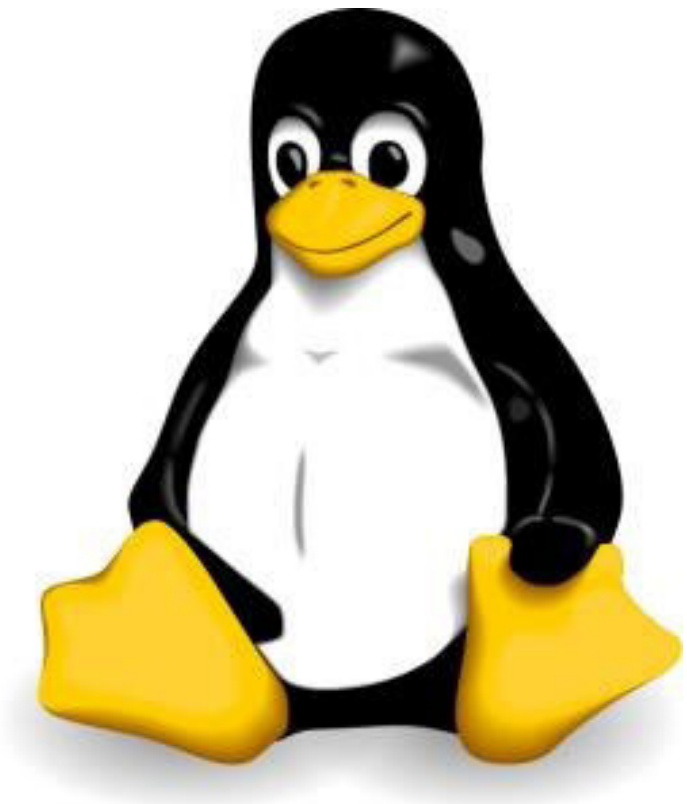
Shell Scripting

李敬、甘**、杨*、杨*、李* No.016 2017.3.27

一些有趣的东东



!=


QQ

An Update is Available For Your Computer

Linux

从入门到放弃

12

Linux

物联网 云计算 大数据
幕后鲜为人知的hero

主流的软件开发环境都是在Linux操作系统上

主流的服务器操作系统都采用Linux

网站：（淘宝、京东；优酷、爱奇艺、百度、新浪、网易）

数据库：（oracle、MySQL、DB2）

网络游戏：(dota刀塔、lol英雄联盟、剑侠情缘)

