, aly 即 [af1] 代表 a 或 f 或 1 的意思！ <code>grep -n 'g[ld]' regular_express.txt</code> |
| <code>[n1-n2]</code> | 意义：字符集合的 RE 字符，里面列出想要撷取的字符范围！ |

| | |
|----------------------|---|
| | <p>范例：搜寻含有任意数字的那一行！需特别留意，在字符集合 <code>[]</code> 中的减号 <code>-</code> 是有特殊意义的，他代表两个字符之间的所有连续字符！但这个连续与否与 ASCII 编码有关，因此，你的编码需要设定正确（在 <code>bash</code> 当中，需要确定 <code>LANG</code> 与 <code>LANGUAGE</code> 的变量是否正确！）例如所有大写字符则为 <code>[A-Z]</code></p> <p><code>grep -n '[0-9]' regular_express.txt</code></p> |
| <code>[^list]</code> | <p><u>意义：字符集合的 RE 字符，里面列出不要的字符串或范围！</u></p> <p>范例：搜寻的字符串可以是 <code>(oog)</code> <code>(ood)</code> 但不能是 <code>(oot)</code>，那个 <code>^</code> 在 <code>[]</code> 内时，代表的意义是『反向选择』的意思。例如，我不要大写字符，则为 <code>[^A-Z]</code>。但是，需要特别注意的是，如果以 <code>grep -n [^A-Z] regular_express.txt</code> 来搜寻，却发现该档案内的所有行都被列出，为什么？因为这个 <code>[^A-Z]</code> 是『非大写字符』的意思，因为每一行均有非大写字符，例如第一行的 "Open Source" 就有 <code>p, e, n, o...</code> 等等的小写字</p> <p><code>grep -n 'oo[^t]' regular_express.txt</code></p> |
| <code>\{n,m\}</code> | <p><u>意义：连续 <code>n</code> 到 <code>m</code> 个的『前一个 RE 字符』</u></p> <p><u>意义：若为 <code>\{n\}</code> 则是连续 <code>n</code> 个的前一个 RE 字符，</u></p> <p><u>意义：若是 <code>\{n,\}</code> 则是连续 <code>n</code> 个以上的前一个 RE 字符！</u></p> <p>范例：在 <code>g</code> 与 <code>g</code> 之间有 2 个到 3 个的 <code>o</code> 存在的字符串，亦即 <code>(goog)</code> <code>(gooog)</code></p> <p><code>grep -n 'go\{2,3\}g' regular_express.txt</code></p> |

再次强调：『正规表示法的特殊字符』与一般在指令列输入指令的『通配符』并不相同，例如，在通配符当中的 `*` 代表的是『0~ 无限多个字符』的意思，但是在正规表示法当中，`*` 则是『重复 0 到无穷多个的前一个 RE 字符』的意思~使用的意义并不相同，不要搞混了！

举例来说，不支持正规表示法的 `ls` 这个工具中，若我们使用『`ls -l *`』代表的是任意档名的档案，而『`ls -l a*`』代表的是以 `a` 为开头的任何档名的档案，但在正规表示法中，我们要找到含有以 `a` 为开头的档案，则必须要这样：（需搭配支持正规表示法的工具）

```
ls | grep -n '^a.*'
```

例题：

以 `ls -l` 配合 `grep` 找出 `/etc/` 底下文件类型为链接文件属性的文件名

答：

由于 `ls -l` 列出连结档时标头会是『`lrwxrwxrwx`』，因此使用如下的指令即可找出结果：

```
ls -l /etc | grep '^l'
```

若仅想要列出几个档案，再以『 |wc -l 』 来累加处理即可。

sed 工具

在了解了一些正规表示法的基础应用之后，再来呢？呵呵～两个东西可以玩一玩的，那就是 sed 跟底下会介绍的 awk 了！这两个家伙可是相当的有用的啊！举例来说，鸟哥写的 [logfile.sh 分析登录文件的小程序](#)，绝大部分分析关键词的取用、统计等等，就是用这两个宝贝蛋来帮我完成的！那么你说，要不要玩一玩啊？^_^

我们先来谈一谈 sed 好了，sed 本身也是一个管线命令，可以分析 standard input 的啦！而且 sed 还可以将数据进行取代、删除、新增、撷取特定行等的功能呢！很不错吧～我们先来了解一下 sed 的用法，再来聊他的用途好了！

```
[root@www ~]# sed [-nefr] [动作]
```

选项与参数：

- n ：使用安静(silent)模式。在一般 sed 的用法中，所有来自 STDIN 的数据一般都会被列出到屏幕上。但如果加上 -n 参数后，则只有经过 sed 特殊处理的那一行(或者动作)才会被列出来。
- e ：直接在指令列模式上进行 sed 的动作编辑；
- f ：直接将 sed 的动作写在一个档案内，-f filename 则可以执行 filename 内的 sed 动作；
- r ：sed 的动作支持的是延伸型正规表示法的语法。(预设是基础正规表示法语法)
- i ：直接修改读取的档案内容，而不是由屏幕输出。

动作说明： [n1[,n2]]function

n1, n2 ：不见得会存在，一般代表『选择进行动作的行数』，举例来说，如果我的动作是需要在 10 到 20 行之间进行的，则『 10,20[动作行为] 』

function 有底下这些咚咚：

- a ：新增，a 的后面可以接字符串，而这些字符串会在新的一行出现(目前的下一行)～
- c ：取代，c 的后面可以接字符串，这些字符串可以取代 n1,n2 之间的行！

d : 删除，因为是删除啊，所以 d 后面通常不接任何咚咚；
i : 插入，i 的后面可以接字符串，而这些字符串会在新的一行出现(目前的上一行)；
p : 打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运作～
s : 取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配
正规表示法！例如 1,20s/old/new/g 就是啦！

- 以行为单位的新增/删除功能

sed 光是用看的是看不懂的啦！所以又要来练习了！先来玩玩删除与新增的功能吧！

范例一：将 /etc/passwd 的内容列出并且打印行号，同时，请将第 2~5 行删除！
[root@www ~]# nl /etc/passwd | sed '2,5d'
1 root:x:0:0:root:/root:/bin/bash
6 sync:x:5:0:sync:/sbin:/bin/sync
7
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
.....(后面省略).....

