Linux & Shell 培训材料

Version 3.0 By 邓强 On 2009/12/15

功能项	命令实例	作用	重要性
Linux 路径说明	/	表示根目录，是绝对路径	高
	./	表示当前目录，是相对路径	高
	../	表示上一级目录，是相对路径	高
	/root	root用户的主目录	高
	/home/username	存放普通用户的个人配置文件	低
	/bin	存放linux常用的命令	中
	/boot	存放系统启动时要用到的文件	低
	/dev	存放linux系统中使用的外部设备	低
	/etc	存放系统管理时用到的配置文件和子目录	中
	/sbin	存放管理员的系统管理程序	中
	/lib	存放系统动态链接共享库	低
	/lost+found	系统运行异常时产生的错误，会将遗失的片断放在这里	低
	/mnt	可临时将别的外部设备挂接在此目录下	低
	/proc	存在系统内存中的信息	低
	/usr	用户的应用程序和文件都存放在这个目录下	低
	/tmp	存放临时文件的目录	低
VI 编辑器	vi filename	生成新文件或者编辑查看文件	高
	i 或者 a	从命令模式进入编辑模式, i为插入文本，a为追加文本	高
	Esc	从编辑模式进入命令模式	高
	:w	保存文本	高
	:wq	保存并退出	高
	:wq!	保存并强制退出	高
	:q	退出	高
	:q!	强制退出	高
	o	添加一行	高
	O	在光标所在行的上方添加一行	高
	dd	删除一行文字	高
	D	删除从当前光标到行尾的内容	
	x	删除一个字符	高
	s	删除一个字符并切换到编辑模式	高
	S	删除一行并切换到编辑模式	高
	:n	光标移至文本第n行	高
	$	光标移到文本的行尾	高
	A	光标移到文本的行尾并切换到编辑模式	高
	^	光标移到文本的行首	高
	G	光标移到文本的末尾	高
	gg	光标移到文本的首行	高
	ZZ	存盘退出	中
	/字符串	查找某个字符串	高
	n	继续查找	高
	:u	撤消(同标准编辑器中的Ctrl+Z)	中
	:redo	重做(同标准编辑器中的Ctrl+Y)	中

Linux 启动级别	文件/etc/inittab中设置默认启动级别		中
	0	代表halt，关机操作，这个0不能设置，否则机器将不能启动	低
	1	代表单用户模式，采用这个设置，系统只能允许一个用户登	中
	2	代表多用户模式，但不支持网络工作	中
	3	代表命令行界面，即文本界面，是企业中服务器通用的启动模式	高
	4	系统预留，该级别目前还没有使用	低
	5	代表图形界面，也是linux系统启动时默认的启动模式	高
	6	代表重启模式，这个6也不能设置，否则系统反复重启	中
系统关机重启	reboot	重启	高
	shutdown –r now	现在立刻重启	高
	shutdown –r 11:30	等到11：30进行重启	高
	shutdown –r +1	等1分钟后重启	高
	halt	关机	高
	shutdown –h now	现在立刻关闭系统	高
	shutdown –h 11:30	等到11：30关闭系统	高

文件操作

功能项	命令实例	作用	重要性
文件创建	vi /opt/learn/hello.txt	在目录/opt/learn下创建文件hello.txt并进入vi编辑界面	高
	touch /opt/learn/test	在目录/opt/learn下创建空白文件test	高
	cat > /opt/learn/catfile	创建文件catfile并在屏幕上输入内容，最后按Ctrl+D退出	高
文件查看	vi /etc/passwd	在vi编辑器中输出文本内容	高
	cat /etc/passwd	在屏幕上输出文本内容	高
	more /etc/passwd	分屏输出文本内容	高
	less /etc/passwd	分屏输出文本内容并按需加载文件(适用于大文件的查看)	高
	head -n 10 /etc/passwd	只输出文件的头10行	高
	tail -n 20 /etc/passwd	只输出文件末尾的20行	高
	strings /bin/ls	查看二进制文件中的可打印字符	高
文件操作	cp hello.txt /opt/test	把文件hello.txt复制到文件夹/opt/test下	高
	cp hello.txt /opt/test/hello.cp	把文件hello.txt复制到文件夹/opt/test下并重命名成hello.cp	高
	mv hello.txt /opt/test	将文件hello.txt剪切到文件夹/opt/test下	高
	mv hello.txt /opt/test/hello.mv	将文件hello.txt剪切到文件夹/opt/test下并重命名成hello.mv	高
	mv hello.txt hello2.txt	重命名	高
	rm /opt/test/hello.cp	删除文件	高
	rm -f /opt/test/hello.mv	强制删除文件，不会有提示信息	高
	du -sk hello.txt	查看文件hello.txt的大小 (以K为单位)	高
链接	ln -s hello.txt shello	为hello.txt文件创建一个名为shello的软链接(类似于快捷方式)	高
	ln -d hello.txt dhello	为hello.txt文件创建一个名为dhello的硬链接 硬链接表示所有文件中更改任意一个，其他文件的所有属性会跟着变化，如大小，更新时间，权限等	高

