

Shell Programming

2021 Fall

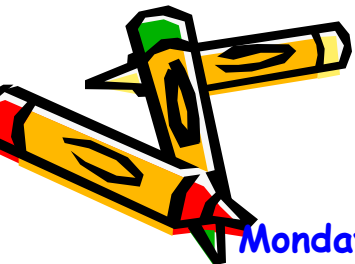
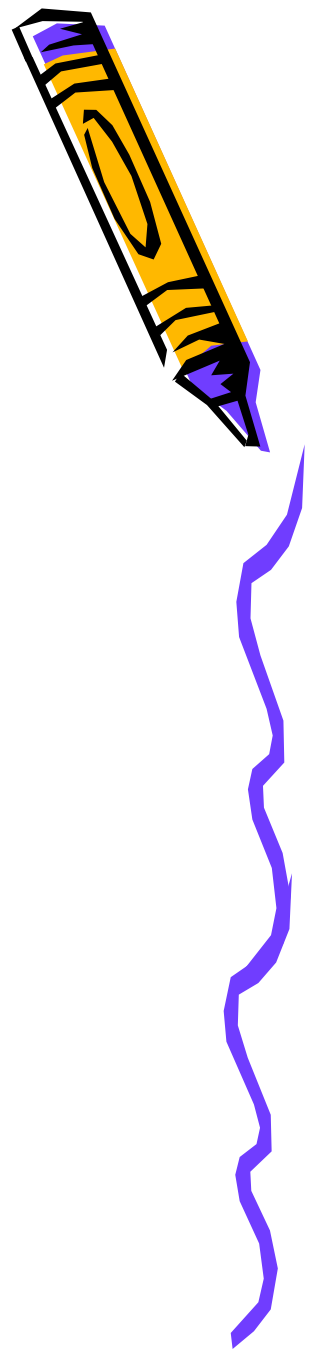
Zhaobin Liu

zhbliu@gmail.com



Outline

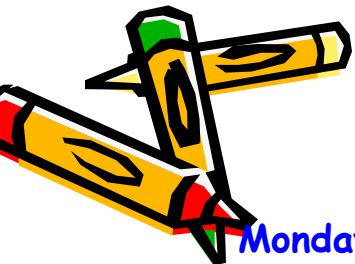
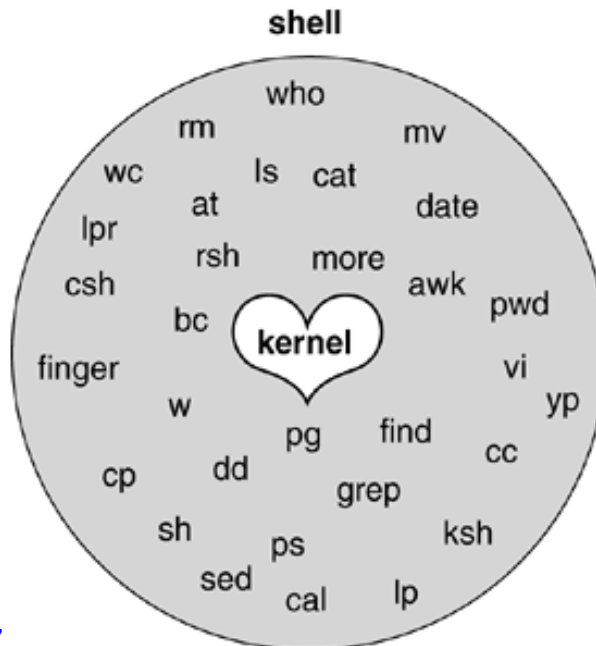
- Definition
- Variables
- Program structure
- Scripts example
-



What Is SHELL?

- The shell is a special program used as an **interface** between the user and the heart of the UNIX/Linux operating system, a program called the kernel.

- The kernel, the shell, and you

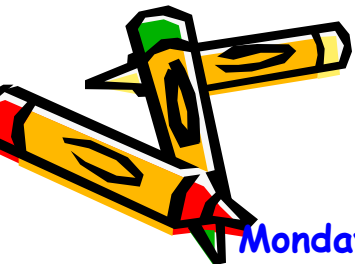


The Linux Shells



- `$cat /etc/shells`
- `$ echo $SHELL`
- Responsibilities of the Shell

The default Bash prompt is the dollar sign (\$).

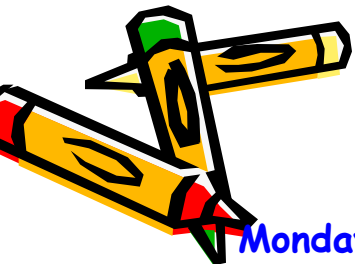


The Linux Shells



- `$cat /etc/shells`
- `$ echo $SHELL //examine default shell`
- Responsibilities of the Shell

1. Reading input and parsing the command line
2. Evaluating special characters, such as wildcards and the history character
3. Setting up pipes, redirection, and background processing
4. Handling signals
5. Setting up programs for execution



The **B**ourne **A**gain **S**hell

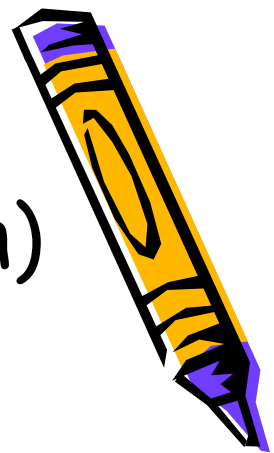


- Abbreviated **bash**
- Default in most Linux distributions
- Most widely used on all UNIX platforms
- Derives features from ksh, csh, sh, etc.



```
$ chsh //change default shell
```

Running a Shell Script



1. Absolute path (need x permission)

➤ `/home/buyhorse/shell/hello.sh`

➤ ``pwd` /hello.sh`

2. Current path (need x permission)

➤ `./hello.sh`

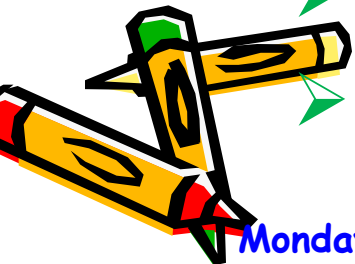
3. `sh/bash`

➤ `bash hello.sh`

4. shell environment

➤ `source hello.sh`

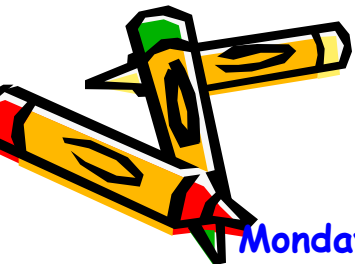
➤ `. hello.sh` (有空格)



Parsing the Command Line

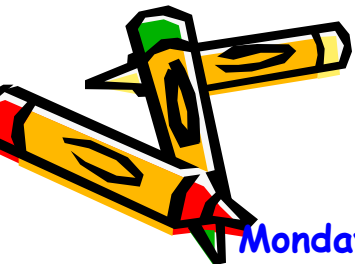
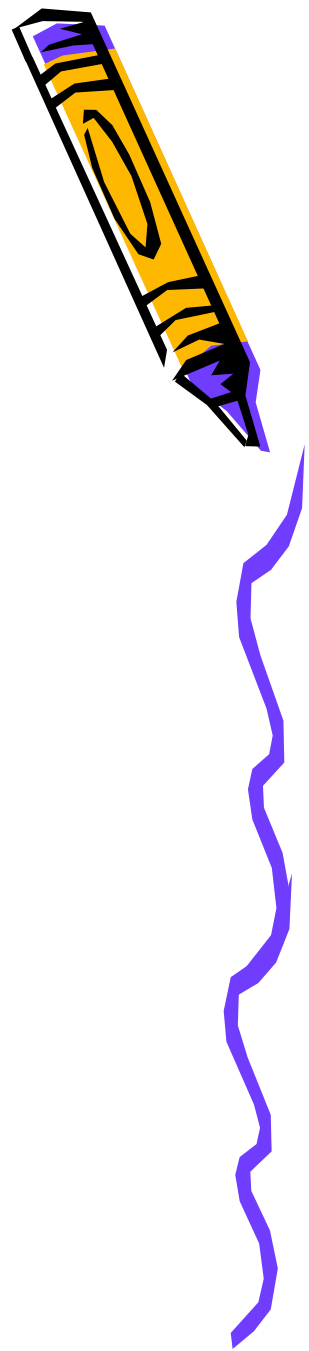


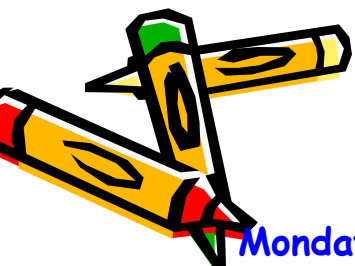
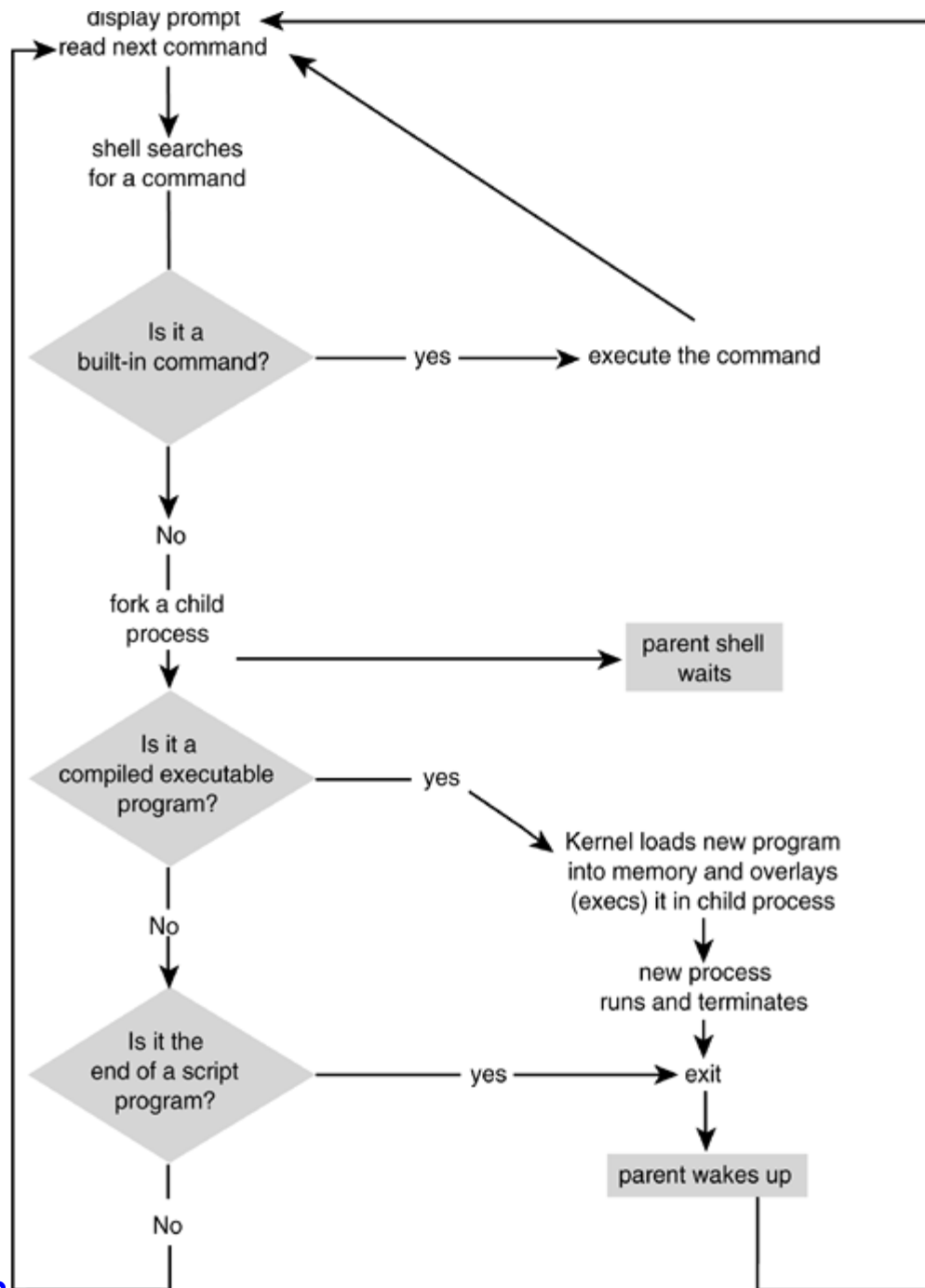
1. History substitution is performed (if applicable).
2. Command line is broken up into tokens, or words.
3. History is updated (if applicable).
4. Quotes are processed.
5. Alias substitution and functions are defined (if applicable).
6. Redirection, background, and pipes are set up.
7. Variable substitution (\$user, \$name, etc.) is performed.
8. Command substitution (echo "Today is `date`") is performed.
9. Filename substitution, called globbing (cat abc.??, rm *.c, etc.) is performed.
10. Command is executed.