即时通讯：（QQ、微信、陌陌）

主流的互联网应用都基于Linux平台

打造物联网时代全能运维工程师

下一个互联网十年

云计算成制胜法宝

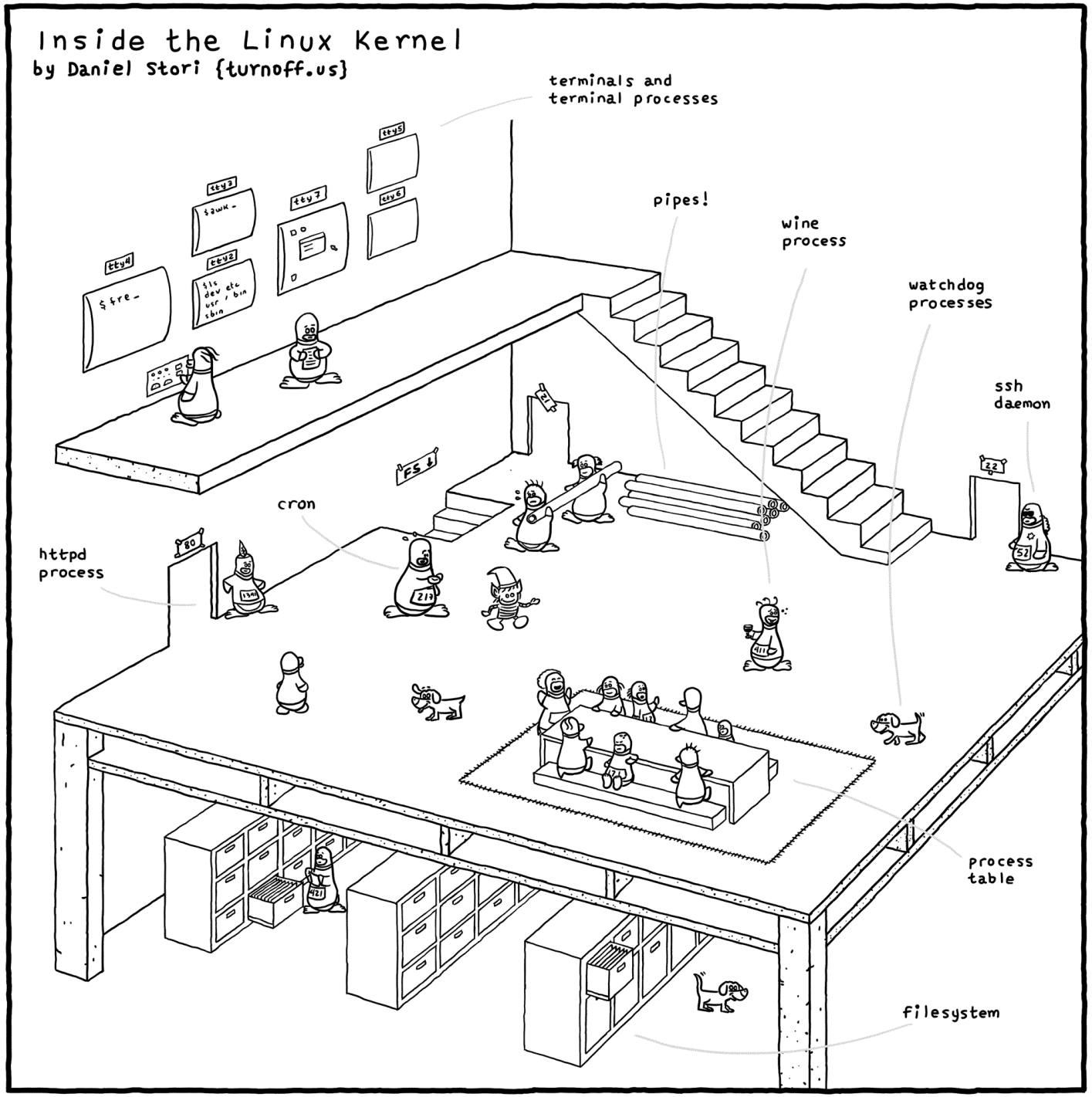
互联网的第一个时代我们定义为PC互联网，互联网的第二个时代我们定义为移动互联网，而互联网的第三个时代我们则定义为万物联网。

当前国内的互联网正处于第二个时代向第三个时代过渡期，而云计算则是支撑起万物联网的基石所在。在互联网的第三个时代，也就是下一个互联网十年里，云计算将成为这场大战的制胜关键所在。

学Linux系统架构

Linux

做大数据云时代的王者



这幅漫画是以一个房子的侧方刨面图来绘画的。使用这样的一个房子来代表 Linux 内核。

地基（底层）由一排排的文件柜组成，井然有序，文件柜里放置着“文件”——电脑中的文件。左上角，有一只胸前挂着 421 号牌的小企鹅，它表示着 PID（进程 IDProcess ID） 为 421 的进程，它正在查看文件柜中的文件，这代表系统中正有一个进程在访问文件系统。在右下角有一只小狗，它是看门狗watchdog ，这代表对文件系统的监控。