看到了吧？sed 的动作为 '2,5d'，那个 d 就是删除！因为 2-5 行给他删除了，所以显示的数据就没有 2-5 行啰～另外，注意一下，原本应该是要下达 sed -e 才对，没有 -e 也行啦！同时也要注意的，sed 后面接的动作，请务必以 '' 两个单引号括住喔！

如果题型变化一下，举例来说，如果只要删除第 2 行，可以使用『nl /etc/passwd | sed '2d'』来达成，至于若是要删除第 3 到最后一行，则是『nl /etc/passwd | sed '3,\$d'』的啦，那个钱字号『\$』代表最后一行！

范例二：承上题，在第二行后(亦即是加在第三行)加上『drink tea?』字样！
[root@www ~]# nl /etc/passwd | sed '2a drink tea?'
1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
drink tea
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....(后面省略).....

嘿嘿！在 a 后面加上的字符串就已将出现在第二行后面啰！那如果是要在第二行前呢？『 nl /etc/passwd | sed '2i drink tea' 』就对啦！就是将『 a 』变成『 i 』即可。增加一行很简单，那如果是要增将两行以上呢？

```
范例三：在第二行后面加入两行字，例如『Drink tea  
or .....』与『drink beer?』  
[root@www ~]# nl /etc/passwd | sed '2a Drink tea  
or .....\  
> drink beer ?'  
1 root:x:0:0:root:/root:/bin/bash  
2 bin:x:1:1:bin:/bin:/sbin/nologin  
Drink tea or .....  
drink beer ?  
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin  
..... (后面省略).....
```

这个范例的重点是『我们可以新增不只一行喔！可以新增好几行』但是每一行之间都必须要以反斜杠『 \ 』来进行新行的增加喔！所以，上面的例子中，我们可以发现在第一行的最后面就有 \ 存在啦！那是一定要的喔！

-
- 以行为单位的取代与显示功能

刚刚是介绍如何新增与删除，那么如果要整行取代呢？看看底下的范例吧：

```
范例四：我想将第 2-5 行的内容取代成为『No 2-5 number』  
呢？  
[root@www ~]# nl /etc/passwd | sed '2,5c No 2-5 number'  
1 root:x:0:0:root:/root:/bin/bash  
No 2-5 number  
6 sync:x:5:0:sync:/sbin:/bin/sync  
..... (后面省略).....
```

透过这个方法我们就能够将数据整行取代了！非常容易吧！sed 还有更好用的东东！我们以前想要列出第 11~20 行，得要透过『head -n 20 | tail -n 10』之类的方法来处理，很麻烦啦～ sed 则可以简单的直接取出你想要的那几行！是透过行号来捉的喔！看看底下的范例先：

```
范例五：仅列出 /etc/passwd 档案内的第 5-7 行  
[root@www ~]# nl /etc/passwd | sed -n '5,7p'  
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

```
6 sync:x:5:0:sync:/sbin:/bin/sync
7
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

上述的指令中有个重要的选项『-n』，按照说明文件，这个 -n 代表的是『安静模式』！那么为什么要使用安静模式呢？你可以自行下达 `sed '5,7p'` 就知道了（5-7 行会重复输出）！有没有加上 -n 的参数时，输出的数据可是差很多的喔！你可以透过这个 sed 的以行为单位的显示功能，就能够将某一个档案内的某些行号捉出来查阅！很棒的功能！不是吗？

- 部分数据的搜寻并取代的功能

除了整行的处理模式之外，sed 还可以用行为单位进行部分数据的搜寻并取代的功能喔！基本上 sed 的搜寻与取代的与 vi 相当的类似！他有点像这样：

```
sed 's/要被取代的字符串/新的字符串/g'
```

上表中特殊字体的部分为关键词，请记下来！至于三个斜线分成两栏就是新旧字符串的替换啦！我们使用底下这个取得 IP 数据的范例，一段一段的来处理给您瞧瞧，让你了解一下什么是咱们所谓的搜寻并取代吧！

步骤一：先观察原始讯息，利用 `/sbin/ifconfig` 查询 IP 为何？

```
[root@www ~]# /sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr
00:90:CC:A6:34:84
          inet addr:192.168.1.100
Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::290:ccff:fea6:3484/64
Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500
Metric:1
```

..... (以下省略).....

因为我们还没有讲到 IP，这里你先有个概念即可啊！我们的重点在第二行，

也就是 192.168.1.100 那一行而已！先利用关键词提出那一行！

步骤二：利用关键词配合 `grep` 撷取出关键的一行数据

```
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr'
```

```

        inet addr:192.168.1.100
Bcast:192.168.1.255 Mask:255.255.255.0
# 当场仅剩下一行！接下来，我们要将开始到 addr: 通通
删除，就是像底下这样：
# inet addr:192.168.1.100 Bcast:192.168.1.255
Mask:255.255.255.0
# 上面的删除关键在于『 ^.*inet addr: 』啦！正规表示
法出现！ ^_^

步骤三：将 IP 前面的部分予以删除
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr'
| \
> sed 's/^.*addr://g'
192.168.1.100 Bcast:192.168.1.255
Mask:255.255.255.0
# 仔细与上个步骤比较一下，前面的部分不见了！接下来则
是删除后续的部分，亦即：
# 192.168.1.100 Bcast:192.168.1.255
Mask:255.255.255.0
# 此时所需的正规表示法为：『 Bcast.*$ 』就是啦！

步骤四：将 IP 后面的部分予以删除
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr'
| \
> sed 's/^.*addr://g' | sed 's/Bcast.*$//g'
192.168.1.100

```

透过这个范例的练习也建议您依据此一步骤来研究你的指令！就是先观察，然后再一层一层的试做，如果有做不对的地方，就先予以修改，改完之后测试，成功后再往下继续测试。以鸟哥上面的介绍中，那一大串指令就做了四个步骤！对吧！ ^_^