Types of Commands

1. Aliases
2. Keywords
3. Functions
4. Built-in commands
5. Executable programs



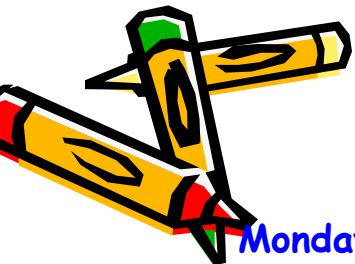


The Environment and Inheritance



- Ownership and Permissions
- The File Creation Mask
- Changing Permissions and Ownership

\$ id

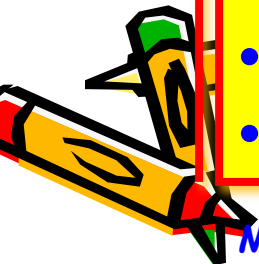


The Environment and Inheritance

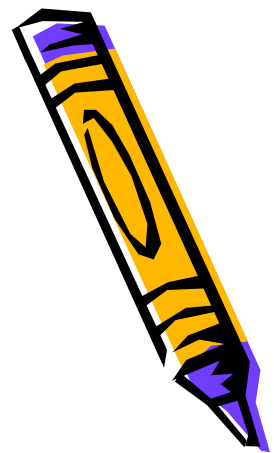


- Ownership and Permissions
- The File Creation Mask
- Changing Permissions and Ownership

- `$ umask`
- **Directory:** `777-umask` (掩码对应位去掉权限) `^umask`
- **File:** `666-umask` (掩码对应位去掉权限) `&~umask` why?
- `umask -S`
- `man umask`
- `man pam_umask`
- `/etc/login.defs` `002 → 022` (root)



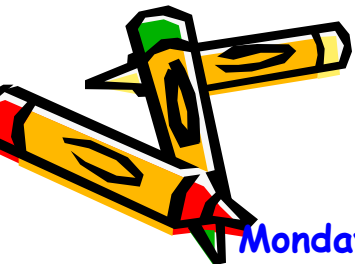
The Environment and Inheritance



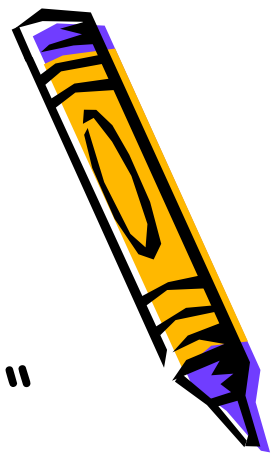
- Ownership and Permissions
- The File Creation Mask

• Changing Permissions and Ownership

- **chmod:** changes permissions on files and directories
- **chown:** changes the owner on files and directories
- **chgrp:** changes the group on files and directories



echo: Displaying Messages and Values



Syntax: `echo "Messages/$Variables"`

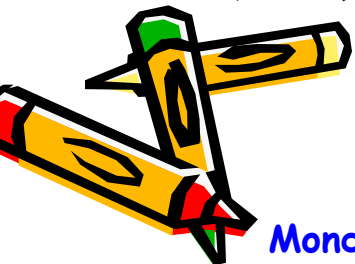
✓ `echo "Linux is popular"`

✓ `x=9`

✓ `echo "the value of x is $x"`

✓ `echo $variable`

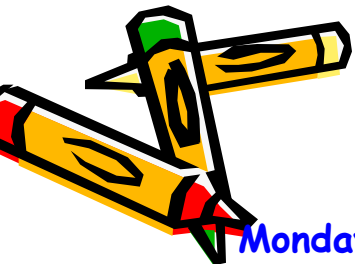
✓ `echo ${variable}**` // 保护变量不受其他字符的影响



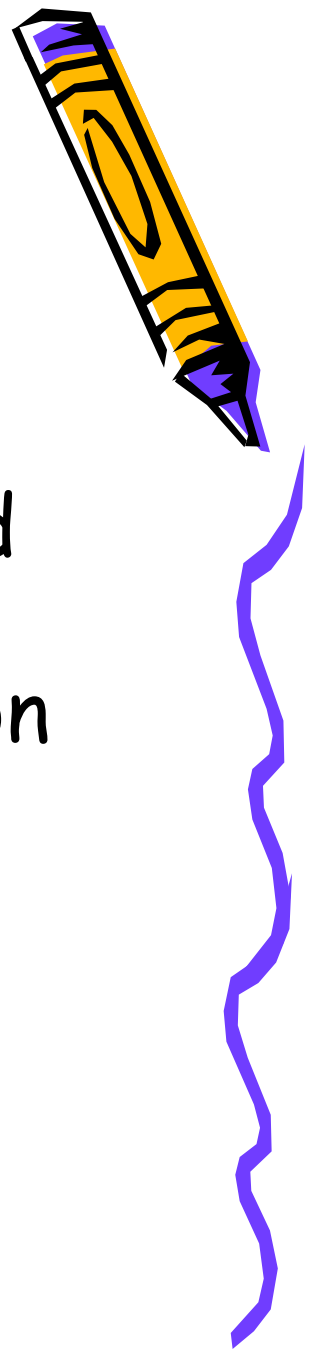
echo命令使用的特殊字符



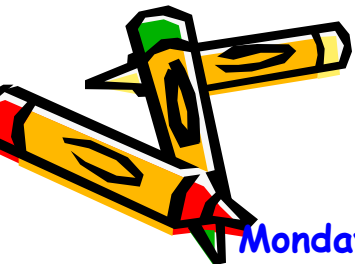
- `echo -n hello` //do not output the trailing newline
- `-e` //开启转义
 - ✓ `echo -e "hello \n world"` //new line
 - ✓ `echo -e "hello \tworld"` //horizontal tab
 - ✓ `echo -e "hello \vworld"` //vertical tab



Variables



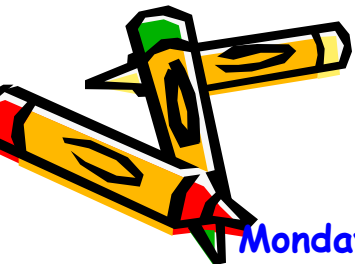
- The variables contain information used for customizing the shell, and information required by other processes so that they will function properly.
- two types of variables: local and environment



Two kinds of shell variables

- ***Environment variables***: available in the current shell and the programs invoked from the shell
- ***Regular shell variables (local)***: not available in programs invoked from this shell

- set
- env or printenv



Shell Variables

- To set regular variables: **(set)** varname=varvalue

```
> (set) s= "linux is easy"
```

```
> echo s
```

```
s
```

```
> echo $s
```

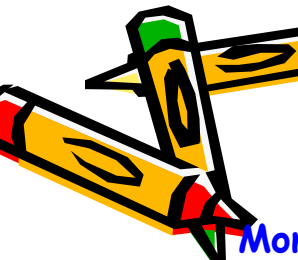
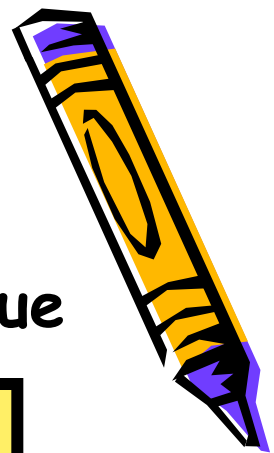
```
linux is easy
```

- Clearing out regular variables:

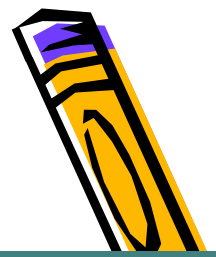
```
> unset myvar
```

```
> echo $myvar
```

```
myvar: Undefined variable.
```



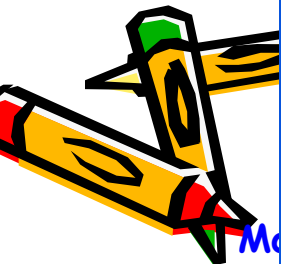
Setting variables



- variable1=value1 [variable2=value2...]

```
[buyhorse@server1 buyhorse]$ name=john
[buyhorse@server1 buyhorse]$ echo $name
john
[buyhorse@server1 buyhorse]$ name=john doe
-bash: doe: command not found
[buyhorse@server1 buyhorse]$ echo $name
john
[buyhorse@server1 buyhorse]$ name="john doe"
[buyhorse@server1 buyhorse]$ echo $name
john doe
[buyhorse@server1 buyhorse]$ name=john*
[buyhorse@server1 buyhorse]$ echo $name
john*
[buyhorse@server1 buyhorse]$ echo "The name of $name sounds familiar"
The name of john* sounds familiar
[buyhorse@server1 buyhorse]$ echo \ $name
$name
[buyhorse@server1 buyhorse]$ echo ' $name'
 $name
[buyhorse@server1 buyhorse]$
```

```
[buyhorse@server1 buyhorse]$ command=pwd
[buyhorse@server1 buyhorse]$ $command
/home/buyhorse
[buyhorse@server1 buyhorse]$ command=hello
[buyhorse@server1 buyhorse]$ $command
-bash: hello: command not found
[buyhorse@server1 buyhorse]$
```

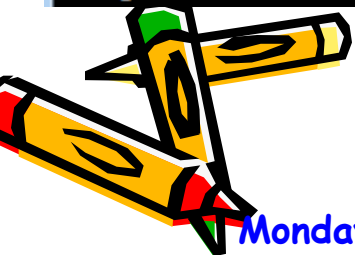


命令替换

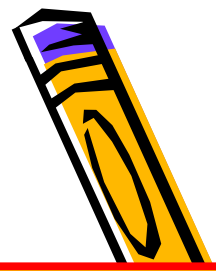


一个命令包含在反引号里时，**shell**执行该命令，然后用该命令的输出结果替换该命令（包括反引号）。

```
buyhorse@ubuntu:~/shell$ cmd=pwd
buyhorse@ubuntu:~/shell$ echo "the value of cmd is $cmd"
the value of cmd is pwd
buyhorse@ubuntu:~/shell$ cmd=`pwd`
buyhorse@ubuntu:~/shell$ echo "the value of cmd is $cmd"
the value of cmd is /home/buyhorse/shell
buyhorse@ubuntu:~/shell$ pwd
/home/buyhorse/shell
buyhorse@ubuntu:~/shell$ echo "the date and time is `date`"
the date and time is Fri May 4 08:45:02 CST 2018
buyhorse@ubuntu:~/shell$
```



部分只读的环境变量



- \$0 程序名称
- \$1-9 前9个命令行参数的值
- \$* 所有命令行参数的值
- @\$ 所有命令行参数的值
- \$# 命令行参数的个数
- \$\$ 当前进程的进程ID
- \$? 最近使用命令的退出状态
- \$! 最近后台进程的进程ID
- PS1 shell命令提示符
- PS4 shell调试提示符

0 OK!

1 Operation not permitted

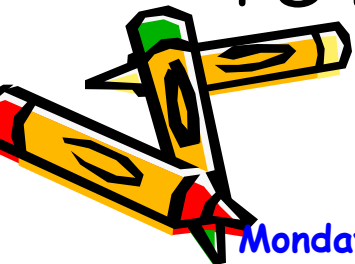
2 No such file or directory

13 Permission denied

127 command not found

130 terminated by Ctrl-C

~/.bashrc



Variables/Func export



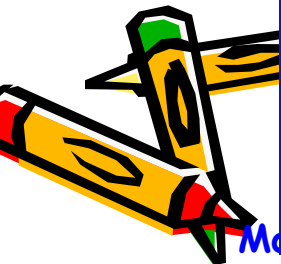
- 一个变量被创建时，它不会自动地为在它之后创建的进程所知。须通过**export**向后面的**shell**传递变量的值。
- **export [name-list]**
- **export -n name** //取消变量输出

```
buyhorse@ubuntu:~/shell$ cat display_name
echo $name
exit 0
buyhorse@ubuntu:~/shell$ bash display_name

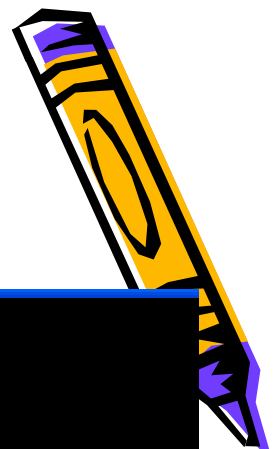
buyhorse@ubuntu:~/shell$ name=tom
buyhorse@ubuntu:~/shell$ bash display_name

buyhorse@ubuntu:~/shell$ export name
buyhorse@ubuntu:~/shell$ bash display_name
tom
buyhorse@ubuntu:~/shell$ export -n name
buyhorse@ubuntu:~/shell$ bash display_name

buyhorse@ubuntu:~/shell$
```



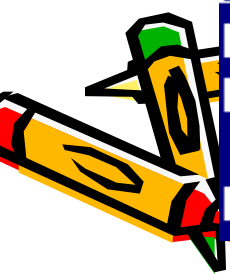

Variables/Func export



```
buyhorse@ubuntu:~/shell$ cat export_demo
#!/bin/sh
name="tom"
export name
sh display_change_name
sh display_name
exit 0
buyhorse@ubuntu:~/shell$ cat display_change_name
#!/bin/sh
echo $name
name="jerry"
echo $name
exit 0
buyhorse@ubuntu:~/shell$ bash export_demo
tom
jerry
tom
buyhorse@ubuntu:~/shell$
```



unset [name-list]