地面一层，最引人瞩目的莫过于中间的一块垫子，众多小企鹅在围着着桌子坐着。这个垫子的区域代表进程表。
左上角有一个小企鹅，站着，仿佛在说些什么这显然是一位家长式的人物，不过看起来周围坐的那些小企鹅不是很听话——你看有好多走神、自顾自聊天的——“喂喂，说你呢，哇塞娃（171），转过身来”。它代表着 Linux 内核中的初始化（init）进程，也就是我们常说的 PID 为 1 的进程。桌子上坐的小企鹅都在等待状态wait中，等待工作任务。

源于企业真实需求

十大实战项目

4个月 = 两年一线工作经验

01 PXE全自动批量装机平台

02 构建CDN分发网络

03 KVM虚拟化平台解决方案

04 ExtMail企业邮件系统集成

05 MySQL高可用集群

06 企业运维常用Shell脚本

07 基于Nginx+Tomcat的高效Java平台

08 Cacti+Nagios服务器群监控方案

09 IP-SAN网络存储解决方案

10 构建千万并发量的高可用集群

项目6 企业运维常用Shell脚本

项目描述：

在管理大量服务器的时候，经常会遇到一些固定、重复但是又复杂的任务，要是把这些任务操作分别写成脚本，以后再有类似任务就省事多了，一次就能节省好几个小时甚至好几天的时间。如何运用专业技能"偷懒"？

项目内容：

批量添加/删除各种用户帐号；
服务状态监控；
服务优化及安全加固；
SSH自动化登录及软件自动分发；
Web访问量分析及统计

```
$ mkdir mydir
$ mv mydir yourdir
$ cd yourdir
```

可以改成:

```
$ mkdir mydir
$ mv !$ yourdir
$ cd !$
```

!$是一个特殊的环境变量，它代表了上一个命令的最后一个字符串。

```
$ var=1
$ var=$var+1
$ echo $var

$ expr var+1
$ expr var + 1
$ expr $var + 1
$ expr $var +1
$ echo $var

$ let var=2+1
$ echo $var
```

```
1+1
var+1
non-interger argument
non-interger argument
syntax error
1+1
3
```

```
hello=hello
$ echo 'hello''world'
$ echo 'hello  world'
$ echo '$hello'$world
$ echo 'hello
> world'
$ echo *
$ echo '*'
$ echo “hello” ” world”
$ echo “hello    world”
$ echo 'hello “world” '
$ echo “hello 'world'”
$ echo “$hello 'world'”
$ echo `helloworld`
$ echo `$hello`
```

单引号防止shell解释其中的内容
双引号忽略大多数，但不包括$、\、`
反引号命令替换

```
$ set 1 2 3 four five six 7 8 9 ten
$ echo “$1 $2 $3 $4 $5 $6 $7 $8 $9 ${10}”
$ echo “$*”
$ echo $#
$ echo $?
$ echo $$
$ echo $!
$ ehco $0
```

多于一个数字的位置参数要放在大括号

特殊参数*，将扩展为从1开始的所有位置参数
特殊参数@，也是，但当它的扩展发生在双引号内时，每个参数都扩展为分隔的单词
特殊参数？，将扩展为最近一个在前台执行的命令的退出状态
特殊参数-，将扩展为当前选项标志
特殊参数$，将扩展为当前Shell进程号
特殊参数！，将扩展为最近一次执行后台命令的进程号
特殊参数0，将扩展为shell或shell脚本名
特殊参数_，在shell启动时设置为开始运行的shell或shell脚本路径，随后扩展为前一个命令的最后一个参数

先入为主

一个脚本

the only way to really learn scripting is to write scripts

操千曲而后晓声 观千剑而后识器

UNIX Philosophy

K.I.S.S
keep it simple, stupid

小即是美

complex projects into
simpler subtasks

杠杆效应和可移植性

使用shell script来提高

```
echo `who | awk ‘{print $1}’ | sort | uniq` |
sed ‘s/ /, /g’
```

echo `who | awk ‘{print $1}’ | sort | uniq` | sed ‘s/ /, /g’