文件夹操作

功能项	命令或格式	作用	重要性
ls / tree	ls [option] [file/directory]	显示指定目录下的所有文件或文件夹 (同Windows->dir命令)	高
	ls	显示当前目录的内容	高
	ls -l	显示当前目录详细内容	高
	ls -a	显示当前目录下的所有文件，包括隐藏文件	高
	ls *.txt	显示目前下所有以.txt为后缀名的文件	高

	ls /opt/training	显示目录/opt/training下的内容	高
	ls -R /opt/	列出所有/opt目录及其子目录的内容	高
	tree /opt	用树状结构显示目录及文件	中
pwd	pwd	显示当前所在目录	高
cd	cd directory	切换到指定目录	高
	cd	切换到当前用户所有的主目录	高
	cd ..	回退到当前目录的上一级目录	高
	cd /opt/learn	用绝对路径切换到/opt/training目录下	高
	cd ../../	使用相对路径切换到当前目录的上一级的上一级目录下	高
	cd .	切换到当前用户，相当于什么也没做	高
mkdir	mkdir [option] [director1] [directory2] …	创建目录	高
	mkdir /opt/learn/other	在目录/opt/learn/下创建目录other	高
	mkdir dir2 dir3 dir4	同时创建dir2 dir3 dir4三个目录	中
	mkdir -p /dir1/dir2/dir3/dir4	同时创建一个4层目录	高
rmdir	rmdir dir1	删除一个空目录	低
其他操作	cp -r /opt/learn /opt/learn2	拷贝文件夹	高
	mv /opt/learn2 /opt/learn3	重命名文件夹	高
	rm -rf /opt/learn3	强制删除文件夹	高

权限操作

功能项	命令实例	作用	重要性
用户组操作	groupadd testing	添加一个新的用户组testing	中
	cat /etc/group	查看组是否被新增成功	中
	groupmod -n test testing	将testing重命名成test	中
	groupdel test	删除组test	中
	groups root	查看用户root所在的所有组	中
useradd	useradd qiang	新增一个用户qiang (默认时将新增一个对应的名为qiang的组)	中
	useradd -g test denny	新增一个用户denny并将其加入test组	中
	useradd -g test -G 501 mary	标准用法，501这个数字与test组的gip对应即可	中
		将	
usermod	usermod -g dev qiang	将用户qiang换到dev组	中
	usermod -G 502 qiang	将用户qiang附加到gid为502的这个组	中
	usermod -d /home/temp/mary	将mary的主目录从/home/mary改为/home/temp	中
userdel	userdel qiang	删除用户qiang	中
	userdel -f qiang	强制删除用户qiang (即使该用户已经登录)	中
	userdel -r qiang	删除用户qiang并删除其主目录	中
chmod	chmod [权限] [文件或目录]	更改文件或目录的权限	高
	ls -l hello.txt	查看文件的详细属性，对其进行解释	高
	左边10位中的第一位代表文件类型	d --- 代表目录 - --- 代表普通文件 l --- 代表链接文件	高

	左边10位中的后9位代表权限	前3位代表文件所有者的权限 (用u表示) 中间3位代表文件所在组的权限 (用g表示) 后3位代表其他组的权限 (用o表示)	高
	权限rwx的意义	权限 r 或数字 4 -- 表示可读 权限 w 或数字 2 -- 表示可写 权限 x 或数字 1 -- 表示可执行	高
	chmod u+x hello.txt	为hello.txt文件所有者添加可执行权限	高
	chmod u-w hello.txt	为hello.txt文件所有者去除可执行权限	高
	chmod g-r hello.txt	为hello.txt文件所在组去除可读权限	高
	chmod o+w hello.txt	为hello.txt文件的所在组的其它组添加可写权限	高
	chmod a+w hello.txt	为所有三种角色添加可写权限	高
	chmod a+wx hello.txt	为所有三种角色添加可写权限	高
	chmod a-rwx hello.txt	去除hello.txt的所有权限(此时仅root可编辑)	高
	chmod 777 hello.txt	将hello.txt的权限设为rwxrwxrwx	高
	chmod 643 hello.txt	将hello.txt的权限设为rw-r---wx	高
	chmod 777 /opt/test	将目录/opt/test的权限更改为777	高
	chmod -R 755 /opt/test	将目录/opt/test及其下所有文件和子目录的权限更改为755	高
chown	chown mary hello.txt	将hello.txt的文件所有者改为mary	中
	chown mary /opt/test	将目录/opt/test的所有者改为mary	中
	chown -R mary /opt/test	将目录/opt/test及其所有子目录和文件的所有者改为mary	中
chgrp	chgrp test hello.txt	将hello.txt所在的组改为test	中
	chgrp mary /opt/test	将目录/opt/test所在的组改为mary	中
	chgrp -R mary /opt/test	将目录/opt/test及其所有子目录和文件所在的组改为mary	中
passwd	passwd mary	修改mary的密码	中