让我们再来继续研究 sed 与正规表示法的配合练习！假设我只要 MAN 存在的那几行数据，但是含有 # 在内的批注我不想要，而且空白行我也不要！此时该如何处理呢？可以透过这几个步骤来实作看看：

```

步骤一：先使用 grep 将关键词 MAN 所在行取出来
[root@www ~]# cat /etc/man.config | grep 'MAN'
# when MANPATH contains an empty substring), to find
out where the cat
# MANBIN                pathname
# MANPATH                manpath_element
[corresponding_catdir]

```

```
# MANPATH_MAP                path_element
manpath_element
# MANBIN                      /usr/local/bin/man
# Every automatically generated MANPATH includes these
fields
MANPATH /usr/man
.... (后面省略)....
```

步骤二：删除掉批注之后的数据！

```
[root@www ~]# cat /etc/man.config | grep 'MAN' | sed
's/#.*$/g'
```

```
MANPATH /usr/man
.... (后面省略)....
```

从上面可以看出来，原本批注的数据都变成空白行啦！所以，接下来要删除掉空白行

```
[root@www ~]# cat /etc/man.config | grep 'MAN' | sed
's/#.*$/g' | \
> sed '/^$/d'
MANPATH /usr/man
MANPATH /usr/share/man
MANPATH /usr/local/man
.... (后面省略)....
```

- 直接修改档案内容(危险动作)

你以为 sed 只有这样的能耐吗？那可不行！sed 甚至可以直接修改档案的内容呢！而不必使用管线命令或数据流重导向！不过，由于这个动作会直接修改到原始的档案，所以请你千万不要随便拿系统配置文件来测试喔！我们还是使用你下载的 regular_express.txt 档案来测试看看吧！

范例六：利用 sed 将 regular_express.txt 内每一行结尾若为 . 则换成 !

```
[root@www ~]# sed -i 's/\.$/!/g' regular_express.txt
# 上头的 -i 选项可以让你的 sed 直接去修改后面接的档案内容而不是由屏幕输出喔！
```

```
# 这个范例是用在取代！请您自行 cat 该档案去查阅结果  
啰！
```

范例七：利用 sed 直接在 regular_express.txt 最后一行
加入『# This is a test』

```
[root@www ~]# sed -i '$a # This is a test'  
regular_express.txt
```

由于 \$ 代表的是最后一行，而 a 的动作是新增，因此该
档案最后新增啰！

sed 的『-i』选项可以直接修改档案内容，这功能非常有帮助！举例来说，如果你有一个 100 万行的档案，你要在第 100 行加某些文字，此时使用 vim 可能会疯掉！因为档案太大了！那怎么办？就利用 sed 啊！透过 sed 直接修改/取代的功能，你甚至不需要使用 vim 去修订！很棒吧！

总之，这个 sed 不错用啦！而且很多的 shell script 都会使用到这个指令的功能～ sed 可以帮助系统管理员管理好日常的工作喔！要仔细的学习呢！



延伸正规表示法

事实上，一般读者只要了解基础型的正规表示法大概就已经相当足够了，不过，某些时刻为了要简化整个指令操作，了解一下使用范围更广的延伸型正规表示法的表示式会更方便呢！举个简单的例子好了，在上节的[例题三](#)的[最后一个例子](#)中，我们要去除空白行与行首为 # 的行列，使用的是

```
grep -v '^$' regular_express.txt | grep -v '^#'
```

需要使用到管线命令来搜寻两次！那么如果使用延伸型的正规表示法，我们可以简化为：

```
egrep -v '^$|^#' regular_express.txt
```

延伸型正规表示法可以透过群组功能『|』来进行一次搜寻！那个在单引号内的管线意义为『或 or』啦！是否变的更简单呢？此外，grep 预设仅支持基础正规表示法，如果要使用延伸型正规表示法，你可以使用 grep -E，不过更建议直接使用 egrep！直接区分指令比较好记忆！其实 egrep 与 grep -E 是类似命令别名的关系啦！

熟悉了正规表示法之后，到这个延伸型的正规表示法，你应该也会想到，不就是多几个重要的特殊符号吗？`^_~y` 是的～所以，我们就直接来说明一下，延伸型正规表示法有哪几个特殊符号？由于底下的范例还是有使用到 regular_express.txt，不巧的是刚刚我们可能将该档案修改过了 @_@，所以，

请重新下载该档案来练习喔！

| RE 字符 | 意义与范例 |
|-------|---|
| + | <p>意义：重复『一个或一个以上』的前一个 RE 字符</p> <p>范例：搜寻 (god) (good) (goood)... 等等的字符串。那个 o+ 代表『一个以上的 o』所以，底下的执行成果会将第 1, 9, 13 行列出来。</p> <p><code>egrep -n 'go+d' regular_express.txt</code></p> |
| ? | <p>意义：『零个或一个』的前一个 RE 字符</p> <p>范例：搜寻 (gd) (god) 这两个字符串。那个 o? 代表『空的或 1 个 o』所以，上面的执行成果会将第 13, 14 行列出来。有没有发现到，这两个案例('go+d' 与 'go?d')的结果集合与 'go*d' 相同？想想看，这是为什么喔！ ^_^</p> <p><code>egrep -n 'go?d' regular_express.txt</code></p> |
| | <p>意义：用或 (or) 的方式找出数个字符串</p> <p>范例：搜寻 gd 或 good 这两个字符串，注意，是『或』！所以，第 1, 9, 14 这三行都可以被打印出来喔！那如果还想要找出 dog 呢？</p> <p><code>egrep -n 'gd good' regular_express.txt</code> <code>egrep -n 'gd good dog' regular_express.txt</code></p> |
| () | <p>意义：找出『群组』字符串</p> <p>范例：搜寻 (glad) 或 (good) 这两个字符串，因为 g 与 d 是重复的，所以，我就可以将 la 与 oo 列于 () 当中，并以 来分隔开来，就可以啦！</p> <p><code>egrep -n 'g(la oo)d' regular_express.txt</code></p> |
| ()+ | <p>意义：多个重复群组的判别</p> <p>范例：将『AxyzxyzxyzxyzC』用 echo 叫出，然后再使用如下的方法搜寻一下！</p> <p><code>echo 'AxyzxyzxyzxyzC' egrep 'A(xyz)+C'</code></p> <p>上面的例子意思是说，我要找开头是 A 结尾是 C，中间有一个以上的 "xyz" 字符串的意思～</p> |