短短一行，6种可执行命令

同步平行执行

管道传输

协同工作

who 将系统上用户名单以多列形式显示

awk 保留第一列包含用户名的输入结果，舍弃其他数据

sort 按字母进行排序

uniq 舍弃用户同一时间登陆多个会话窗口而重复的结

```
echo `who | awk  '{print $1}'  | sort | uniq` | sed
                 's/ /, /g'
```

命令名	行数
echo	177
who	755
awk	3412
sort	2614
uniq	302
sed	2093
合计	9353

不要重复造轮子

Stop Trying to Reinvent the Wheel

function 函数

sed & grep 正则表达式

Debug Shell 调试

一些补充

重点回顾

定义整型变量：

let VAR_NAME=INTEGER_VALUE

例如：let a=3

eclare -i VAR_NAME=INTEGER_VALUE

例如：declare -i a=3

提取子字符串

```
string="yzu is a great school"
echo ${string:1:5} #输出zu is
```

实现算术运算的方式：自增举例

```
m=$[ m + 1]
m=`expr $m + 1`
m=$(($m + 1))
let m=m+1
```

环境变量

PATH	决定了shell将到哪些目录中寻找命令或程序
HOME	当前用户主目录
PS1	基本提示符，对于root用户是#，对于普通用户是$

命令都有其状态返回值：

成功：0，真

失败：1-255, 假

command > file	将输出重定向到 file。
command < file	将输入重定向到 file。
command >> file	将输出以追加的方式重定向到 file。

菜单

举个栗子

函数

- bash机制：将一系列命令映射为函数
- 简化
- 清晰
- 可维护
- 高可用

定义 ——函数的定义格式，调用函数的方法

参数 ——函数如何处理参数，如何输出数据

变量 ——函数中变量的作用域

调用 ——如何递归调用函数，如何创建自己的函数库

函数定义

```
[ function ] funname [()]
{
action;
[return int;]
}
```

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

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

3、注意，()内是没有参数的，它并不像C语言那样，在()里可以有参数。

向函数传递参数

```
[~/shell/function]# cat ./testfun1.sh
#!/bin/bash
if [ $# -ne 3 ]
then
   echo "usage: $0 a b c"
   exit
fi
fun3() {
   echo $[ $1 * $2 * $3 ]
}
result=`fun3 $1 $2 $3`
echo the result is $result
[~/shell/function]# ./testfun1.sh  1 2 3
the result is 6
[~/shell/function]# ./testfun1.sh  1 2
usage: ./testfun1.sh a b c
```

```
[~/shell/function]# cat array.sh
#!/bin/bash
a=(11 12 13 14 15)
echo ${a[*]}
function array(){
  echo parameters : "$@"
  local factorial=1
  for value in "$@"
  do
    factorial=$[ $factorial * $value ]
  done
  echo $factorial
}
array ${a[*]}
[~/shell/function]# ./array.sh
11 12 13 14 15
parameters : 11 12 13 14 15
360360
```

函数中变量作用域

```
#!/bin/sh
echo $(uname);
declare num=1000;
uname()
{
  echo "test!";
  ((num++));
  return 100;
}
testvar()
{
  local num=10;
  ((num++));
  echo $num;
}
uname;
echo $?
echo $num;
testvar;
echo $num;
```

```
sh testfun2.sh
Linux
test!
100
1001
11
1001
```

1、定义函数可以与系统命令相同，说明shell搜索命令时候，首先会在当前的shell文件定义好的地方查找，找到直接执行。

2、需要获得函数值：通过$?获得

3、如果需要传出其它类型函数值，可以在函数调用之前，定义变量（这个就是全局变量）。在函数内部就可以直接修改，然后在执行函数就可以读出修改过的值。

4、如果需要定义自己变量，可以在函数中定义：local 变量=值，这时变量就是内部变量，它的修改，不会影响函数外部相同变量的值 。

递归调用

```
#!/bin/bash
function factorial
{
 if  [ $1 -eq 1 ]
 then
   echo 1
 else
   local temp=$[ $1 -1 ]
   local result=`factorial $temp`
   echo $[ $result * $1 ]
 fi
}
read -p "Enter value:" value
result=`factorial $value`
echo "The factorial of $value is:$result"
```