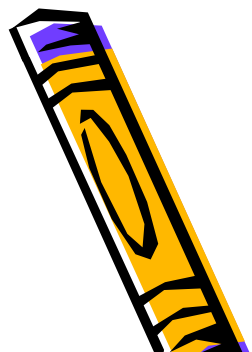
```
[buyhorse@server1 shell]$ name=apple place=dalian
[buyhorse@server1 shell]$ echo "$name $place"
apple dalian
[buyhorse@server1 shell]$ unset name
[buyhorse@server1 shell]$ echo "$name"

[buyhorse@server1 shell]$ echo "$place"
dalian
[buyhorse@server1 shell]$ unset name place
[buyhorse@server1 shell]$ echo "$name $place"

[buyhorse@server1 shell]$ name=apple
[buyhorse@server1 shell]$ echo $name
apple
[buyhorse@server1 shell]$ name=
[buyhorse@server1 shell]$ echo $name

[buyhorse@server1 shell]$
```


readonly [name-list]



```
[buyhorse@server1 shell]$ name=apple place=dalian
[buyhorse@server1 shell]$ readonly name place
[buyhorse@server1 shell]$ name=banana
-bash: name: readonly variable
[buyhorse@server1 shell]$ place=beijing
-bash: place: readonly variable
[buyhorse@server1 shell]$ readonly
declare -ar BASH_VERSION='([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]'
declare -ir EUID="793"
declare -ir PPID="2024"
declare -r SHELL_OPTS="braceexpand:emacs:hashall:histexpand:history:interactive-"
declare -ir UID="793"
declare -r name="apple"
declare -r place="dalian"
[buyhorse@server1 shell]$
```

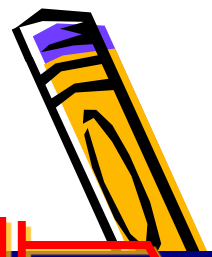
read [variable-list]

read variable1 [variable2 ...]

- Read one line of standard input
- Assign each word to the corresponding variable, with the leftover words assigned to last variables
- If only one variable is specified, the entire line will be assigned to that variable.

```
[buyhorse@server1 buyhorse]$ cat read_demo
#!/bin/sh
echo "enter input: \c"
read line
echo "You entered: $line"
echo "Enter another line: \c"
read word1 word2 word3
echo "The first word is: $word1"
echo "The second word is: $word2"
echo "The rest of the line is: $word3"
exit 0
[buyhorse@server1 buyhorse]$ sh read_demo
enter input: \c
unix rules the network computing world
You entered: unix rules the network computing world
Enter another line: \c
unix rules the network computing world
The first word is: unix
The second word is: rules
The rest of the line is: the network computing world
[buyhorse@server1 buyhorse]$
```

向shell脚本传递参数



如果参数>9个? \${10}

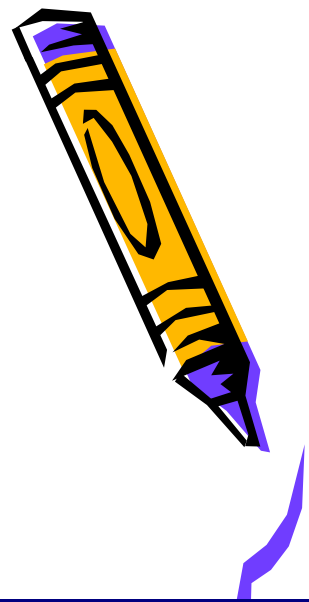
```
[buyhorse@server1 shell]$ cat cmdargs_demo
#!/bin/sh
echo "The command name is: $0."
echo "The number of command line arguments passed as parameters are $#."
echo "The value of the command line arguments are: $1 $2 $3 $4 $5 $6 $7 $8 $9."
echo "Anoter way to display values of all of the arguments: $@"
echo "Yet another way is: $*."
exit 0

[buyhorse@server1 shell]$ sh cmdargs_demo a b c d e f g h
The command name is: cmdargs_demo.
The number of command line arguments passed as parameters are 8.
The value of the command line arguments are: a b c d e f g h .
Anoter way to display values of all of the arguments: a b c d e f g h.
Yet another way is: a b c d e f g h.

[buyhorse@server1 shell]$ sh cmdargs_demo one two 3 4 five
The command name is: cmdargs_demo.
The number of command line arguments passed as parameters are 5.
The value of the command line arguments are: one two 3 4 five .
Anoter way to display values of all of the arguments: one two 3 4 five.
Yet another way is: one two 3 4 five.

[buyhorse@server1 shell]$
```

向shell脚本传递参数



- cat shift_demo

```
[buyhorse@server1 shell]$ sh shift_demo 1 2 3 4 5 6 7 8 9 10 11 12
The command name is: shift_demo.
The arguments are: 1 2 3 4 5 6 7 8 9 10 11 12.
The first three arguments are: 1 2 3.
The command name is: shift_demo.
The arguments are: 2 3 4 5 6 7 8 9 10 11 12.
The first three arguments are: 2 3 4.
The command name is: shift_demo.
The arguments are: 5 6 7 8 9 10 11 12.
The first three arguments are: 5 6 7.
[buyhorse@server1 shell]$
```



set [options] [argument-list]

把各位置参数的值依次设为argument-list里指定的参数

```
buyhorse@ubuntu:~/shell$ date
Fri Apr 28 08:45:22 CST 2017
buyhorse@ubuntu:~/shell$ set `date`
buyhorse@ubuntu:~/shell$ echo "$6"
2017
buyhorse@ubuntu:~/shell$ echo "$6-$2-$3"
2017-Apr-28
buyhorse@ubuntu:~/shell$
```

set [options] [argument-list]



- cat set_demo
- sh set_demo set_demo

```
[buyhorse@server1 shell]$ ls -il
```

总用量 28

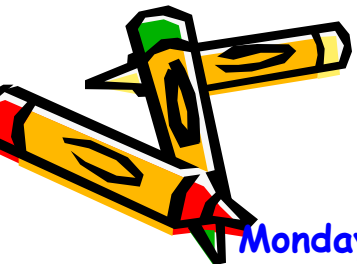
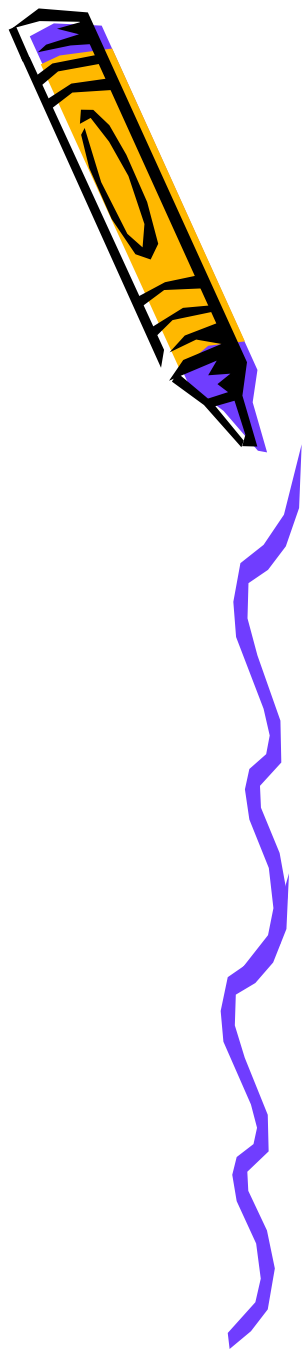
1601567	-rwx-----	1	buyhorse	buyhorse	298	5月	5 16:07	cmdargs_demo
1601564	-rwx-----	1	buyhorse	buyhorse	57	5月	5 14:56	display_changee
1601556	-rw-rw-r--	1	buyhorse	buyhorse	18	5月	5 14:52	display_name
1601552	-rwx-----	1	buyhorse	buyhorse	84	5月	5 14:55	export_demo
1601565	-rwx-----	1	buyhorse	buyhorse	235	5月	5 16:05	read_demo
1601569	-rwx-----	1	buyhorse	buyhorse	141	5月	5 16:18	set_demo
1601568	-rwx-----	1	buyhorse	buyhorse	361	5月	5 16:07	shift_demo

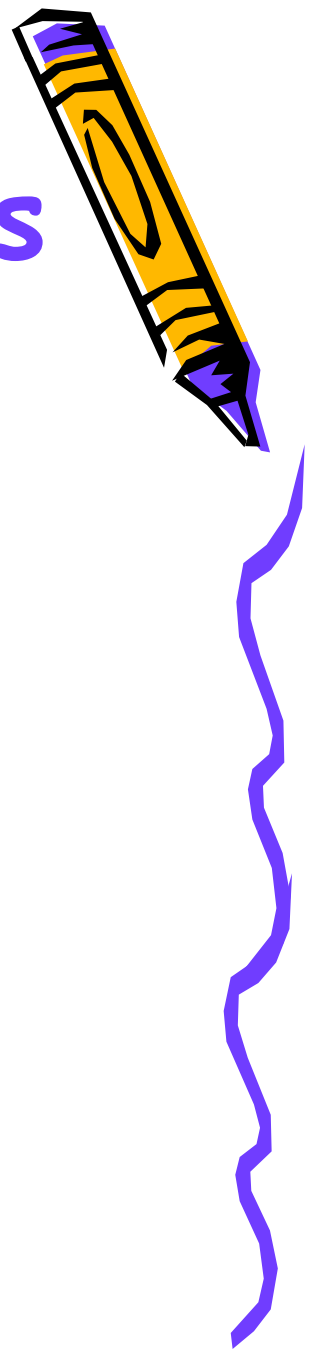
```
[buyhorse@server1 shell]$
```



shell structure

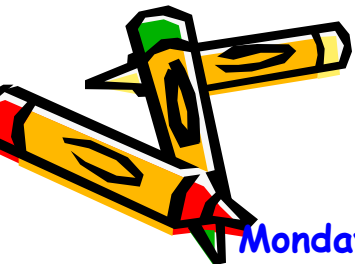
- expr
- test
- if then
- for
- while
- until
- break and continue
- case
- let



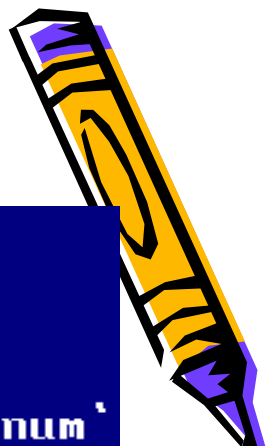


expr: Evaluating Expressions

- `expr 3+6`
- `expr 3 + 6`
- `x=3 y=6`
- `expr $x + $y`

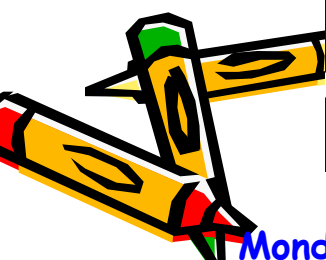


Evaluate expressions: expr



```
[buyhorse@server1 shell]$ num=10
[buyhorse@server1 shell]$ num=`expr $num + 1`
[buyhorse@server1 shell]$ echo $num
11
[buyhorse@server1 shell]$ num=`expr $num \+ $num`
[buyhorse@server1 shell]$ echo $num
22
[buyhorse@server1 shell]$ num=`expr $num \* $num`
[buyhorse@server1 shell]$ echo $num
484
[buyhorse@server1 shell]$ echo `expr $num /4`
expr: syntax error

[buyhorse@server1 shell]$ echo `expr 484 % 10`
4
[buyhorse@server1 shell]$ echo `expr 484 / 4`
121
```



```
[buyhorse@server1 shell]$ value=`expr 12345 + 54321`
[buyhorse@server1 shell]$ echo $value
66666
[buyhorse@server1 shell]$
```

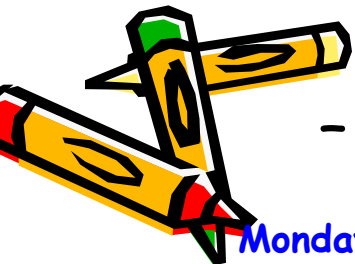
test

- Command **test** is a built-in command
- Syntax

test expression

[expression]

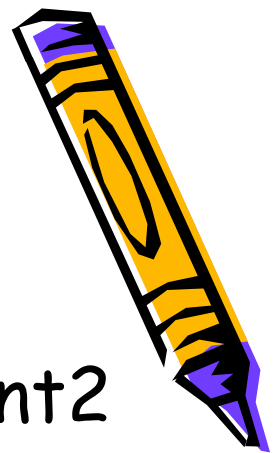
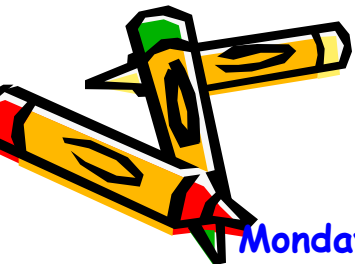
- The test command evaluate an expression
- Returns a condition code indicating that the expression is either true (0) or false (not 0)
- Argument
 - Expression contains one or more criteria
 - Logical AND operator to separate two criteria: -a
 - Logical OR operator to separate two criteria: -o
 - Negate any criterion: !
 - Group criteria with parentheses
 - Separate each element with a SPACE