以上这些就是延伸型的正规表示法的特殊字符。另外，要特别强调的是，那个 ! 在正规表示法当中并不是特殊字符，所以，如果你想要查出来档案中含有 ! 与 > 的字行时，可以这样：

```
grep -n '[!>]' regular_express.txt
```

这样可以了解了吗？常常看到有陷阱的题目写：『反向选择这样对否？ '[!a-z]' ? 』，呵呵！是错的呦～要 '[^a-z]' 才是对的！至于更多关于正规

表示法的进阶文章，请参考文末的参考数据(注 2)



文件的格式化与相关处理

接下来让我们来将文件进行一些简单的编排吧！底下这些动作可以将你的讯息进行排版的动作，不需要重新以 vim 去编辑，透过数据流重导向配合底下介绍的 printf 功能，以及 awk 指令，就可以让你的讯息以你想要的模样来输出了！试看看吧！



格式化打印：printf

在很多时候，我们可能需要将自己的数据给他格式化输出的！举例来说，试卷分数的输出，姓名与科目及分数之间，总是可以稍微作个比较漂亮的版面配置吧？例如我想要输出底下的样式：

| Name | Chinese | English | Math | Average |
|--------|---------|---------|------|---------|
| DmTsai | 80 | 60 | 92 | 77.33 |
| VBird | 75 | 55 | 80 | 70.00 |
| Ken | 60 | 90 | 70 | 73.33 |

上表的数据主要分成五个字段，各个字段之间可使用 tab 或空格键进行分隔。请将上表的资料转存成为 printf.txt 档名，等一下我们会利用这个档案来进行几个小练习的。因为每个字段的原始数据长度其实并非是如此固定的 (Chinese 长度就是比 Name 要多)，而我就是想要如此表示出这些数据，此时，就得需要打印格式管理员 printf 的帮忙了！printf 可以帮我们将资料输出的结果格式化，而且而支持一些特殊的字符～底下我们就来看看！

```
[root@www ~]# printf '打印格式' 实际内容  
选项与参数：
```

关于格式方面的几个特殊样式：

| | |
|------|----------------------|
| \a | 警告声音输出 |
| \b | 退格键(backspace) |
| \f | 清除屏幕 (form feed) |
| \n | 输出新的一行 |
| \r | 亦即 Enter 按键 |
| \t | 水平的 [tab] 按键 |
| \v | 垂直的 [tab] 按键 |
| \xNN | NN 为两位数的数字，可以转换数字成为字 |

符。

关于 C 程序语言内，常见的变数格式

`%ns` 那个 `n` 是数字，`s` 代表 `string`，亦即多少个字符；

`%ni` 那个 `n` 是数字，`i` 代表 `integer`，亦即多少整数字数；

`%N.nf` 那个 `n` 与 `N` 都是数字，`f` 代表 `floating` (浮点)，如果有小数字数，

假设我共要十个位数，但小数点有两位，即为 `%10.2f` 啰！

接下来我们来进行几个常见的练习。假设所有的数据都是一般文字（这也是最常见的状态），因此最常用来分隔数据的符号就是 [Tab] 啦！因为 [Tab] 按键可以将数据作个整齐的排列！那么如何利用 `printf` 呢？参考底下这个范例：

范例一：将刚刚上头数据的档案 (`printf.txt`) 内容仅列出姓名与成绩：(用 [tab] 分隔)

```
[root@www ~]# printf '%s\t%s\t%s\t%s\t%s\t\n' $(cat printf.txt)
```

| Name | Chinese | English | Math |
|---------|---------|---------|------|
| Average | | | |
| DmTsai | 80 | 60 | 92 |
| VBird | 75 | 55 | 80 |
| Ken | 60 | 90 | 70 |

由于 `printf` 并不是管线命令，因此我们得要透过类似上面的功能，将档案内容先提出来给 `printf` 作为后续的资料才行。如上所示，我们将每个数据都以 [tab] 作为分隔，但是由于 `Chinese` 长度太长，导致 `English` 中间多了一个 [tab] 来将资料排列整齐！啊～结果就看到资料对齐结果的差异了！

另外，在 `printf` 后续的那一段格式中，`%s` 代表一个不固定长度的字符串，而字符串与字符串中间就以 `\t` 这个 [tab] 分隔符来处理！你要记得的是，由于 `\t` 与 `%s` 中间还有空格，因此每个字符串间会有一个 [tab] 与一个空格键的分隔喔！

既然每个字段的长度不固定会造成上述的困扰，那我将每个字段固定就好啦！没错没错！这样想非常好！所以我们就将数据给他进行固定字段长度的设计吧！

范例二：将上述资料关于第二行以后，分别以字符串、整数、小数点来显示：

```
[root@www ~]# printf '%10s %5i %5i %5i %8.2f \n' $(cat printf.txt | \
```

```
> grep -v Name)
    DmTsai    80    60    92    77.33
    VBird     75    55    80    70.00
    Ken       60    90    70    73.33
```

上面这一串格式想必您看得很辛苦！没关系！一个一个来解释！上面的格式共分为五个字段，`%10s` 代表的是一个长度为 10 个字符的字符串字段，`%5i` 代表的是长度为 5 个字符的数字字段，至于那个 `%8.2f` 则代表长度为 8 个字符的具有小数点的字段，其中小数点有两个字符宽度。我们可以使用底下的说明来介绍 `%8.2f` 的意义：

字符宽度：12345678
`%8.2f` 意义：00000.00

如上所述，全部的宽度仅有 8 个字符，整数部分占有 5 个字符，小数点本身（.）占一位，小数点下的位数则有两位。这种格式经常使用于数值程序的设计中！这样了解乎？自己试看看如果要将小数点位数变成 1 位又该如何处理？

`printf` 除了可以格式化处理之外，他还可以依据 ASCII 的数字与图形对应来显示数据喔(注 3)！举例来说 16 进位的 45 可以得到什么 ASCII 的显示图(其实是字符啦)？


```
范例三：列出 16 进位数值 45 代表的字符为何？
[root@www ~]# printf '\x45\n'
E
# 这东西也很好玩～他可以将数值转换成为字符，如果你会
写 script 的话，
# 可以自行测试一下，由 20~80 之间的数值代表的字符是
啥喔！ ^_^
```