（输入5，输出120。大环套小环，小环套更小，逐步执行。
我们一步一步剖析
5输入得到5*4!
4又得到4*3！
3又得到3*2！
2又得到2*1
由此得到5*4*3*2*1也就是120

库的创建和访问

不是真正意义上的库

脚本调用其他文件，获取其内容，而不用运行任何实际代码，使得这些文件中的函数供我们使用

func.sh:

```
#!/bin/bash

printHello()
{
        printf "hello\n"
    # exit 0
}

printWorld()
{
        printf "world\n"
    # exit 0
}
```

testlib.sh:

```
#!/bin/bash

. func.sh #装载函数库

printHello
printWorld
```

过滤器

函数的作用是从传递给它的参数接受数据
然后输出结果

文本处理三剑客

- grep： 文本过滤工具
- sed ： stream editor，文本编辑工具
- awk ： Linux上的实现gawk，文本报告生成器

80/20原则

sed

- 非交互式的面向字符流的编辑器
- 实现文档一致性
- 自动化编辑多个文件
- 简化在多个文件中执行相同的编辑任务
- 编写转换程序

sed语法及常用选项

sed [OPTIONS]... 'COMMAND' [FILE]... #在命令行执行sed指令
sed [OPTIONS] -f SCRIPTFILE [FILE]... #将sed指令放入一个文件中并将其文件名作为参数

- -e command,--expression=command允许多台编辑。
- -h,--help打印帮助，并显示bug列表的地址。
- -n,--quiet,--silent取消默认输出。
- -f,--filer=script-file引导sed脚本文件名。
- -V,--version打印版本和版权信息。

sed实例

删除：d命令

$ sed '2,$d' example-----删除example文件的第二行到末尾所有行。

替换：s命令

$ sed 's/test/mytest/g' example-----在整行范围内把test替换为mytest。如果没有g标记，则只有每行第一个匹配的test被替换成mytest。

选定行的范围：逗号

$ sed -n '5,/^test/p' example-----打印从第五行开始到第一个包含以test开始的行之间的所有行。

多点编辑：e命令

$ sed -e '1,5d' -e 's/test/check/' example-----允许在同一行里执行多条命令。如该例，第一条命令删除1至5行，第二条命令用check替换test。命令的执行顺序对结果有影响。如果两个命令都是替换命令，那么第一个替换命令将影响第二个替换命令的结果。

写入文件：w命令

$ sed -n '/test/w file' example-----在example中所有包含test的行都被写入file里。

退出：q命令

插入：i命令

awk

- 数据提取和报告工具的解释性程序设计语言
- Alfred Aho、Peter Weinberger、Brian Kernighan
- 数据驱动
- 与sed类似
- 具有算术和字符串操作符
- 具有普通的程序设计结构

awk语法及常用选项

awk运行方式：
awk [OPTIONS] [--] program-text file ... # awk程序短直接写在命令行
awk [OPTIONS] -f program-file [--] file ... # awk程序长放在一个文件中访问

awk程序语法：
pattern { action } # 一个awk指令由一个模式后跟一个动作组成
pattern { action } # 动作与模式分隔，每个awk指令间常用换行符分隔
...
一个完整的awk语句为：Awk '[patten]{action}……', 其中pattern缺省为1，action缺省为{print}。