Test Criteria

- Test Operator for *integers*: int1 relop int2

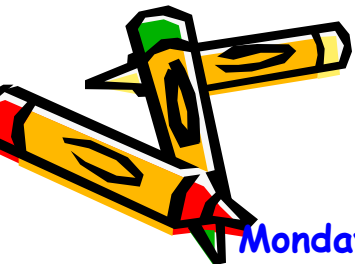
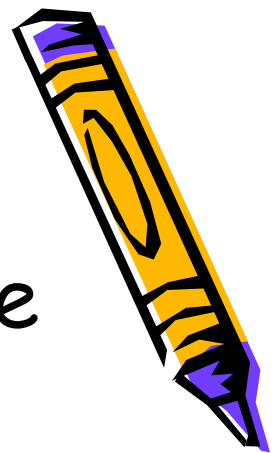
Relop	Description
-gt	Greater than
-ge	Greater than or equal to
-eq	Equal to
-ne	Not equal to
-le	Less than or equal to
-lt	Less than





Exercise

- Create a shell script to check there is at least one parameter
 - Something like this:

```
...  
if test $# -eq 0  
then  
    echo "you must supply at  
least one arguments"  
    exit 1  
fi  
...
```



Exercise



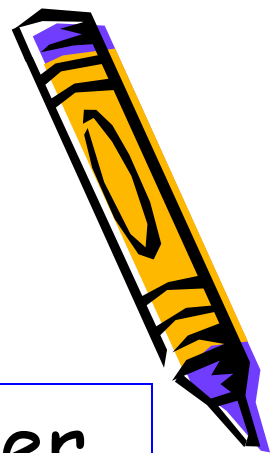
```
[buyhorse@server1 shell]$ cat debug_demo
#!/bin/sh
echo "enter a digit: "
read var1
if [ "$var1" -ge 1 -a "$var1" -le 9 ]
then
echo "good input!"
fi
exit 0
[buyhorse@server1 shell]$ sh debug_demo
enter a digit:
2
good input!
[buyhorse@server1 shell]$ sh debug_demo
enter a digit:
10
[buyhorse@server1 shell]$
```

Test Criteria

- The test built-in options for files

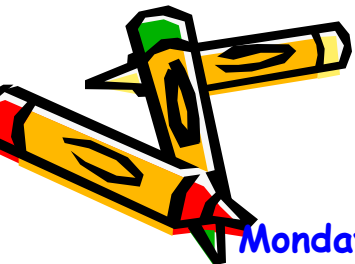
Option	Test Performed on file
-d filename	Exists and is a directory file
-f filename	Exists and is a regular file
-r filename	Exists and it readable
-s filename	Exists and has a length greater than 0
-u filename	Exists and has setuid bit set
-w filename	Exists and it writable
-x filename	Exists and it is executable
FILE1 -ot FILE2	FILE1 is older than FILE2

Exercise



- Check whether or not the parameter is a non-zero readable file name
 - Continue with the previous script and add something like

```
if [ -r "$filename" -a -s "$filename" ]  
then  
    ... ..  
fi
```

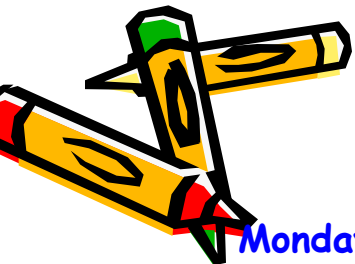


Test Criteria



- String testing

Criteria	meaning
-n string	True if string has a length greater than zero (True if nozero)
-z string	True if string has a length of zero
String1 = string2	True if string1 is equal to string2
String1 != string2	True if string1 is not equal to string2



Exercise- yesno

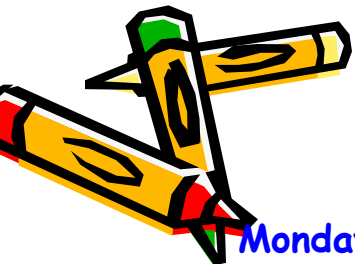
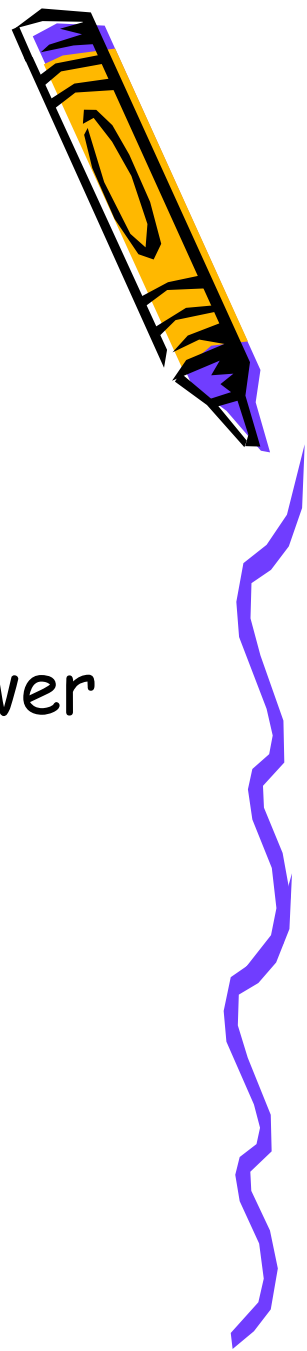
- Check users confirmation

- First, read user input

```
echo -n "Please confirm: [Yes | No] "  
read user_input
```

- Then, compare it with standard answer
'yes'

```
if [ "$user_input" = Yes ]  
then  
    echo "Thanks for your confirmation!"  
fi
```



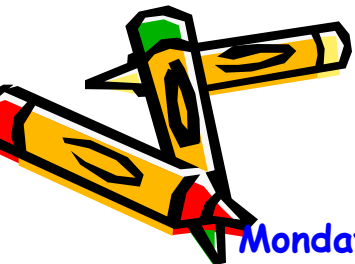
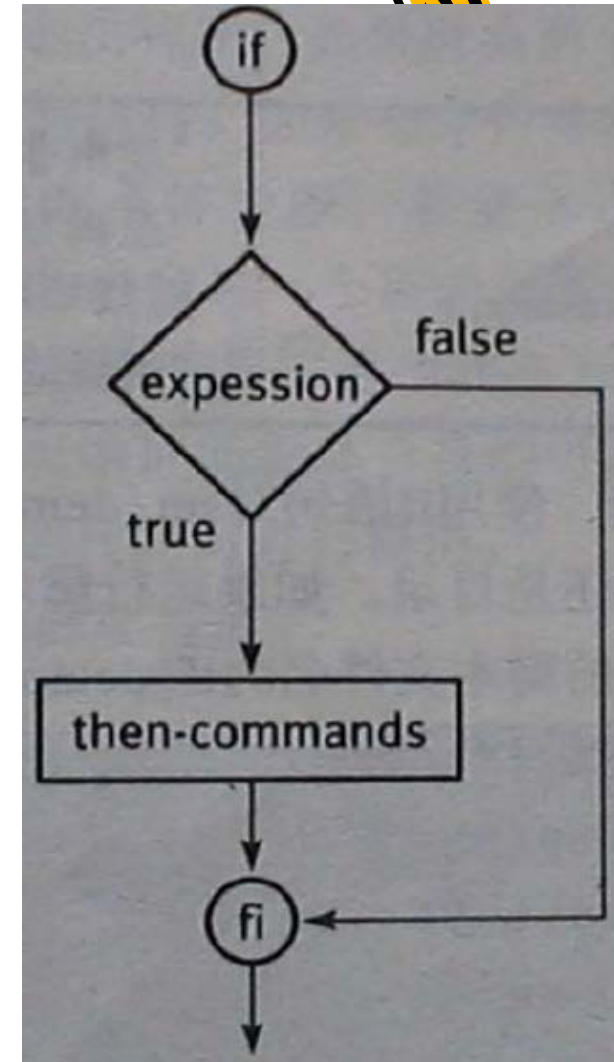
if ... then

- Structure

if expression
then
then-command
fi

Example:

```
if test "$word1" = "$word2"  
then  
    echo "Match"  
fi
```



if...then...else

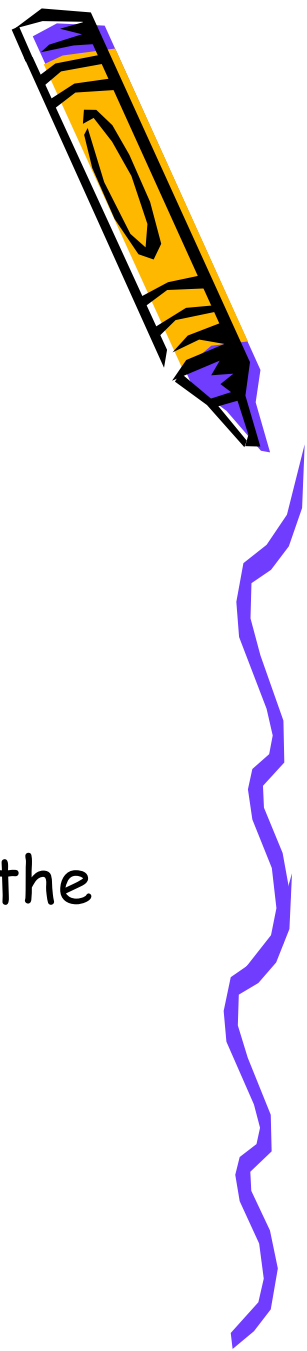
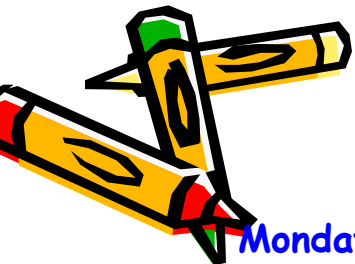
- Structure

```
if test-command
then
    commands
else
    commands
fi
```

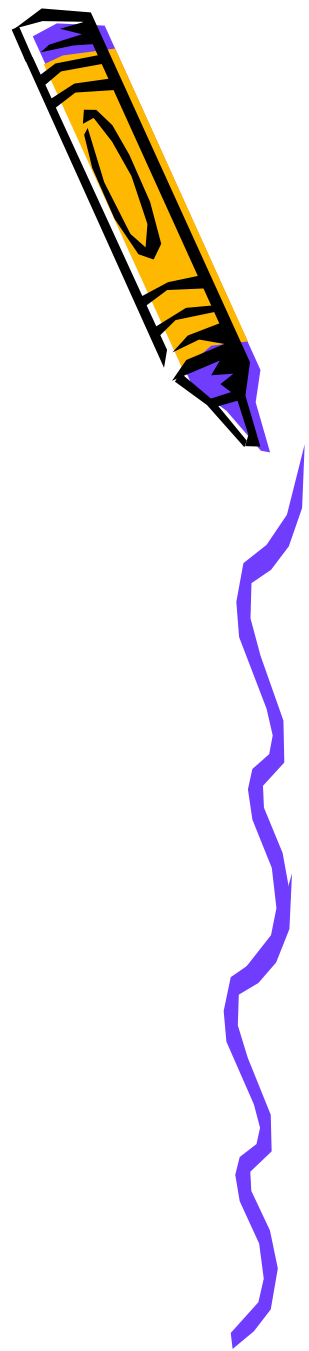
- You can use semicolon (;) ends a command the same way a NEWLINE does.

```
if [ ... ]; then
    ... ..
fi
```

```
if [ 5 = 5 ]; then echo "equal"; fi
```

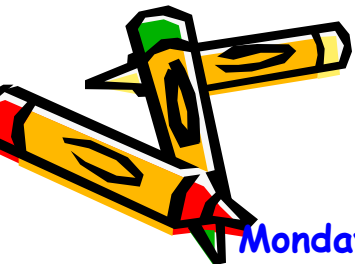


if...then...elif



- Structure

```
if test-command
then
    commands
elif test-command
then
    commands
.
.
.
else
    commands
fi
```



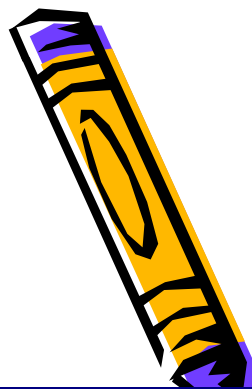
Exercise : if_demo



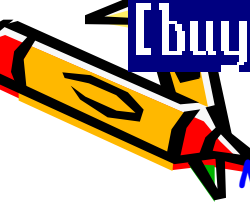
```
#!/bin/sh
if test $# -eq 0
then
    echo "usage: $0 ordinary_file"
    exit 1
fi
if test $# -gt 1
then
    echo "usage: $0 ordinary_file"
    exit 1
fi
if test -f "$1"
then
    filename="$1"
    set `ls -il $filename`
    inode="$1"
    size="$6"
    echo "Name \tInode \tSize"
    echo
    echo "$filename \t$inode \t$size"
    exit 0
fi
echo "$0: argument must be an ordinary file"
exit 1
```