`printf` 的使用相当的广泛喔！包括等一下后面会提到的 `awk` 以及在 C 程序语言当中使用的屏幕输出，都是利用 `printf` 呢！鸟哥这里也只是列出一些可能会用到的格式而已，有兴趣的话，可以自行多作一些测试与练习喔！ ^_^

Tips:

打印格式化这个 `printf` 指令，乍看之下好像也没有什么很重要的～不过，如果你需要自行撰写一些软件，需要将一些数据在屏幕上头漂漂亮亮的输出的话，那么 `printf` 可也是一个很棒的工具喔！



 **awk**: 好用的数据处理工具

awk 也是一个非常棒的数据处理工具！相较于 sed 常常作用于一整个行的处理，awk 则比较倾向于一行当中分成数个『字段』来处理。因此，awk 相当的适合处理小型的数据数据处理呢！awk 通常运作的模式是这样的：

```
[root@www ~]# awk '条件类型 1{动作 1} 条件类型 2{动作 2} ...' filename
```

awk 后面接两个单引号并加上大括号 {} 来设定想要对数据进行的处理动作。awk 可以处理后续接的档案，也可以读取来自前个指令的 standard output。但如前面说的，awk 主要是处理『每一行的字段内的数据』，而默认的『字段的分隔符为“空格键”或“[tab]键”』！举例来说，我们用 last 可以将登入者的数据取出来，结果如下所示：

```
[root@www ~]# last -n 5 <==仅取出前五行
root      pts/1    192.168.1.100  Tue Feb 10 11:21
still logged in
root      pts/1    192.168.1.100  Tue Feb 10 00:46 -
02:28    (01:41)
root      pts/1    192.168.1.100  Mon Feb  9 11:41 -
18:30    (06:48)
dmtsai    pts/1    192.168.1.100  Mon Feb  9 11:41 -
11:41    (00:00)
root      tty1          Fri Sep  5 14:09 -
14:10    (00:01)
```

若我想要取出账号与登入者的 IP，且账号与 IP 之间以 [tab] 隔开，则会变成这样：

```
[root@www ~]# last -n 5 | awk '{print $1 "\t" $3}'
root      192.168.1.100
root      192.168.1.100
root      192.168.1.100
dmtsai    192.168.1.100
root      Fri
```

上表是 awk 最常使用的动作！透过 print 的功能将字段数据列出来！字段的分隔则以空格键或 [tab] 按键来隔开。因为不论哪一行我都要处理，因此，就不需要有“条件类型”的限制！我所想要的是第一栏以及第三栏，但是，第五行的内容怪怪的～这是因为数据格式的问题啊！所以啰～使用 awk 的时候，请先确认一下你的数据当中，如果是连续性的数据，请不要有空格或 [tab] 在内，否则，就会像这个例子这样，会发生误判喔！

另外，由上面这个例子你也会知道，在每一行的每个字段都是有变量名称的，那就是 \$1, \$2... 等变量名称。以上面的例子来说，root 是 \$1，因为他是第一栏嘛！至于 192.168.1.100 是第三栏，所以他就是 \$3 啦！后面以此类推～呵呵！还有个变数喔！那就是 \$0，\$0 代表『一整列资料』的意思～以上面的例子来说，第一行的 \$0 代表的就是『root....』那一行啊！由此可知，刚刚上面五行当中，整个 awk 的处理流程是：

1. 读入第一行，并将第一行的资料填入 \$0, \$1, \$2... 等变数当中；
2. 依据“条件类型”的限制，判断是否需要进行后面的“动作”；
3. 做完所有的动作与条件类型；
4. 若还有后续的『行』的数据，则重复上面 1~3 的步骤，直到所有的数据都读完为止。

经过这样的步骤，你会晓得，awk 是『以行为一次处理的单位』，而『以字段为最小的处理单位』。好了，那么 awk 怎么知道我到底这个数据有几行？有几栏呢？这就需要 awk 的内建变量的帮忙啦～

| 变量名称 | 代表意义 |
|------|---------------------|
| NF | 每一行 (\$0) 拥有的字段总数 |
| NR | 目前 awk 所处理的是『第几行』数据 |
| FS | 目前的分隔字符，默认是空格键 |

我们继续以上面 last -n 5 的例子来做说明，如果我想要：

- 列出每一行的账号(就是 \$1)；
- 列出目前处理的行数(就是 awk 内的 NR 变量)
- 并且说明，该行有多少字段(就是 awk 内的 NF 变量)

则可以这样：

Tips:

要注意喔，awk 后续的所有动作是以单引号『 ' 』括住的，由于单引号与双引号都必须是成对的，所以，awk 的格式内容如果想要以 print 打印时，记得非变量的文字部分，包含上一小节 printf 提到的格式中，都需要使用双引号来定义出来喔！因为单引号已经是 awk 的指令固定用法了！



```
[root@www ~]# last -n 5 | awk '{print $1 "\t lines: " NR "\t columes: " NF}'
root      lines: 1      columes: 10
root      lines: 2      columes: 10
```

```
root      lines: 3      colums: 10
dmtsai    lines: 4      colums: 10
root      lines: 5      colums: 9
# 注意喔，在 awk 内的 NR, NF 等变量要用大写，且不需要有钱字号 $ 啦！
```

这样可以了解 NR 与 NF 的差别了吧？好了，底下来谈一谈所谓的“条件类型”了吧！

- awk 的逻辑运算字符

既然有需要用到“条件”的类别，自然就需要一些逻辑运算啰～例如底下这些：

| 运算单元 | 代表意义 |
|------|-------|
| > | 大于 |
| < | 小于 |
| >= | 大于或等于 |
| <= | 小于或等于 |
| == | 等于 |
| != | 不等于 |