文件查找

功能项	命令实例	作用	重要性
find	find 起始目录 查找类型 查找条件	查找起始目录及所有子目录下的文件及文件夹	中
	find . -name "hello.txt"	查找当前目录下文件名为hello.txt的文件或文件夹	中
	find . -name "*hello*"	查找当前目录下文件名包含hello的文件或文件夹	高
	find /home -name "*bash*"	查找目录/home下文件名包含bash的文件或文件夹	中
	find . -name "*"	查找当前目录下的所有文件或文件夹 (作用同ls -R)	中
	find . -name "[h]*"	查找当前目录下以h开头的文件或文件夹	中
	find . -name "[h|f]*"	查找当前目录下所有以h或f开头的文件或文件夹	中
	find . -name "[a-z]*"	查找当前目录下所有以小写字母开头的文件或文件夹	中
	find . -name "[A-Z]*"	查找当前目录下所有以大写字母开头的文件或文件夹	中
	find . -name "[a-Z]*"	查找当前目录下所有以字母开头的文件或文件夹	中
	find . -name "[h-w]*"	查找当前目录下所有以字母h-w开头的文件或文件夹	中
	find . -name "[0-9]*"	查找当前目录下所有以数字开头的文件或文件夹	中
	find . -name "[5-8]*"	查找当前目录下所有以数字5-8开头的文件或文件夹	中
	find . -name "h?llo*"	查找当前目录下所有以h后面带一个字符再加llo开头的文件或文件夹	中
	find . -name "[^a-h]*"	查找当前目录下所有不以a-h开头的文件或文件夹	中
	find . -name '**'	查找当前目录下所有包含特殊字符\的文件(注意使用单引号)	中
			中
	find . -perm 777	查找当前目录下权限为777的文件或文件夹	中
	find . -path "./test" -prune -o -name "*hello*"	查找当前目录下除test目录的其他所有目录中包含hello的文件或文件夹	中
	find . -user mary	查找当前目录下文件所有者为mary的文件或文件夹	中
	find . -group dev	查找当前目录下文件或文件夹所在组为dev的内容	中

	find . -mtime -3	查找当前目录下在3天内更新过的文件或文件夹	中
	find . -mtime +3	查找当前目录下在3天前更新过的文件或文件夹	中
	find . -newer helllo.txt	查找当前目录下比hello.txt新的文件或文件夹	中
	find . ! -newer helllo.txt	查找当前目录下比hello.txt旧的文件或文件夹	中
	find . -type d	查找当前目录下所有的文件夹 (普通文件的类型为 f)	中
	find . -type l	查找当前目录下所有的软链接文件	中
	find . -size 602c	查找当前目录下文件大小为602字节的文件，可用单位为：c - byte, k - Kilobytes, M - Megabytes, G- Gigabytes	中
	find . -size +602c	查找当前目录下文件大小大于602字节的文件 (用-表明小于)	中
	find . -name "hello*" -exec ls -l {} \;	查找当前目录下所有以hello开头的文件并将其细节显示出来，如果查找出了目录，那么此时要注意目录会被ls -l列出来	中
	find . -name "hello*" -exec rm {} \;	查找当前目录下所有以hello开头的文件并将其删除	中
	find . -name "hello*" | xargs ls -l	查找当前目录下所有以hello开头的文件并将其细节显示出来	高
grep	grep [选项] 匹配模式 目标文件	基于行对目标文件的内容进行查找	中
	grep "root" /etc/passwd	查找到/etc/passwd文件中包含root的行	中
	grep -n "root" /etc/passwd	查找到/etc/passwd文件中包含root的行并输出行号	中
	grep "^ma" /etc/passwd	查找以ma为行首的行	中
	grep "bash$" /etc/passwd	查找以bash为行尾的行	中
	grep "^[r|d]" /etc/passwd	查找以r或d为行首的行	中
	正则表达式参考	http://zh.wikipedia.org/wiki/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F	低