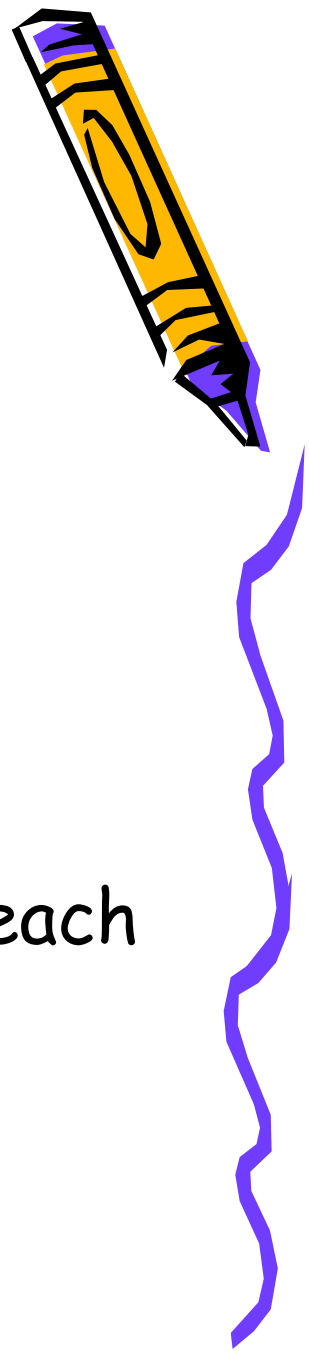
Exercise (cont')



```
[buyhorse@server1 shell]$ sh if_demo
usage: if_demo ordinary_file
[buyhorse@server1 shell]$ sh if_demo set_demo shift_demo
usage: if_demo ordinary_file
[buyhorse@server1 shell]$ sh if_demo dirtest/
if_demo: argument must be an ordinary file
[buyhorse@server1 shell]$ sh if_demo set_demo
Name \tInode \tSize
set_demo \t1601569 \t141
[buyhorse@server1 shell]$
```



for

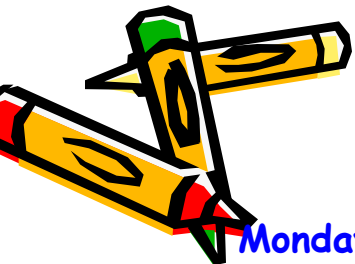


- Structure

```
for loop-index  
do  
    commands  
done
```

- Automatically takes on the value of each of command line arguments, one at a time. Which implies

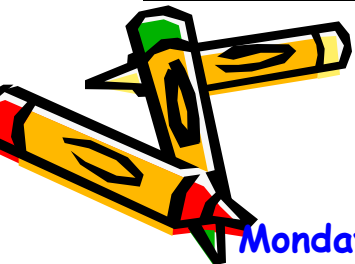
for arg in "\$@"



Exercise (for_demo)



```
buyhorse@ubuntu-server-1804:~/shell$ cat for_demo
#!/bin/sh
for animal in dog cat fox kitty horse rabbit
do
    echo "$animal"
done
exit 0
buyhorse@ubuntu-server-1804:~/shell$ bash for_demo
dog
cat
fox
kitty
horse
rabbit
buyhorse@ubuntu-server-1804:~/shell$
```

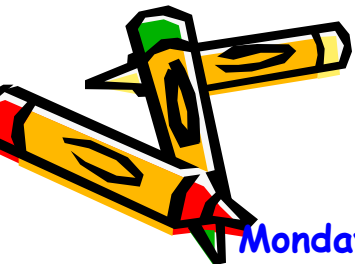


Exercise:

(ls-demo & filter-txt)



```
buyhorse@ubuntu-server-1804:~/shell$ cat ls-demo
for file in `ls`
do
echo $file
done
buyhorse@ubuntu-server-1804:~/shell$ cat filter-txt
for file in *.txt
do
echo $file
done
buyhorse@ubuntu-server-1804:~/shell$ █
```



for... in



- Structure

```
for loop-index in argument_list  
do
```

```
    commands
```

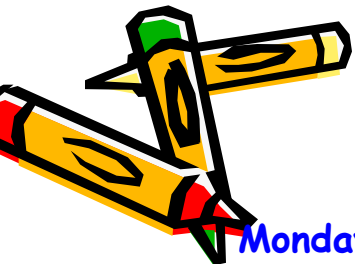
```
done
```

Example: **sh dirlist**

```
for file in *  
do  
    if [ -d "$file" ]; then  
        echo $file  
    fi  
done
```

如何运行:

1. ls |bash dirlist
2. bash dirlist



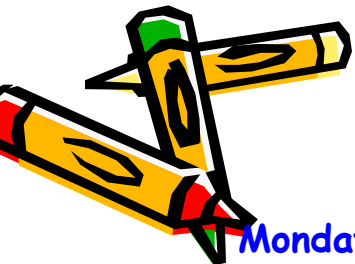
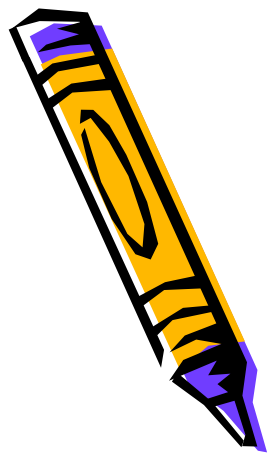
while

- Structure

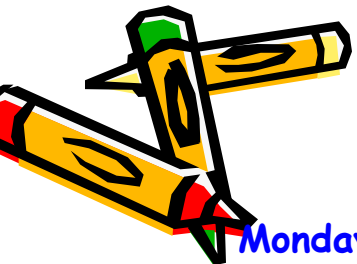
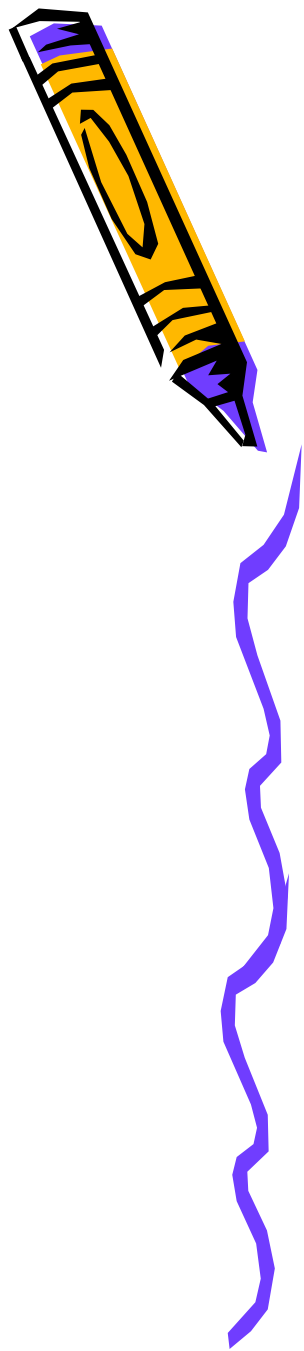
```
while test_command
do
    commands
done
```

Example:

```
while [ "$number" -lt 10 ]
do
    ... ..
    number=`expr $number + 1`
done
```



Exercise: `while_demo`



until

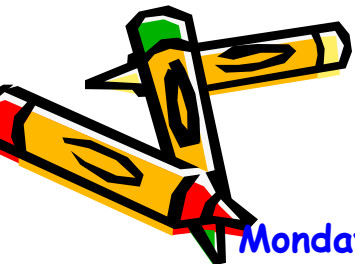


- Structure

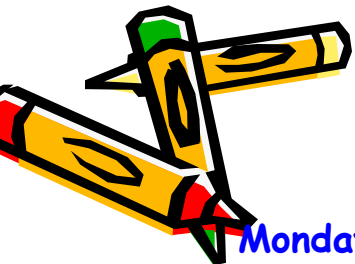
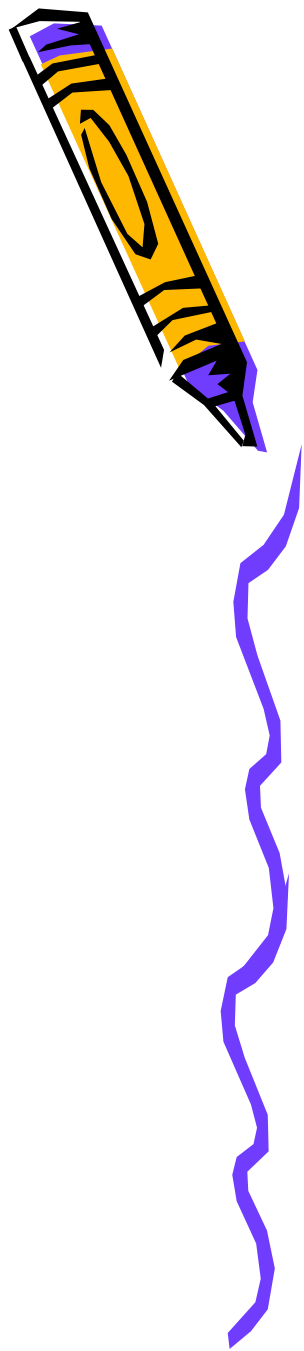
```
until test_command
do
    commands
done
```

Example:

```
secretname=jerry
until [ "$name" = "$secretname" ]
do
    echo " Your guess: \c"
    read name
done
```

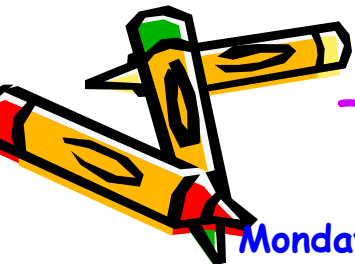


Exercise: until_demo



break and continue

- Interrupt for, while or until loop
- The continue statement
 - Transfer control to the statement **TO** the done statement
 - Skip the test statements for the current iteration
 - Continues execution of the loop
- The break statement
 - transfer control to the statement **AFTER** the done statement
 - terminate execution of the loop



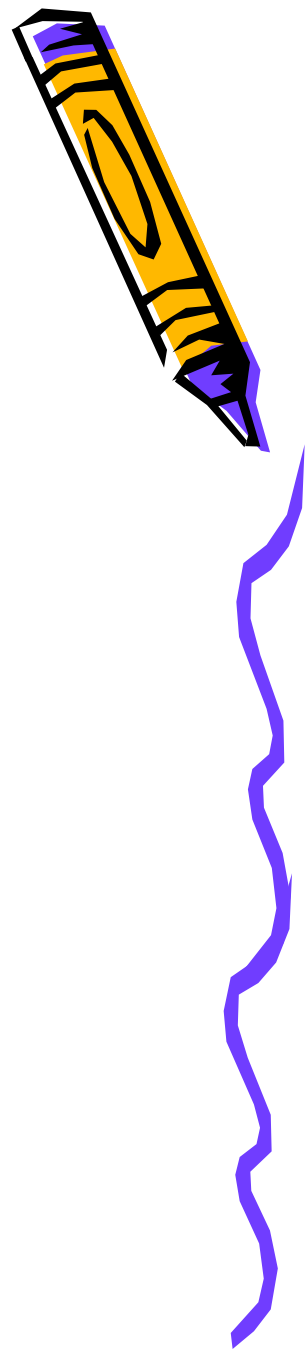
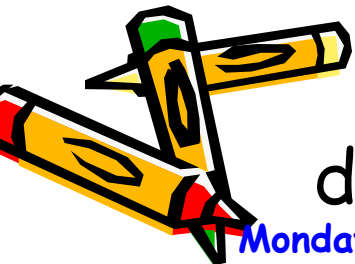
Example: breakcontinue

```
for index in 1 2 3 4 5 6 7 8 9 10  
do
```

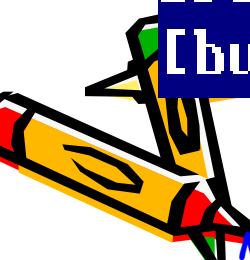

```
    if [ $index -le 3 ]; then  
        echo continue  
        continue  
    fi
```

```
    echo $index  
    if [ $index -ge 8 ]; then  
        echo "break"  
        break  
    fi
```


```
done
```



breakcontinue



```
[buyhorse@server1 shell1]$ sh breakcontinue
continue
continue
continue
4
5
6
7
8
break
[buyhorse@server1 shell1]$
```