- -F fs 指定用于输入数据的列分隔符fs
- -v var=value 执行前指定变量，其值用于BEGIN块
- -f program-file 指定awk程序文件
- -- 表示命令行选项的结束

awk实例

```
#!/bin/bash
awk  -F":"  '{print $6}'  passwd
```

passwd内容：

```
f_byte1:f_byte2:f_byte3:f_byte4:f_byte5:f_byte6
s_byte1:s_byte2:s_byte3:s_byte4:s_byte5:s_byte6
```

运行脚本得到结果：

```
f_byte6
s_byte6
```

分析结果：

由passwd作为输入文件，以”:”作为分隔符，{print $6}输出第六个字段。

其中在{print $6}语句中可以选择输出多个字段。如{print $5 $6}则会输出passwd文件中的第5、第6字段。

正则表达式

- globbing 通配符
- 匹配一组字符串的模式
- 普通字符和元字符组成的字符集
- 文本搜索
- 字符串处理

基本正则表达式的元字符 BRE

- 星号*：匹配它前面字符串或正则表达式任意次
- 句点.：匹配除换行符外任意一个字符
- 插入符^：匹配一行开始，有时根据上下文表否定
- 美元符$：在正则表达式的末尾，匹配一行的结尾
- 方括号[ ]：匹配方括号内指定字符集的一个字符
- 反斜线\：转义特殊字符
- 转义尖括号\<\>：标记单词边界

1122* 将匹配112 + 任意个2，可以匹配112、11222、112233

112. 将匹配112 + 至少一个字符，可以匹配1121、11223、112a

^abc 将只匹配行首的abc字符串

123$ 将只匹配行尾的123
^$ 将只匹配一个空行

[abc] 将匹配abc中任意一个字符
[a-h] [A-Z][a-z]
[^a-d] 将匹配a-d之外的所有字符

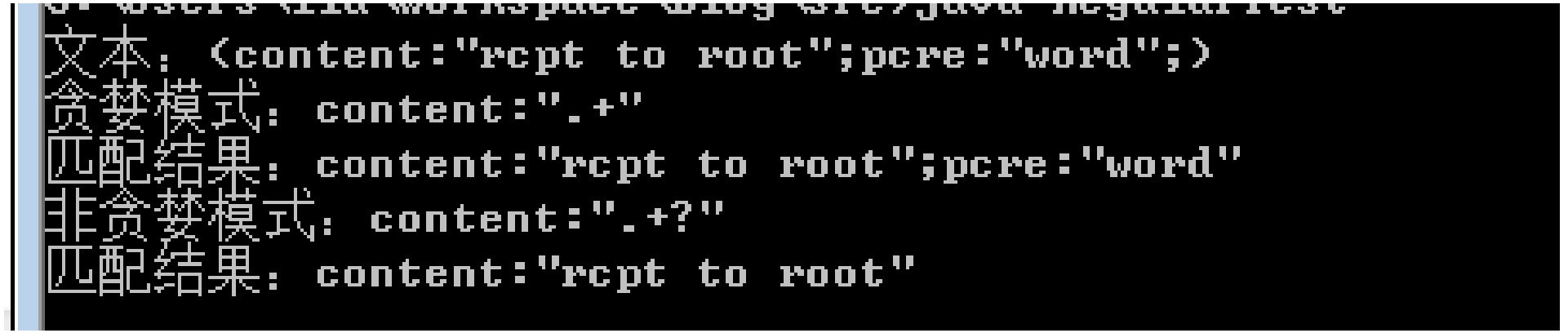
\<the\> 匹配单词the，但不匹配them、there、other等

扩展正则表达式的元字符 ERE

- 问号?：匹配0个或1个前面的字符　　ab?c 将匹配ac 或 abc
- 加号+：匹配1个或多个前面的字符　　ab+c 将匹配abc、abbc、abb.....c
- 转义花括号\{\}：匹配前面正则表达式的次数　　[0-9]\{5\} 匹配5位数字
- 圆括号( )：包含一组正则表达式　　与竖线一起使用或expr提取字符串用
- 竖线|：或操作　　a(b|c)d 匹配abd 或 acd