值得注意的是那个『 == 』的符号，因为：

- 逻辑运算上面亦即所谓的大于、小于、等于等判断式上面，习惯上是以『 == 』来表示；
- 如果是直接给予一个值，例如变量设定时，就直接使用 = 而已。

好了，我们实际来运用一下逻辑判断吧！举例来说，在 /etc/passwd 当中是以冒号 “:” 来作为字段的分隔，该档案中第一字段为账号，第三字段则是 UID。那假设我要查阅，第三栏小于 10 以下的的数据，并且仅列出账号与第三栏，那么可以这样做：

```
[root@www ~]# cat /etc/passwd | \
> awk ' {FS=":"} $3 < 10 {print $1 "\t" $3}'
root:x:0:0:root:/root:/bin/bash
bin      1
daemon   2
.... (以下省略)....
```


有趣吧！不过，怎么第一行没有正确的显示出来呢？这是因为我们读入第一行的时候，那些变数 \$1, \$2... 默认还是以空格键为分隔的，所以虽然我们定义了 FS=":" 了，但是却仅能在第二行后才开始生效。那么怎么办呢？我们可以预先设定 awk 的变量啊！利用 BEGIN 这个关键词喔！这样做：

```
[root@www ~]# cat /etc/passwd | \
> awk 'BEGIN {FS=":"} $3 < 10 {print $1 "\t " $3}'
root      0
bin        1
daemon     2
..... (以下省略).....
```

很有趣吧！而除了 BEGIN 之外，我们还有 END 呢！另外，如果要用 awk 来进行『计算功能』呢？以底下的例子来看，假设我有一个薪资数据表档名为 pay.txt，内容是这样的：

| Name | 1st | 2nd | 3th |
|--------|-------|-------|-------|
| VBird | 23000 | 24000 | 25000 |
| DMTsai | 21000 | 20000 | 23000 |
| Bird2 | 43000 | 42000 | 41000 |

如何帮我计算每个人的总额呢？而且我还想要格式化输出喔！我们可以这样考虑：

- 第一行只是说明，所以第一行不要进行加总（NR==1 时处理）；
- 第二行以后就会有加总的情况出现（NR>=2 以后处理）

```
[root@www ~]# cat pay.txt | \
> awk 'NR==1{printf
"%10s %10s %10s %10s %10s\n", $1, $2, $3, $4, "Total" }
NR>=2{total = $2 + $3 + $4
printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4,
total}}'
      Name      1st      2nd      3th
Total
      VBird      23000      24000      25000
72000.00
      DMTsai      21000      20000      23000
64000.00
      Bird2      43000      42000      41000
126000.00
```

上面的例子有几个重要事项应该要先说明的：

- awk 的指令间隔：所有 awk 的动作，亦即在 {} 内的动作，如果有需要多个指令辅助时，可利用分号『;』间隔，或者直接以 [Enter] 按键来隔开每个指令，例如上面的范例中，鸟哥共按了三次 [enter] 喔！
- 逻辑运算当中，如果是『等于』的情况，则务必使用两个等号『==』！
- 格式化输出时，在 printf 的格式设定当中，务必加上 \n ，才能进行分行！
- 与 bash shell 的变量不同，在 awk 当中，变量可以直接使用，不需加上 \$ 符号。

利用 awk 这个玩意儿，就可以帮我们处理很多日常工作了呢！真是好用的很～此外，awk 的输出格式当中，常常会以 `printf` 来辅助，所以，最好你对 `printf` 也稍微熟悉一下比较好啦！另外，awk 的动作内 {} 也是支持 if (条件) 的喔！举例来说，上面的指令可以修订成为这样：

```
[root@www ~]# cat pay.txt | \
> awk ' {if(NR==1) printf
"%10s %10s %10s %10s %10s\n", $1, $2, $3, $4, "Total"}
NR>=2{total = $2 + $3 + $4
printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4,
total}
'
```

你可以仔细的比对一下上面两个输入有啥不同～从中去了解两种语法吧！我个人是比较倾向于使用第一种语法，因为会比较有统一性啊！ ^_^

除此之外，awk 还可以帮我们进行循环计算喔！真是相当的好用！不过，那属于比较进阶的单独课程了，我们这里就不再多加介绍。如果你有兴趣的话，请务必参考延伸阅读中的相关连结喔（[注 4](#)）。

档案比对工具

什么时候会用到档案的比对啊？通常是『同一个软件包的不同版本之间，比较配置文件与原始档的差异』。很多时候所谓的档案比对，通常是用在 ASCII 纯文本档的比对上的！那么比对档案的指令有哪些？最常见的就是 `diff` 啰！另外，除了 `diff` 比对之外，我们还可以藉由 `cmp` 来比对非纯文本档！同时，也能够藉由 `diff` 建立的分析档，以处理补丁 (patch) 功能的档案呢！就来玩玩先！

-
- `diff`

diff 就是用在比对两个档案之间的差异的，并且是以行为单为来比对的！一般是用在 ASCII 纯文本档的比对上。由于是以行为比对的单位，因此 diff 通常是用在同一的档案(或软件)的新旧版本差异上！举例来说，假如我们要将 /etc/passwd 处理成为一个新的版本，处理方式为：将第四行删除，第六行则取代成为『no six line』，新的档案放置到 /tmp/test 里面，那么应该怎么做？

```
[root@www ~]# mkdir -p /tmp/test <==先建立测试用的目录
[root@www ~]# cd /tmp/test
[root@www test]# cp /etc/passwd passwd.old
[root@www test]# cat /etc/passwd | \
> sed -e '4d' -e '6c no six line' > passwd.new
# 注意一下， sed 后面如果要接超过两个以上的动作时，
每个动作前面得加 -e 才行！
# 透过这个动作，在 /tmp/test 里面便有新旧的 passwd
档案存在了！
```

接下来讨论一下关于 diff 的用法吧！

```
[root@www ~]# diff [-bBi] from-file to-file
选项与参数：
from-file : 一个档名，作为原始比对档案的档名；
to-file   : 一个档名，作为目的比对档案的档名；
注意，from-file 或 to-file 可以 - 取代，那个 - 代表
『Standard input』之意。

-b  : 忽略一行当中，仅有多余空白的差异(例如 "about me"
与 "about    me" 视为相同)
-B  : 忽略空白行的差异。
-i  : 忽略大小写的不同。

范例一：比对 passwd.old 与 passwd.new 的差异：
[root@www test]# diff passwd.old passwd.new
4d3    <==左边第四行被删除 (d) 掉了，基准是右边的第三行
< adm:x:3:4:adm:/var/adm:/sbin/nologin <==这边列出
左边(<)档案被删除的那一行内容
6c5    <==左边档案的第六行被取代 (c) 成右边档案的第五行
< sync:x:5:0:sync:/sbin:/bin/sync <==左边(<)档案第六
行内容
---
```

```
> no six line                                <==右边(>)档案第
五行内容
# 很聪明吧！用 diff 就把我们刚刚的处理给比对完毕了！
```