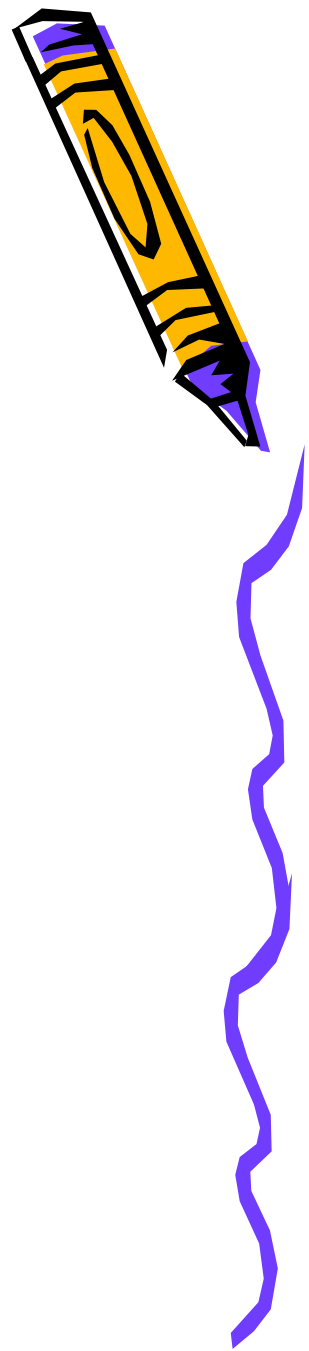
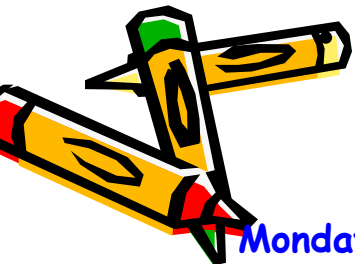


case

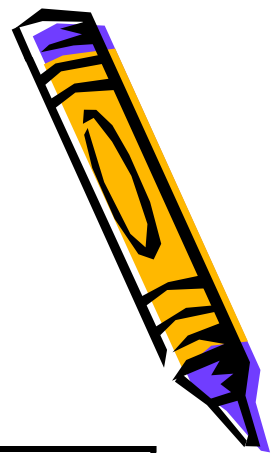
- Structure

```
case test_string in
    pattern-1 )
        commands_1
        ;;
    pattern-2 )
        commands_2
        ;;
    ... ..
esac
```

default case: catch all pattern
*)

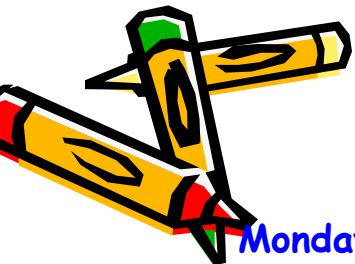


case

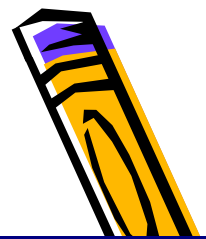


- Special characters used in patterns

Pattern	Matches
*	Matches any string of characters.
?	Matches any single character.
[...]	Defines a character class. A hyphen specifies a range of characters
	Separates alternative choices that satisfy a particular branch of the case structure



Exercise: case_demo

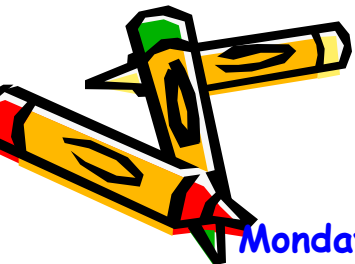
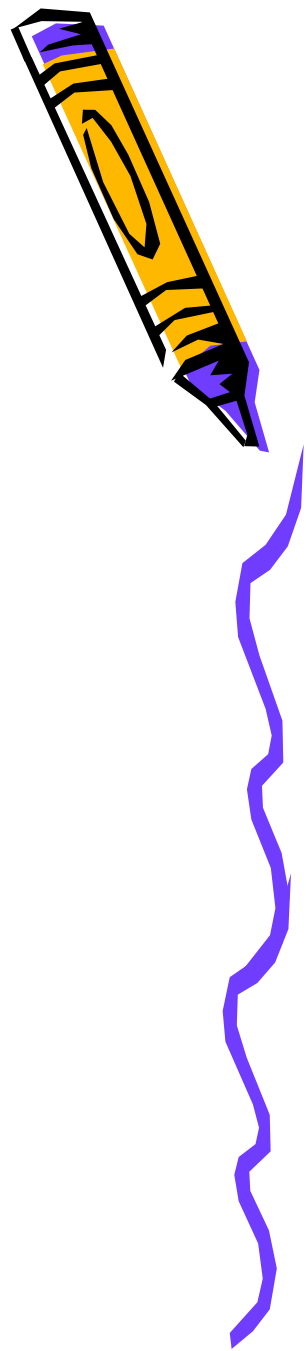


```
[buyhorse@server1 shell]$ cat case_demo
#!/bin/sh
echo "Use one of the following options:"
echo "d: To display today's date and time"
echo "l: To see the listing of files in your present working directory"
echo "w: To see who's logged in"
echo "q: To quit this program"
echo "Enter your options and hit <Enter>:"
read option
case "$option" in
    d) date
      ;;
    l) ls
      ;;
    w) who
      ;;
    q) exit 0
      ;;
    *)
esac
exit 0
[buyhorse@server1 shell]$
```



Example: case_1

```
[bughorse@server1 shell]$ cat case_1
#!/bin/sh
echo "\n Command MENU\n"
echo "a. Current data and time"
echo "b. Users currently logged in"
echo "c. Name of the working directory\n"
echo "Enter a,b, or c: \c"
read answer
echo
case "$answer" in
a)
date
;;
b)
who
;;
c)
pwd
;;
*)
echo "There is no selection: $answer"
;;
esac
```



case_demo revised

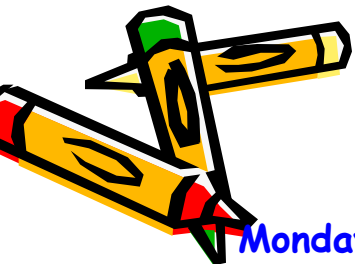
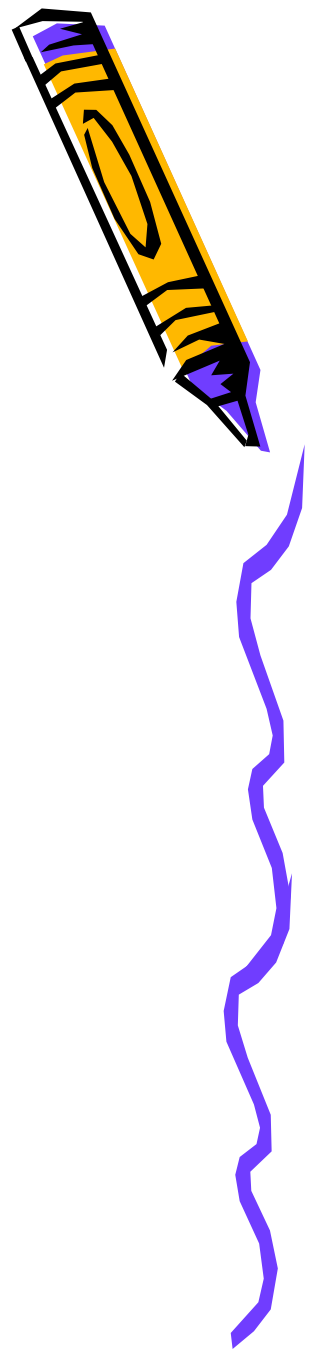


```
[bughorse@server1 shell]$ cat case_demo_revise
#!/bin/sh
echo "Use one of the following options:"
echo "d or D: To display today's date and time"
echo "l or L: To see the listing of files in your present working directory"
echo "w or W: To see who's logged in"
echo "q or Q: To quit this program"
echo "Enter your options and hit <Enter>:"
read option
case "$option" in
    d|D) date
        ;;
    l|L) ls
        ;;
    w|W) who
        ;;
    q|Q) exit 0
        ;;
    *) echo "Invalid option; try running the program again."
        exit 1
        ;;
esac
exit 0
[bughorse@server1 shell]$
```



Example: bundle

- `sh bundle file1 file2`



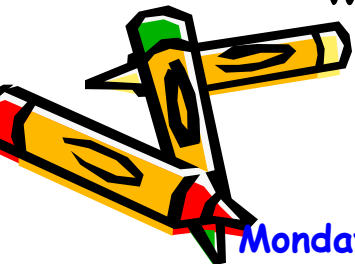
Pipelines again



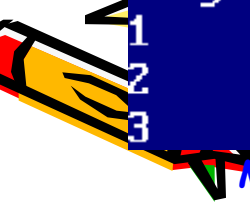
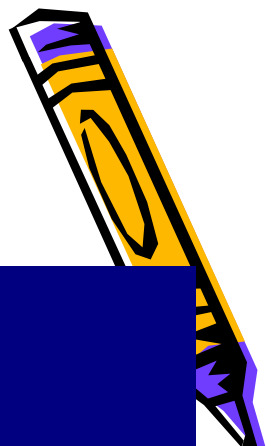
- `w | wc -l` //memory operation
- `w >temp.txt`
- `wc -l temp.txt`
- `rm temp.txt` //io operation

```
[buyhorse@server1 shell1]$ ls /dev/hda*
/dev/hda      /dev/hda14  /dev/hda2   /dev/hda25  /dev/hda30  /dev/hda7
/dev/hda1     /dev/hda15  /dev/hda20  /dev/hda26  /dev/hda31  /dev/hda8
/dev/hda10    /dev/hda16  /dev/hda21  /dev/hda27  /dev/hda32  /dev/hda9
/dev/hda11    /dev/hda17  /dev/hda22  /dev/hda28  /dev/hda4
/dev/hda12    /dev/hda18  /dev/hda23  /dev/hda29  /dev/hda5
/dev/hda13    /dev/hda19  /dev/hda24  /dev/hda3   /dev/hda6
[buyhorse@server1 shell1]$ ls /dev/hda* | wc -l
33
[buyhorse@server1 shell1]$
```

`who | grep "hacker" | mail -s "hacker's terminal" root@admin.com`



Exercise: countup




```
[buyhorse@server1 shell]$ cat countup
#!/bin/sh
if [ $# != 1 ]
then
echo "Usage: $0 integer-argument"
exit 1
fi

target="$1"      # Set target to the number passed at the command line
current=1        #the first number to be displayed

# Loop here until the current number becomes greater than the target
while [ $current -le $target ]
do
echo "$current"
current=`expr $current + 1 `
done
echo
exit 0

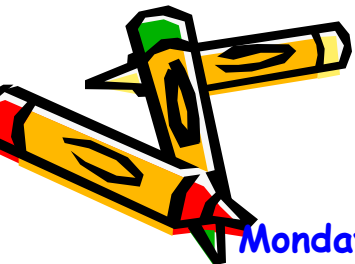
[buyhorse@server1 shell]$ sh countup 3
1
2
3
```



Exercise: addall



```
[buyhorse@server1 shell]$ sh addall
Usage: addall number-list
[buyhorse@server1 shell]$ sh addall 123 321
the sum of the given 2 numbers is 444.
[buyhorse@server1 shell]$ sh addall 123 321 333
the sum of the given 3 numbers is 777.
[buyhorse@server1 shell]$
```



Another expressions: let

- 格式: `let arg1 [arg2]`
- 说明:
 - 与`expr`命令相比, `let`命令更简洁直观
 - 当运算符中有`<`、`>`、`&`、`|`等符号时, 同样需要用引号(单引号、双引号)或者斜杠来修饰运算符

```
[buyhorse@server1 shell]$ let s=(2+3)*4
[buyhorse@server1 shell]$ echo $s
20
[buyhorse@server1 shell]$
```

here doc



command << [-] **input_marker**

...input data ...