归档压缩及其它

功能项	命令实例	作用	重要性
tar / gzip	tar -cvf hello.tar hello.txt	将hello.txt归档并命名成hello.tar	中
	tar -cvf test.tar /opt/test	将目录/opt/test归档并命名成test.tar	中
	tar -tf test.tar	将归档文件test.tar中的文件显示出来	中
	tar -xvf test.tar	提取归档文件中的内容	中
	gzip hello.tar	将归档文件hello.tar压缩成hello.tar.gz	中
	gzip -d hello.tar.gz	解压缩文件成hello.tar	中
	tar -zcvf hello.tar.gz hello.txt	将hello.txt归档并压缩成hello.tar.gz	中
	tar -zxvf hello.tar.gz	解压缩	中
zip / unzip	zip hello.zip hello.txt	将hello.txt压缩并命名成hello.zip	低
	zip -r test.zip /opt/test	将/opt/test目录压缩	低
	unzip -v hello.zip	查看压缩文件hello.zip中的文件信息	低
	unzip hello.zip	解压缩hello.zip	低
mount	mount -t cifs -o username=admin,password=admin //192.168.0.10/share /mnt/share	将Windows机器192.168.0.10的共享目录share挂载到/mnt/share目录下	高
	mount /dev/cdrom /mnt	将光驱挂载到/mnt目录	低
	mount /dev/sda2 /mnt	将USB设备挂载到/mnt目录	低
	mount	查看已经挂载的设备	中
	mount -a	挂载所有/etc/fstab文件中定义的设备	低
	umount -a	弹出所有/etc/fstab文件中定义的设备	低
ifconfig	ifconfig	查看当前网卡的信息 (同Windows下ipconfig)	中
	ifconfig -a	查看所有 (同Windows下ipconfig /all)	中
	ifconfig eth0	查看设备eth0的信息	中

	ifconfig eth0 172.168.1.100	设置eth0网卡的IP地址	中
	ifconfig eth0 down	禁用eth0网卡	中
	ifconfig eth0 up	启用eth0网卡	中
	ping 172.168.1.200	检查是否能连通172.168.1.200	高
	hostname	输出机器名	中
	netconfig	打开字符界面，可修改IP地址及网关等	中
变量	export NAME=51Testing	定义一个环境变量	高
	EMAIL=linux@abc.com	定义一个变量	高
	echo $NAME	输出一个变量的值	高
su	su - root	从当前用户切换到root用户(然后使用root的环境变量)	高
	su root	从当前用户切换到root用户(然后使用当前用户的环境变量)	高
top	top	查看当前系统的资源使用率 (默认刷新间隔为5秒)	低
	top -d 1	设置刷新间隔为1秒	低
	top -n 3	输出3次后便退出	低
ps	ps	查看当前终端正在运行的进程	中
	ps -ef	查看系统正在运行的所有进程	中
	ps -ef | grep bash	查看系统正在运行的进程名包含bash的进程	中
kill	kill pid	根据进程号终止进程的运行	中
	kill -9 pid	强制终止	中
chkconfig	chkconfig --list	列出所有服务 (同Windows下运行services.msc后所看到的内容)	低
	chkconfig --list | grep sshd	查找是否存在服务sshd及其启动级别	低
	chkconfig --add httpd	将httpd文件注册成服务	低
	chkconfig --del httpd	将httpd服务从系统中删除（同Windows下sc delete servicename)	低
	chkconfig --level 35 httpd on	将httpd服务设为3和5级别下自动启动	低
	service httpd start|stop|restart|status	启动|停止|重启|查看状态	中
rpm	rpm -qa | grep gcc	查找和gcc相关的rpm安装包	中
	rpm -ivh xxxx.rpm	安装一个rpm包	中
	rpm -e xxxx	卸载一个rpm包	中
杂项	clear	清除屏幕内容	中
	man 命令	查看某个命令的帮助内容	中
	whereis 命令	查看某个命令所在的位置	中
	id	查看当前用户的基本信息	低
	date	输出当前系统日期和时间	中
	date -s 2009-10-10 / 12:12:12	修改日期或者时间	中
	date 月日时分年.秒	一次性修改日期时间，如date 112315452009.59修改后为2009年11月23日15点45分59秒	中
	date "+%Y-%m-%d %H:%M:%S"	格式化输出	中
	cal	输出当前月的日历	低
	cal 9 1752	输出1752年9月的日历，看看发生了什么，仔细看	低
	free	查看内存使用情况	中
	history	列出执行过的命令的历史记录	中

	history -c	清除历史记录	中
	whoami	我是谁	中
	who am i	当前我在哪个终端	中
	yes	在屏幕上无限输出y	低
	export LANG=C export LC_ALL=en	如果默认语言设为中文，在终端可能存在乱码，运行此命令可解决	中
	rm -rf /	你敢试试吗	低