用 diff 比对档案真的是很简单喔！不过，你不要用 diff 去比对两个完全不相干的档案，因为比不出个啥咚咚！另外，diff 也可以比对整个目录下的差异喔！举例来说，我们想要了解一下不同的开机执行等级 (runlevel) 内容有啥不同？假设你已经知道执行等级 3 与 5 的启动脚本分别放置到 /etc/rc3.d 及 /etc/rc5.d，则我们可以将两个目录比对一下：

```
[root@www ~]# diff /etc/rc3.d/ /etc/rc5.d/
Only in /etc/rc3.d/: K99readahead_later
Only in /etc/rc5.d/: S96readahead_later
```

我们的 diff 很聪明吧！还可以比对不同目录下的相同文件名的内容，这样真的很方便喔～

- cmp

相对于 diff 的广泛用途，cmp 似乎就用的没有这么多了～cmp 主要也是在比对两个档案，他主要利用『字节』单位去比对，因此，当然也可以比对 binary file 啰～(还是要再提醒喔，diff 主要是以『行』为单位比对，cmp 则是以『字节』为单位去比对，这并不相同！)

```
[root@www ~]# cmp [-s] file1 file2
选项与参数：
-s  ： 将所有不同点的字节处都列出来。因为 cmp 预设
仅会输出第一个发现的不同点。

范例一：用 cmp 比较一下 passwd.old 及 passwd.new
[root@www test]# cmp passwd.old passwd.new
passwd.old passwd.new differ: byte 106, line 4
```

看到了吗？第一个发现的不同点在第四行，而且字节数是在第 106 个字节处！这个 cmp 也可以用来比对 binary 啦！^_^

- patch

patch 这个指令与 diff 可是有密不可分的关系啊！我们前面提到，diff 可以

用来分辨两个版本之间的差异，举例来说，刚刚我们所建立的 passwd.old 及 passwd.new 之间就是两个不同版本的档案。那么，如果要『升级』呢？就是『将旧的档案升级成为新的档案』时，应该要怎么做呢？其实也不难啦！就是『先比较新旧版本的差异，并将差异档制作成为补丁档，再由补丁档更新旧档案』即可。举例来说，我们可以这样做测试：

```
范例一：以 /tmp/test 内的 passwd.old 与 passwd.new
制作补丁档案
[root@www test]# diff -Naur passwd.old passwd.new >
passwd.patch
[root@www test]# cat passwd.patch
--- passwd.old  2009-02-10 14:29:09.000000000 +0800
<==新旧档案的信息
+++ passwd.new  2009-02-10 14:29:18.000000000 +0800
@@ -1,9 +1,8 @@    <==新旧档案要修改数据的界定范围，
旧档在 1-9 行，新档在 1-8 行
  root:x:0:0:root:/root:/bin/bash
  bin:x:1:1:bin:/bin:/sbin/nologin
  daemon:x:2:2:daemon:/sbin:/sbin/nologin
-adm:x:3:4:adm:/var/adm:/sbin/nologin      <==左侧
档案删除
  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
-sync:x:5:0:sync:/sbin:/bin/sync          <==左侧
档案删除
+no six line                             <==右侧新
档加入
  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
  halt:x:7:0:halt:/sbin:/sbin/halt
  mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
```

一般来说，使用 diff 制作出来的比较档案通常使用扩展名为 .patch 啰。至于内容就如同上面介绍的样子。基本上就是以行为单位，看看哪边有一样与不一样的，找到一样的地方，然后将不一样的地方取代掉！以上面表格为例，新档案看到 - 会删除，看到 + 会加入！好了，那么如何将旧的档案更新成为新的内容呢？就是将 passwd.old 改成与 passwd.new 相同！可以这样做：

```
[root@www ~]# patch -pN < patch_file    <==更新
[root@www ~]# patch -R -pN < patch_file <==还原
选项与参数：
-p   : 后面可以接『取消几层目录』的意思。
-R   : 代表还原，将新的文件还原成原来旧的版本。
```

范例二：将刚刚制作出来的 patch file 用来更新旧版数据

```

[root@www test]# patch -p0 < passwd.patch
patching file passwd.old
[root@www test]# ll passwd*
-rw-r--r-- 1 root root 1929 Feb 10 14:29 passwd.new
-rw-r--r-- 1 root root 1929 Feb 10 15:12 passwd.old <==
档案一模一样！

范例三：恢复旧档案的内容
[root@www test]# patch -R -p0 < passwd.patch
[root@www test]# ll passwd*
-rw-r--r-- 1 root root 1929 Feb 10 14:29 passwd.new
-rw-r--r-- 1 root root 1986 Feb 10 15:18 passwd.old
# 档案就这样恢复成为旧版本啰

```

为什么这里会使用 `-p0` 呢？因为我们在比对新旧版的数据时是在同一个目录下，因此不需要减去目录啦！如果是使用整体目录比对（`diff` 旧目录 新目录）时，就得要依据建立 `patch` 档案所在目录来进行目录的删减啰！

更详细的 `patch` 用法我们会在后续的第五篇的[原始码编译](#)再跟大家介绍，这里仅是介绍给你，我们可以利用 `diff` 来比对两个档案之间的差异，更可进一步利用这个功能来制作修补档案（`patch file`），让大家更容易进行比对与升级呢！很不赖吧！ ^_^

💡档案打印准备： `pr`

如果你曾经使用过一些图形接口的文字处理软件的话，那么很容易发现，当我们在打印的时候，可以同时选择与设定每一页打印时的标头吧！也可以设定页码呢！那么，如果我是在 Linux 底下打印纯文本档呢 可不可以具有标题啊？可不可以加入页码啊？呵呵！当然可以啊！使用 `pr` 就能够达到这个功能了。不过，`pr` 的参数实在太多了，鸟哥也说不完，一般来说，鸟哥都仅使用最简单的方式来处理而已。举例来说，如果想要打印 `/etc/man.config` 呢？

```

[root@www ~]# pr /etc/man.config

2007-01-06 18:24                               /etc/man.config
Page 1

#
# Generated automatically from man.conf.in by the