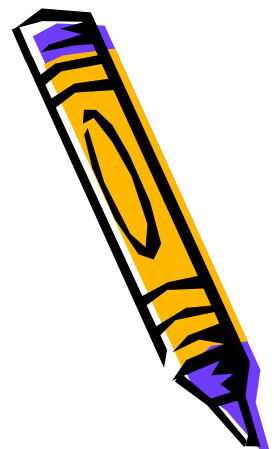
input_marker

作用：执行**command**命令，其输入来自**here**文档——从开始到结束标记之间的数据。

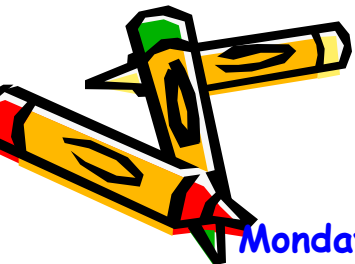
```
[buyhorse@server1 shell]$ cat here doc_demo
#!/bin/sh
cat << DataTag
this is a simple use of the here document. these data are the input to
the cat command.
DataTag

[buyhorse@server1 shell]$ sh here doc_demo
this is a simple use of the here document. these data are the input to
the cat command.
[buyhorse@server1 shell]$
```

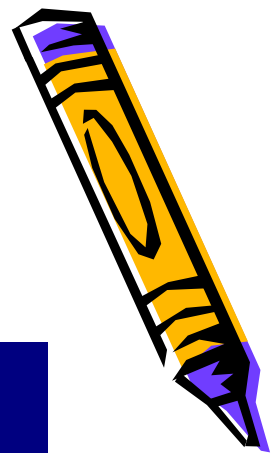
Exercise: grep_hello



```
buyhorse@ubuntu-server-1804:~/shell$ cat grep_hello
#!/bin/sh
grep "hello" <<EOF
this some sample text.
here is a line with hello in it.
here is anothe line with hello.
no more lines with that words.
EOF
```



Debugging shell scripts



```
[buyhorse@server1 shell]$ cat -n debug_demo
 1  #!/bin/sh
 2  echo "enter a digit: "
 3  read var1
 4  if ["$var1" -ge 1 -a "$var1" -le 9 ]
 5  then
 6  echo "good input!"
 7  fi
 8  exit 0

[buyhorse@server1 shell]$
[buyhorse@server1 shell]$ sh debug_demo
enter a digit:
4
debug_demo: line 4: [4: command not found
[buyhorse@server1 shell]$
```

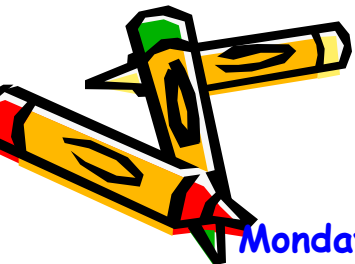


Debugging cont'

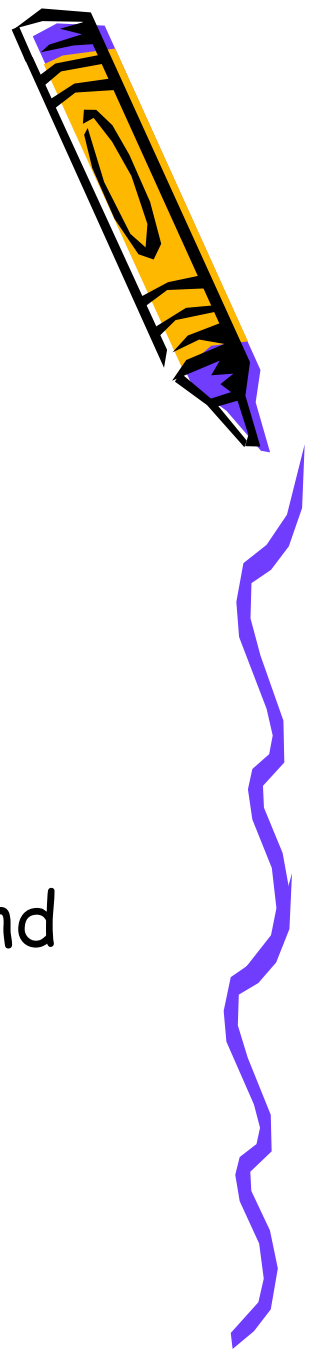


```
[buyhorse@server1 shell]$ sh -x debug_demo
+ echo 'enter a digit: '
enter a digit:
+ read var1
4
+ '[4' -ge 1 -a 4 -le 9 ']'
debug_demo: line 4: [4: command not found
+ exit 0
[buyhorse@server1 shell]$
```

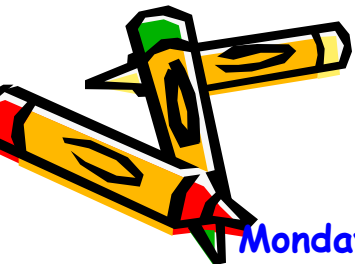
- PS4 recall



Example 1



- Write a Bourne shell script that accepts an arbitrarily long list of integers on the command line, and reports their sum and average (integer)
 - **First solution**: uses the `shift` command
 - **Second solution**: uses a `for` loop and avoids `shift`





```
#!/bin/sh
```

```
sum=0
```

```
numvals=$#
```

```
if [ $numvals -gt 0 ]
```

```
then
```

```
    while [ $# -gt 0 ]
```

```
    do
```

```
        sum=`expr $sum + $1`
```

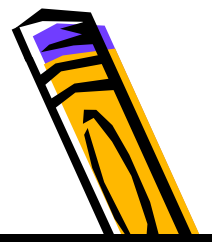
```
        shift
```

```
    done
```

```
# cont' d..
```



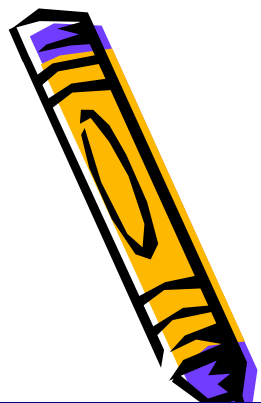
**First solution uses
the command shift**



```
# ..cont' d  
    echo Sum is $sum  
    echo Average is `expr $sum / $numvals`  
else  
    echo Error in use of $0: integer arguments expected  
fi
```



average-1 result



```
[buyhorse@server1 shell]$ sh average-1 4 6
sum is 10
average is 5
[buyhorse@server1 shell]$ sh average-1
error in use of average-1: integer arguments expected
[buyhorse@server1 shell]$ sh average-1 1234 4321
sum is 5555
average is 2777
[buyhorse@server1 shell]$ sh average-1 1 12 123 1234 12345
sum is 13715
average is 2743
[buyhorse@server1 shell]$
```



```
#!/bin/sh
```

```
if [ $# -gt 0 ]
```

```
then
```

```
    sum=0
```

```
    for k in $*
```

```
    do
```

```
        sum=`expr $sum + $k`
```

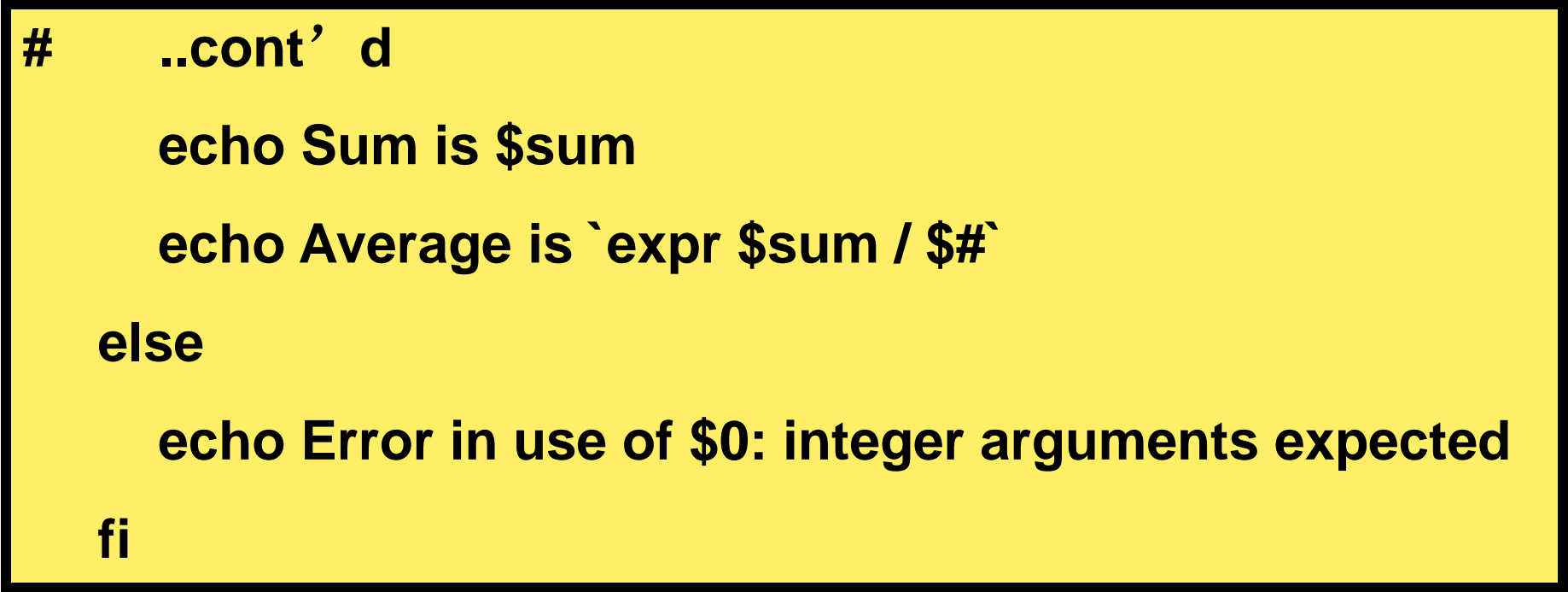
```
    done
```

```
#  cont' d..
```

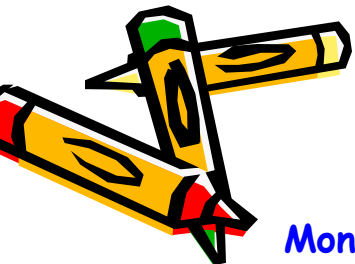


**Second solution uses a
for loop and avoids the
command shift**





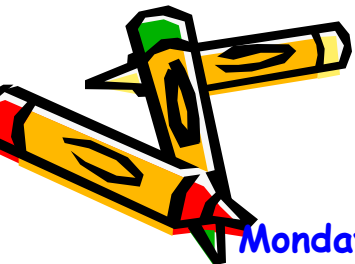
```
#    ..cont' d
    echo Sum is $sum
    echo Average is `expr $sum / $#`
else
    echo Error in use of $0: integer arguments expected
fi
```



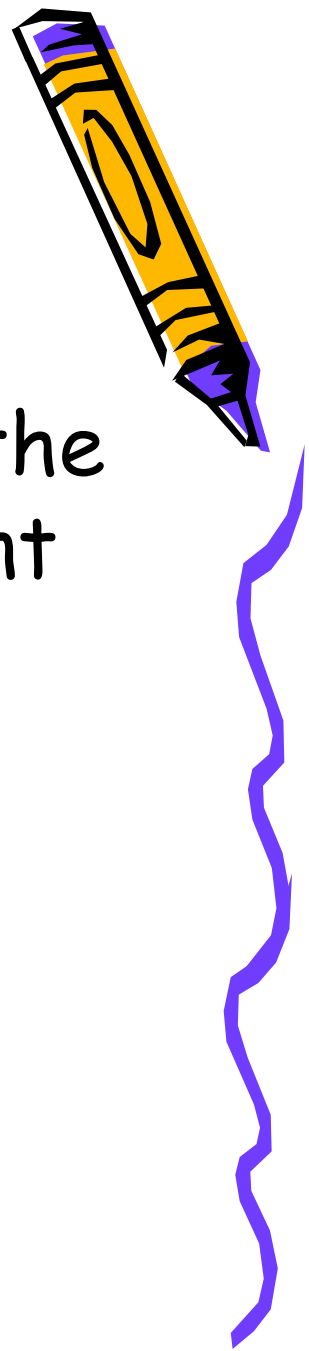
average-2 result



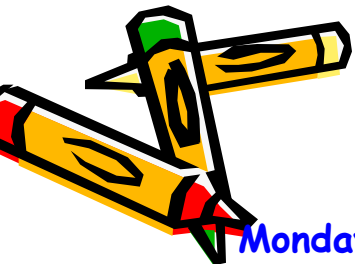
```
[buyhorse@server1 shell]$ sh average-2
error in use of average-2: integer arguments expected
[buyhorse@server1 shell]$ sh average-2 4 6
sum is 10
average is 5
[buyhorse@server1 shell]$ sh average-2 12345 54321 11111
sum is 77777
average is 25925
[buyhorse@server1 shell]$
```



Example 2



- Write a shell script that reports the number of file names in the current working directory that consist of exactly **six** characters





```
#!/bin/sh
```

```
count=0
```

```
for k in *
```

```
do
```

```
    result= “`echo $k | grep ‘^.....$’ `”
```

```
    if [ -n “$result” ]
```

```
    then
```

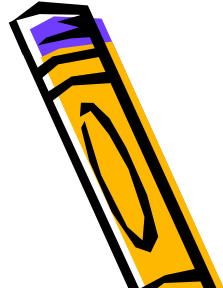
```
        count=`expr $count + 1`
```

```
    fi
```

```
done
```

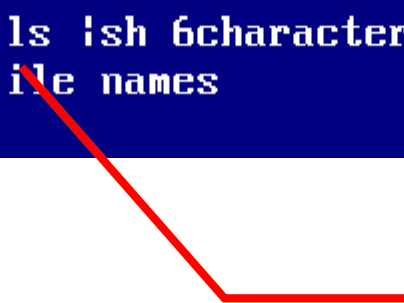
```
echo There were $count 6-character file names
```


6character

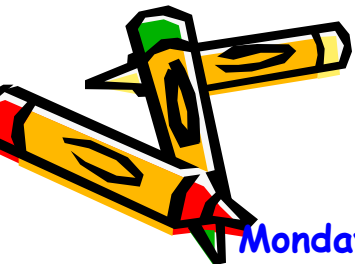


```
[bughorse@server1 shell]$ ls
6character      countup~        for_demo        statis
addall          countup~        grep_hello      student_records
average-1       debug_demo      heredoc_demo    student_records_sorted
average-2       dirtest         if_demo         trap_demo
bundle          display_change_name inter.txt        try.txt
case_1          display_name    marks.txt       until_demo
case_demo       export_demo     read_demo       while_demo
case_demo_revise file1            set_demo
cmdargs_demo    file2           shift_demo


[bughorse@server1 shell]$ ls ish 6character
There were 4 6-character file names
[bughorse@server1 shell]$
```