Shell基本用法

功能项	命令实例	作用	重要性
Shell	Bourne Shell	是贝尔实验室开发的，Unix普遍使用的Shell，在编程方面比较优秀，但在用户交互方面没有其他Shell优秀。	低
	Korn Shell	是对Bourne Shell 的发展，在大部分内容上与Bourne Shell兼容，集成了C Shell和Bourne shell优点。	低
	C Shell	是SUN公司Shell的BSD版本，语法与c语言相似，比bourne shell 更适合编程	低
	BASH	是GNU的Bourne Again Shell，是GNU操作系统上默认的Shell，在Bourne Shell基础上增强了很多特性，如命令补全，命令历史表	低
Shell操作	cat /etc/shells	列出系统中所有的Shell	中
	ksh/csh/zsh/bash	切换到其它Shell	中
	chsh qiang	使用命令chsh更改用户qiang的默认Shell	中
	cat /etc/passwd	查看用户使用的默认的Shell	中
	echo $SHELL	查看当前环境变量$SHELL的值	中
Bash的功能	TAB键	命令补全功能 (很实用)	中
	history命令或上下箭头	命令的历史记录	中
	alias gohome="shutdown -h now"	命令别名功能	中
	crontab	作业控制功能	中
	shell脚本编程	非常灵活的脚本编程能力	中
	ls;cat /etc/passwd;mount	三个命令放在一起通过;分隔	中
echo	echo "HelloWorld"	在屏幕上输出HelloWorld	高
	echo "Hello\nWorld"	在屏幕上输出Hello\nWorld	高
	echo -e "Hello\nWorld"	在屏幕上输出 Hello World	高
	\a	蜂鸣	低
	\b	退格	低
	\c	不带换行符打印一行	高
	\f	换页	低
	\n	换行	高
	\r	回车	低
	\t	制表符	高
	\v	纵向制表符	低
	\\	反斜杠	高
	\0nnn	ASCII码是nnn(八进制)的字符	低
	练习：	使用echo命令输出如下格式的内容(注意对齐格式)： id name msg 01 mike “hello” 02 john “hi” echo -e "id\tname\tmsg\n01\tmike\t\"hello\"\n02\tjohn\t\"hi\""	高
read	# read NAME # Bill Gates	从终端将值读入并赋值给变量NAME	高

	# echo $NAME	将变量NAME的值输出	高
	# read NAME SURNAME # Bill Gates	此时会将Bill赋值给NAME，而将Gates赋值给SURNAME	高
中文或乱码	vi /etc/profile	在里面新增: export LANG=C export LC_ALL=en_US.UTF-8 (解决终端乱码的问题) 或者新增: export LC_ALL=zh_CN.GB2312 export LC_CTYPE=zh_CN.GB2312 (解决终端不能显示中文的问题)	中
环境变量	set	显示当前shell的变量，包括当前用户的变量	中
	env	显示当前用户的变量	中
	export	显示当前导出成用户变量的shell变量	中
	cat /etc/profile	全局的环境变量，对任何用户生效	中
	cat ~/.bash_profile	用户主目录下的环境变量，仅对当前用户生效 (本地变量定义在此)	中
	export NAME=Denny	定义一个NAME环境变量并赋值为Denny	中
	echo $NAME	输出一个环境变量的值	中
	unset NAME	删除环境变量	中
常用的环境变量	echo $USER	当前登录的用户名	中
	echo $UID	当前登录的用户ID号	中
	echo $SHELL	当前所使用的SHELL	中
	echo $HOME	当前用户的主目录	中
	echo $PWD	当前命令行所在的目录	高
	echo $PATH	当前可执行程序的路径 (设定了PATH，执行命令就不用输入命令的绝对路径)	中
	echo $PS1	命令的提示字符串 (可以试一下export PS1="Welcome Linux# "	中
	echo $PS2	命令一行未写完时换行提示符	中
本地变量	cat ~/.bash_profile	本地变量在此定义，将只对本用户生效	中
	# NAME=Denny # echo $NAME	在命令行定义变量NAME并取得其值	中
	# SOURCE="/etc/passwd" # DEST="/opt/learn" # cp $SOURCE $DEST	同上	中
	# NAME=Denny # readonly NAME	设置只读变量，此时变量的值将不能被修改	中
管道及重定向	set | grep USER	从set的输出中查找包含USER的行	高
	ls | wc -l	根据ls输出的行数来统计该文件夹下的文件数量	高
	head /etc/passwd > passwd.txt	将/etc/passwd文件的输出重定向到文件passwd.txt中	高
	tail /etc/passwd >> passwd.txt	在文件passwd.txt后面累加	高
	cat < /etc/passwd	将/etc/passwd文件作为cat的输入	高
	cat < /etc/passwd > passwd2.txt	将/etc/passwd文件作为cat的输入，并将输出结果重定向到passwd2.txt中	高
	ifconfig | tee ifconfig.tee	将ifconfig的内容输出到屏幕上同时输出到文件ifconfig.tee中	高
	ifconfig | tee -a ifconfig.tee	以追加的方式输出到文件ifconfig.tee中	高
	ifconfig 1> ifconfig.out	标准输出重定向到文件ifconfig.out中	高
	ifconfige 1> ifconfig.error	将标准输出重定向到文件ifconfig.error中，此处由于不存在命令ifconfige，所以ifconfig.error中将没有内容	高
	ifconfig 2> ifconfig.error	标准错误重定向到文件ifconfig.error中，由于此处没有错误，所以ifconfig.erro文件中将没有内容	高
	ifconfige 2> ifconfig.error	标准错误重定向到文件ifconfig.error中	高
	ls /opt /optt 2> cat.error	将标准输出输出到屏幕而将标准错误重定向到cat.error中	高
	ls /opt /optt 1> cat.error 2>&1	标准输出和标准错误一起重定向到一个文件	高