```

```
# configure script.  
..... 以下省略.....
```

上面特殊字体那一行呢，其实就是使用 `pr` 处理后所造成的标题啦！标题中会有『档案时间』、『档案档名』及『页码』三大项目。更多的 `pr` 使用，请参考 `pr` 的说明啊！ ^_^



重点回顾

- 正规表示法就是处理字符串的方法，他是以行为单位来进行字符串的处理行为；
- 正规表示法透过一些特殊符号的辅助，可以让使用者轻易的达到『搜寻/删除/取代』某特定字符串的处理程序；
- 只要工具程序支持正规表示法，那么该工具程序就可以用来作为正规表示法的字符串处理之用；
- 正规表示法与通配符是完全不一样的东西！通配符 (wildcard) 代表的是 `bash` 操作接口的一个功能，但正规表示法则是一种字符串处理的表示方式！
- 使用 `grep` 或其他工具进行正规表示法的字符串比对时，因为编码的问题会有不同的状态，因此，你最好将 `LANG` 等变量设定为 `C` 或者是 `en` 等英文语系！
- `grep` 与 `egrep` 在正规表示法里面是很常见的两支程序，其中，`egrep` 支持更严谨的正规表示法的语法；
- 由于编码系统的不同，不同的语系 (`LANG`) 会造成正规表示法撷取资料的差异。因此可利用特殊符号如 `[:upper:]` 来替代编码范围较佳；
- 由于严谨度的不同，正规表示法之上还有更严谨的延伸正规表示法；
- 基础正规表示法的特殊字符有： `*`, `?`, `[]`, `[-]`, `[^]`, `^`, `$` 等！
- 常见的正规表示法工具有：`grep` , `sed`, `vim` 等等
- `printf` 可以透过一些特殊符号来将数据进行格式化输出；
- `awk` 可以使用『字段』为依据，进行数据的重新整理与输出；
- 文件的比对中，可利用 `diff` 及 `cmp` 进行比对，其中 `diff` 主要用在纯文本档案方面的新旧版本比对
- `patch` 指令可以将旧版数据更新到新版（主要亦由 `diff` 建立 `patch` 的补丁来源档案）



本章习题

（ 要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看 ）

- 我想要知道某个档案里面含有 `boot` 的字眼，而这个档案在 `/etc/` 底

下，我要如何找出这个档案？

既然知道有这个字眼那就好办了！可以直接下达：

```
grep boot /etc/*
```

- 我想知道，在 /etc 底下，只要含有 XYZ 三个字符的任何一个字符的那一行就列出来，要怎样进行？

```
grep [XYZ] /etc/*
```

- 我想要找出在 /etc 底下，档案内容含有 * 的文件名？

由于 * 是特殊字符，在变量的订定法则里面曾经提过要将特殊字符移除，需要使用跳脱字符，亦即是 \ 符号，所以我可以这样下达指令：

```
grep \* /etc/*
```



参考数据与延伸阅读

- 注 1：关于正规表示法与 POSIX 及特殊语法的参考网址可以查询底下的来源：
维基百科的说明：http://en.wikipedia.org/wiki/Regular_expression
ZYTRAX 网站介绍：<http://zytrax.com/tech/web/regex.htm>
- 注 2：其他关于正规表示法的网站介绍：
洪朝贵老师的网页：<http://www.cyut.edu.tw/~ckhung/b/re/index.php>
龙门少尉的窝：<http://main.rtfiber.com.tw/~changyj/>
PCRE 官方网站：<http://perldoc.perl.org/perlre.html>
- 注 3：关于 ASCII 编码对照表可参考维基百科的介绍：
维基百科 (ASCII) 条目：
<http://zh.wikipedia.org/w/index.php?title=ASCII&variant=zh-tw>
- 注 4：关于 awk 的进阶文献，包括有底下几个连结：
中研院计算中心 ASPAC 计划之 awk 程序介绍：
<http://phi.sinica.edu.tw/aspac/reports/94/94011/>
鸟哥备份：
http://linux.vbird.org/linux_basic/0330regularex/awk.pdf
这份文件写的非常棒！欢迎大家多多参考！
Study Area：
http://www.study-area.org/linux/system/linux_shell.htm

2002/07/29：第一次完成；

2003/02/10：重新编排与加入 FAQ ；

2005/01/28: 重新汇整基础正规表示法的内容! 重点在 regular_express.txt 的处理与练习上!

2005/03/30: 修订了 `grep -n 'goo*g' regular_express.txt` 这一段

2005/05/23: 修订了 `grep -n '^[a-z]' regular_express.txt` 所要撷取的是小写, 之前写成大写, 错了!

2005/08/22: 加入了 `awk`, `sed` 等工具的介绍, 还有 `diff` 与 `cmp` 等指令的说明!

2005/09/05: 加入 `printf` 内, 关于 `\xNN` 的说明!

2006/03/10: 将原本的 `sed` 内的动作(action)中, `s` 由『搜寻』改成『取代』了!

2006/10/05: 在 `sed` 当中多了一个 `-i` 的参数说明, 也多了一个范例八可以参考。感谢讨论区的 thyme 兄!

2008/10/08: 加入 `grep` 内的 `--color=auto` 说明!

2009/02/07: 将旧的基于 FC4 版本的文章移动到[此处](#)

2009/02/10: 重新排版, 并且加入[语系](#)的说明, 以及特殊 `[:资料:]` 的说明! 更改不少范例的说明。

2009/05/14: 感谢网友 Jack 的回报, `cmp` 应该是使用『字节 bytes』而非位 bits, 感谢 Jack 兄。

2002/06/28 以来统计人数

374110



本网页主要以 [firefox](#) 配合分辨率 1024x768 作为设计依据

<http://linux.vbird.org> is designed by [VBird](#) during 2001-2009. [Aerosol Lab.](#)