贪婪与懒惰

- 贪婪模式：选择匹配表达式的可能的最长模式
- 非贪婪模式：选择匹配表达式的可能的最短模式



grep

- Global search Regular Expression and Print out the line.
- 根据指定的“模式”对文本逐行进行匹配
- 模式：由正则表达式编写的过滤条件

grep语法

grep -i pattern files：不区分大小写地搜索。默认情况区分大小写

grep -l pattern files ：只列出匹配的文件名

grep -L pattern files ：列出不匹配的文件名

grep -w pattern files：只匹配整个单词，而不是字符串的一部分（如匹配‘magic’，而不是‘magical’）

grep -E pattern files：支持ERE

grep -r pattern files：递归搜索，不仅搜索当前工作目录，而且搜索子目录

grep实例

找出合法邮件的地址

新建文本example3.grep如下:

```
jack@qq.com
Mary@qq.com
Mike.li@qq.com
snow_chen@qq.coim
jack_2@hotmail.com
jack_3@qq.com
jack@.com
13345678921@qq.com
aaa@gmail.com
a@bc@qq.com
@@baidu.com
_abc@qq.com
.@qq.comi
abc+abc@qq.com
```

找出所有合规的邮件地址

```
test@sha> grep -i -E'^[a-zA-Z0-9_]+[a-z.]*@[a-zA-Z0-9]+\..*'
example3.grep
jack@qq.com
Mary@qq.com
Mike.li@qq.com
snow_chen@qq.coim
jack_2@hotmail.com
jack_3@qq.com
13345678921@qq.com
aaa@gmail.com
_abc@qq.com
```

找出所有不合规的邮件地址

```
test@sha> grep -i -E '^[^a-zA-Z0-9_]*@'example3.grep
@@baidu.com
.@qq.comi
```

Debug

- 发现引发脚本错误的原因
- 定位
- 最简单——echo

看懂脚本输出的错误信息，逐步定位

检查语法错误并通过track模式定位

debug的一般步骤

如何在script中添加DEBUG支持

常见错误信息

- syntax error:unexpected end of file

可能出现在if语句没有fi结束，或for和while等循环语句和case语句中，检查do/done及esac等关键字

google + experience

debug模式

模式	选项	描述
语法检查	-n	不执行命令，仅语法检查
vebose	-v	打印shell读取的所有语句
trace	-x	打印替换后shell实际执行语句

Bash –x选项启动子Shell以调试模式运行整个脚本

- 执行脚本过程中显示实际执行的每个命令
- 行首显示 + 号
- + 号后面显示经过参数扩展后的命令行内容
- 用set -(+)选项对某段脚本进行调试

结合Bash特性增强调试输出信息

- Bash -v 激活详细输出模式（与-x 共用更好）
- 借助内部环境变量
- $ export PS4=’ + {$LINENO:${FUNCNAME[0]}} ’
- 由 + 修改为 +{行号:函数名数组变量}

编程风格

运维标准 + 优雅

每个代码行不多于80个字符

保持一致的缩进

文件头文档注释

自定义变量名函数名小写

使用 _ 分隔单词

返回值用变量$?验证

常用脚本实例

1、设计一个shell程序，添加一个新组为class1，然后添加属于这个组的30个用户，用户名的形式为stdxx，其中xx从01到30。 # useradd2.sh

2、递归求阶乘 # testfun3.sh # testfun4.sh

3、编写shell程序，实现自动删除50个账号的功能。账号名为stud1至stud50。 # deluser.sh

4、设计一个Shell程序，在/userdata目录下建立50个目录，即user1～user50，并设置每个目录的权限，其中其他用户的权限为：读；文件所有者的权限为：读、写、执行；文件所有者所在组的权限为：读、执行。 # crazymakedir.sh

5、求100以内偶数之和 # sumeven1.sh # sumeven2.sh

Q&A