	ls /opt /optt 2> /dev/null	标准错误信息会输送到系统垃圾箱，而不会输送到屏幕	高
	命令1 && 命令2	命令1成功执行后才执行命令2	高
	命令1 || 命令2	命令1执行失败后才执行命令2	高
	tty	查看当前终端设备编号	高
	练习：	打开SecureCRT, 在VMWare中登录进虚拟机命令行界面，用tty查看两个终端的编号，然后用 > 将输出重定向到另外的终端	高
	练习：	写一个简单的脚本文件catfile.sh，要求实现的功能： 1) 用户随意输入3个文件名，这3个文件的内容能够被cat命令连接起来显示，并且所有行都被标号； 2) 用户输入的文件名可能真实存在，也可能不存在，需要将标准输出和标准错误分别重定向到文件catfile.log和catfile.err： echo "Please input 3 files' name:\n" read file1 file2 file3 cat -n $file1 $file2 $file3 1> catfile.log 2> catfile.err	高

Shell 脚本基础

功能项	命令实例	作用	重要性
Hello World	#!/bin/bash # This is the first Shell program … echo "Hello World …"	一个Shell包含3个基本部分： 第一行声明该脚本将用/bin/bash来解释 第二行是备注，简单介绍脚本的作用 第三行是程序体，用来输出Hello World … 可使用/bin/csh来解析export命令，会出错，由此来验证第一行的作用	高
	bash helloworld.sh	使用bash命令执行helloworld.sh	低
	sh helloworld.sh	使用sh命令执行helloworld.sh	高
	ksh helloworld.sh	使用ksh命令执行helloworld.sh	低
	. helloworld.sh	使用.命令执行helloworld.sh	低
	source helloworld.sh	使用source命令执行helloworld.sh	低
	./helloworld.sh	自执行 (确保helloworld.sh有执行权限)	高
参数	echo "Hello, $1 $2 $3"	程序体输出Hello后面带三个参数的值	高
	sh hello.sh Denny Bill Mary	运行时会输出Hello, Denny, Bill, Mary	高
	1	代表第一个参数	高
	2	代表第二个参数	高
	…..	以此类推，但不能超过9个参数 (试试看$10会输出什么)	高
	0	特殊的，$0表示该Shell脚本的名称	高
特殊变量	$#	传递到脚本的参数个数	高
	$0	脚本的名称	高
	$*	以一个单字符串的形式显示所有向脚本传递的参数，与位置变量不同，此项参数可超过9个	高
	$$	脚本运行的当前进程id号	高
	$!	后台运行的最后一个进程的进程id号	高
	$@	与$*相同，但是使用时加引号，并在引号中返回每个参数	高
	$?	显示最后命令的退出状态，0表示正确，其他任何值表示错误 特别的：在脚本中可自定义0~255的退出状态码	高
	$_	代表上一个命令的最后一个参数	高

	理解$*和$@的区别：	function testargs
		{
		echo "There is $# args"
		echo $10
		}
		testargs "$*"
		testargs "$@"
			高

引号的魅力	echo "$1"	输出第一个参数的值	高
	echo '$1'	输出$1	高
	echo 'date'	输出date	高

	echo `date`	输出当前的时间 (注意这里使用的是~下面的那个反引号而不是单引号)	高
expr	expr 10 + 10	expr为一个手工计算器，此处会输出20 (注意空格)	低
	expr 10+10	此处会输出10+10	低
	expr 10.1 + 1	expr不能处理小数	低
	expr "hello" = "hello"	成功返回1，失败返回0	低
	练习：	使用echo命令输出一句话:300/5*6=360，其中360由运算而来： echo "300/5*6=`expr 300 / 5 \* 6`"	低
test 文件	test -e /etc/passwd	测试文件/etc/passwd是否存在，存在则 $? 返回0，否则返回1	高
	[-e /etc/passwd]	与test -e /etc/passwd作用一样，注意空格	高
	-d file	如果文件为一个目录，则为真	高
	-e file	如果文件存在，则为真	高
	-f file	如果文件为一个普通文件，则为真	高
	-r file	如果文件可读，则为真	高
	-w file	如果文件可写，则为真	高
	-x file	如果文件可执行，则为真	高
	-s file	如果文件的长度不为零，则为真	高
	-L file	如果文件为符号文件，则为真	高
逻辑	[-e /etc/passwd -a -r /etc/passwd]	逻辑与，操作符两边均为真，结果为真，否则为假	高
	[-e /etc/passwd -o -r /etc/passwd]	逻辑或，操作符两边一边为真，结果为真，否则为假	高
	[! -e /etc/passwd]	逻辑否，条件为假，结果为真	高
test 字符串	[$USER = "root"]	字符串比较	高
	["$USER" = "root"]	建议使用此方式	高
	=	等于	高
	!=	不等于	高
	-z	为空字符串	高
	-n	非空字符串	高
test 数值	[$$ -eq 18646]	对数值的测试	高
	-eq	数值相等	高
	-ne	数值不相等	高
	-le	第一个数小于等于第二个数	高
	-ge	第一个数大于等于第二个数	高
	-gt	第一个数大于第二个数	高
	-lt	第一个数小于第二个数	高

程序控制结构	if 条件
	then
	  命令
	fi
		如果脚本的参数个数少于3个，则提示用户需要3个参数：
		if [$# -lt 3]
		then
		  echo "Sorry, it needs 3 args"
		fi
			高

	if 条件; then 命令 fi	同上	高

	if 条件; then
	  命令1
	else
	  命令2
	fi
		提示用户输入他的账号，并显示欢迎信息，如为空则提示:
		echo "Please input your name: "
		read NAME
		if ["$NAME" = ""]; then
		  echo "Your name is null"
		else
		  echo "Welcome, $NAME"
		fi
			高

	if 条件1; then
	  命令1
	elif 条件2; then
	  命令2
	else
	  命令3
	fi
		如果用户输入的帐号为空，则提示，否则如果帐号在/etc/passwd中能找到，则欢迎，否则拒绝服务：
		echo "Please input your name: "
		read NAME
		if ["$NAME" = ""]; then
		  echo "Your name is null"
		elif grep $NAME /etc/passwd > /dev/null ; then
		  echo "Welcome, $NAME"
		else
		  echo "Sorry, $NAME"
		fi
		用grep查找进也可按查找到的具体行数进行判断：grep -c ^$NAME /etc/passwd
			高

	for 变量名 in 列表
	do
	  命令
	done
		以脚本的参数作为循环依据，将参数输出
		#for loop in 1 2 3 4 5 6
		for loop in "$@"
		do
		  echo $loop
		done
			高

	练习：	统计出某个目录下有多少个文件： counter=0 for files in `ls` do counter=`expr $counter  +  1` done echo “There are $counter files in `pwd`" -- 注意变量counter赋值时等号前后不能有空格，否则会认为counter是一个命令	高
	练习：	求1～100之间的所有整数和	高

	while 条件
	do
	  命令
	done
		根据参数1决定循环次数，并将每次结果输出：
		i=0
		while [$i -lt $1]
		do
		  echo "Hello, $i"
		  i=`expr $i + 1`
		done
			高

	练习：	从某文件中一行一行读取内容，并将其加上行号输出：
		line=1
		while read FILE
		do
		  echo "$line: $FILE"
		  line=`expr $line + 1`
		done < /etc/group
			高

	练习：	从某文件中一行一行读取内容，并将其中用空格或者TAB分隔的内容按列输出：
		Data文件为：
		NAME EMAIL PHONE
		Denny dq@qq.com 12345678
		Qiang qq@dd.net 45678900
		Angel an@12.cn 087y8438
		
		代码为：
		line=0
		while read NAME EMAIL PHONE
		do
		  line=`expr $line + 1`
		  if [$line -eq 1]
		  then
		  continue
		  fi
		
		  echo "----$line----"
		  echo "$NAME"
		  echo "$EMAIL"
		  echo "$PHONE"
		done < $1
			高

	case 变量 in 模式1) 命令 ;; 模式2) 命令;; esac	根据输入的命令在程序中进行相应的调用： case $1 in "date") echo "executing date: `date`" ;; "cal") echo "executing cal: `cal`" ;; "id") echo "executing id: `id`" ;; *) echo "usage: $0 date|cal|id" ;; esac	高
	break / continue	以下代码如果用break，则只会输出1,2,3,4，如果用continue，则只有5不会输出 i=0 while [$i -lt 10] do i=`expr $i + 1` if [$i -eq 5]; then #continue break fi echo $i done	高

函数	函数名()
	{
	命令1
	…
	}
	或
	function 函数名()
	{ ….}
		检查参数是否是一个目录(注意此处参数的调用)
		is_directory() {
		if [-d $1]; then
		echo "Great, $1 is a directory ..."
		else
		echo "Sorry, $1 is not a directory ..."
		fi
		}
		
		echo "check if the input is a directory ..."
		is_directory $1
		echo "check end ..."
			中

	带返回值的函数	function myfunc() {
		i=$1
		j=$2
		echo `expr $i \* $j`
		}
		
		result=`myfunc 59 100`
		
		if [$result -lt 6000]; then
		echo "Too Low"
		else
		echo "Great"
		fi
			中

综合练习	写一个脚本，实现创建目录的功能，目录的名称由用户给出，需要对如下情况进行判断处理： 1) 用户没有给出参数，提示脚本用法 2) 用户给出的目录名称是否存在，如果存在，则提示存在并退出脚本，否则询问用户是否需要重新创建: 如果Y，则创建目录，否则，退出脚本 3) 目录创建成功或者失败，都给出说明信息	DIRECTORY=$1 if [$# -lt 1] then echo "Usage: `basename $0` dirname" exit 1 fi if [-d $DIRECTORY] then echo "Directory is existent" else echo "The $DIRECTORY doesn't exist" echo -n "Create it now? [y...n]: " read ANS if ["$ANS" = "y"] || ["$ANS" = "Y"] then echo "Creating now ..." mkdir $DIRECTORY > /dev/null 2>&1 if [$? -eq 0] then echo "Create directory $DIRECTORY successfully ..." else echo "Failed to create direcotory $DIRECTORY" exit 1 fi fi fi	高

Shell 扩展知识

功能项	命令实例	作用	重要性
cron	crontab	每个用户都可以有一个crontab文件来保存调度信息，通过该命令运行任意一个shell脚本或者命令	高
	cron.deny/cron.allow	系统管理员可以通过cron.deny和cron.allow这两个文件来禁止或允许用户拥有自己的crontab文件	高
	crontab的域	第1列 分钟1～59 第2列 小时1～23(0表示子夜) 第3列 日1～31 第4列 月1～12 第5列 星期0～6(0表示星期天) 第6列 要运行的命令	高
	常用规则	*: 匹配任何值 */1: 匹配每1个单位(如每一分钟，每一小时) x: 匹配x x-y: 匹配从x-y的值 x,y,z: 只匹配x, y, z三个值	高
	crontab [-u user] -e -l -r	-u 用户名 -e 编辑crontab文件 -l 列出crontab文件中的内容 -r 删除crontab文件	高
	service crond start/stop	启动停止crond进程，如crond进程停止，则不会有任务被自动执行	高
at	at 时间	指定某一特定时间去做某件事情	中
	at HH:MM	at 16:00	中
	at HH:MM YYYY-MM-DD	at 16:00 2009-10-11	中
	at now + 5 minutes	从现在开始的5分钟后	中
	如何退出at编辑模式	Ctrl + D	中
nohup &	command &	后台运行，随着终端的退出而退出	中
	nuhup command &	后台运行，退出终端不影响该进程	中
sleep	sleep n	让Shell脚本暂停n秒	高
	usleep n	让Shell脚本暂停n钠秒	高
time	time command	计算某一个命令或者脚本运行时花的时间(精确到毫秒)： 如：time ls (ls这个命令所花的时间) time sh myshell.sh (运行myshell.sh这个脚本所花的时间)	中
getopts	myshell.sh -a -n 3	使脚本运行起来更像一个命令，参数没有顺序限制，此处-a表明是一个选项，无值，而-n表示该选项所对应的值为3： while getopts ahn:d: OPTION do case $OPTION in a) echo "a is here" ;; n) echo "n is here and the value is $OPTARG" ;; esac done	中
awk	awk	对文本进行逐行处理的编程语言,它来源于3个创作者的名字：Aho、(Peter)Weinberg和(Brain)Kernighan.与sed和grep很相似，awk是一种样式扫描与处理工具，但其功能却大大强于sed和grep	中
	awk -F : '$1~/root/ {print $1}' /etc/passwd	查找/etc/passwd下面第一个域为root的行并将其第一个域打印出来 (-F : 表示以冒号分隔域)	中
	awk -F : 'BEGIN {sum=0} $0!~/root/ {sum+=1} END {print sum}' /etc/passwd	查找/etc/passwd中不包含root的行并统计一个有多少行	中

	awk -F : '{if ($1=="root") print $1; else print $2}' /etc/passwd	文件/etc/passwd中如果第一个域包含root则打印它，否则打印第三个域的值	中
	awk -F : '$0!~/(bin|root)/' /etc/passwd	打印文件中不包含bin或者root的行 (特别的$0表示整行)	中
	top -d 1 | awk '$0~/yes/ {print $9}'	查找进程中包含yes的进程并打印出其CPU使用率	中
sed		http://www.ringkee.com/note/opensource/sed.htm	低
数组	Shell脚本中数组的使用	在Bash中，可以采用显示的delare -a variable语句引入整个数组。 如：declare -a weekday weekday[0]=Sunday weekday[1]=Monday .... 输出数组中的所有元素：${weekday[@]}, 输出某一元素${weekday[1]} 使用#字符求取整个数组有多少个元素：echo ${#weekday[@]}或echo ${#weekday[*]} 还可以使用#号求取数组元素的长度:echo ${#weekday[0]}，将输出6。 unset命令可以删除数组元素，甚至整个数组。如： unset weekday[1]，将去除weekday[1]的值。 unset weekday，将去除数组weekday所有元素的值。 通过赋值语句可以实现数组的复制，如： array2="${array1[@]}" 或 array2=("$array1[@]}")	低