What exactly?
6character-new



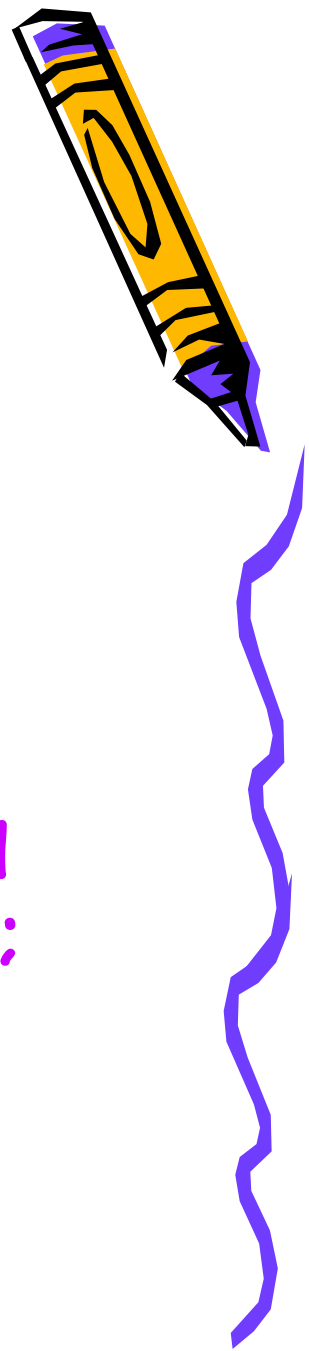
Example 3: statis



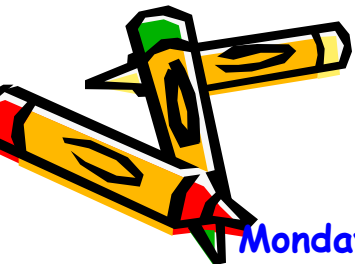
```
buyhorse@ubuntu-server-1804:~/shell$ cat scores.txt
083234673      90      H. tom
334892983      80      A. jerry
834683376      75      J. kitty
733626272      45      K. ross
buyhorse@ubuntu-server-1804:~/shell$ cat -n statis
 1  #/bin/sh
 2  sum=0;countfail=0;count=0;
 3  while read studentnum grade name; do
 4  sum=`expr $sum + $grade`
 5  count=`expr $count + 1`
 6  if [ $grade -lt 50 ]; then
 7  countfail=`expr $countfail + 1`
 8  fi
 9  done
10  echo The average is `expr $sum / $count`.
11  echo $countfail students failed.
```



Example 3 : statis cont'



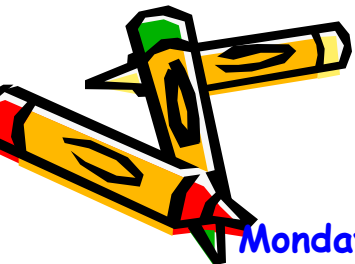
- How could we execute it?
- As usual
 - `$ cat scores.txt | sh statis`
 - `$ sh statis < scores.txt`
- We could also just execute statis and provide marks through standard input; use `^D` to signal end of input.



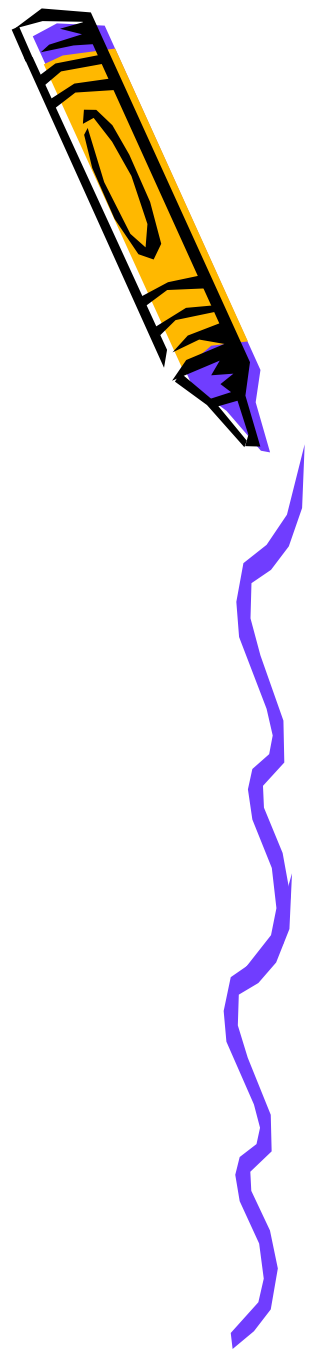
Cmds again: Handy utility commands



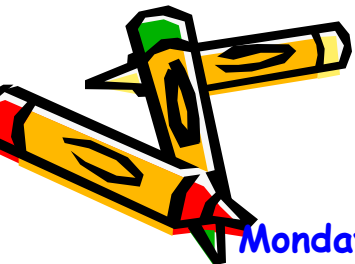
- Show you some clever things to do with Unix.
- You might consider this an "advanced" commands though most of the things here are still simply.



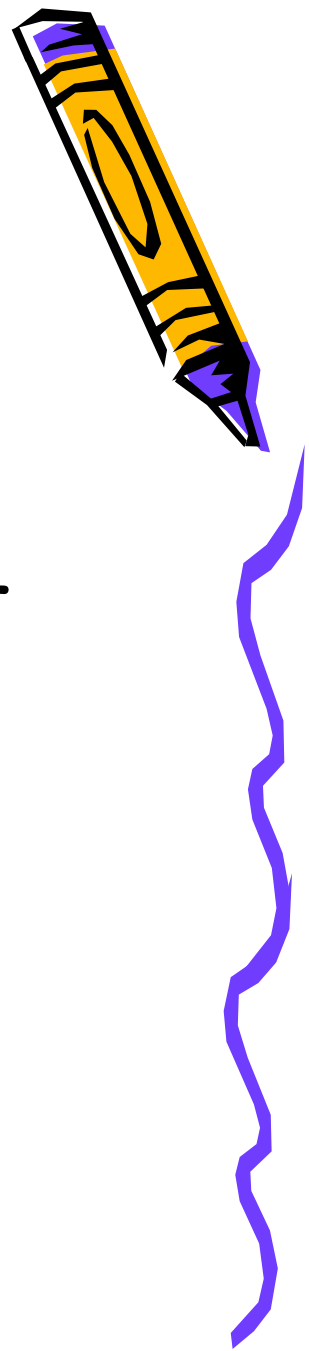
Sending and Reading Email with mail



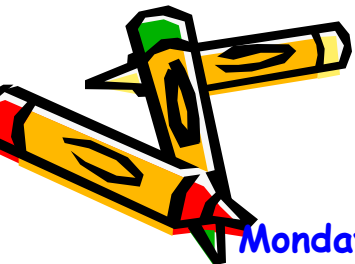
- mail userid <1.txt
- mail -s “subject” userid
- ~r contact
- Ctrl+d



Redirecting to Multiple Locations with **tee**

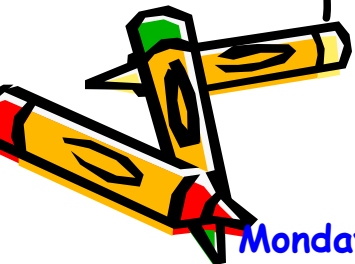
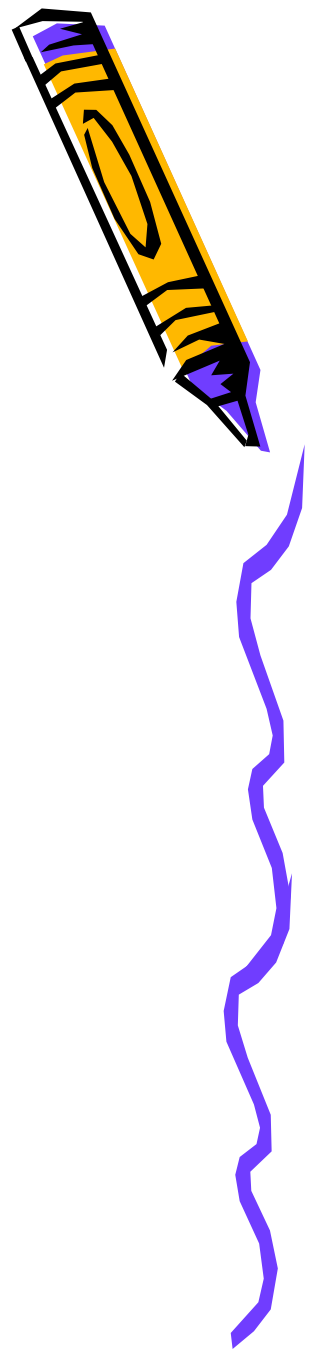


- `$ sort 1.txt 2.txt |tee 1-2.sorted
|mail buyhorse -s "here's the 1.txt
and 2.txt"`



search for files in a directory hierarchy with **find**

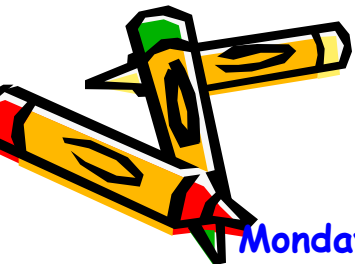
- `find ~ -name 1.txt -print`
- `find ./ -name "*.txt"`
- `find ./ -mtime -3`
- `find /var/log/ -mtime -1`
- `find ./ -size 0` //仅普通文件
- `find ./ -empty` //还包含空目录
- `find ./ -type d`
- `find ./ newer 1.txt`



search con't



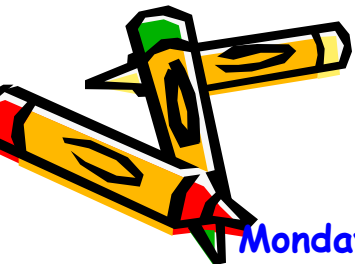
- whereis: 简单快速 whereis passwd
- locate: 快而全 locate passwd
- which: 小而精 which passwd
- find: 精而细



Changing with **tr**



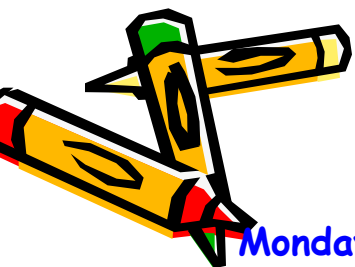
- `$: cat 1.txt | tr a-f A-F`
- `$: cat 1.txt | tr a-zA-Z A-Za-z`
- `$: cat 1.txt | tr " " "\n"`
- `$: tr a-zA-Z A-Za-z < 1.txt`
- `$: cat 1.txt | tr -c a-zA-Z "\n"`
- `$ cat 1.txt | tr "[:lower:]" "[:upper:]"`
- `$ cat 1.txt | sed 'y/abcd/ABCD/'`



grep结合正则，扩展

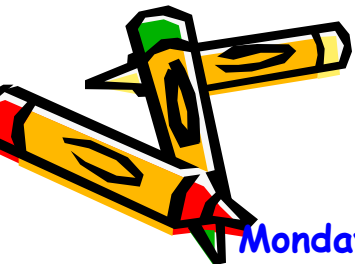
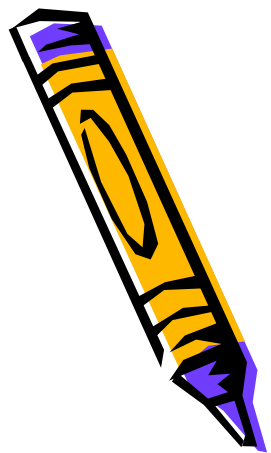


- `grep '^j' student_records`
- `grep '3\..' student_records`
- `egrep 'ECE|CS' student_records` //包含ECE或者CS的行
- `egrep '3+' student_records` //一个或多个3的行
- `ls -l |grep '^d' ; bash dirlist ; find ./ -type d ;`
`ls -d */ vs. ls */`



tar

- `tar -cf txt.tar *.txt`
- **`tar -cvf txt.tar *.txt`**
- `touch testtar.txt`
- `tar -rf txt.tar testtar.txt` //append files to the end
- `tar -tf txt.tar` //list the contents of an archive
- **`tar -czf txt.tar.gz *.txt`**
- `cp txt.tar.gz doc/`
- `tar -xzf txt.tar.gz`



Encoded & Compressed

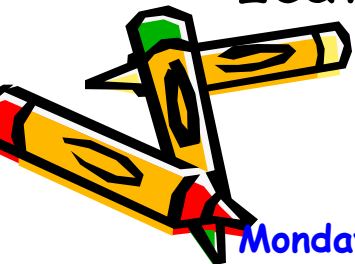


- `compress txt.tar`
- `uncompress txt.tar.Z`

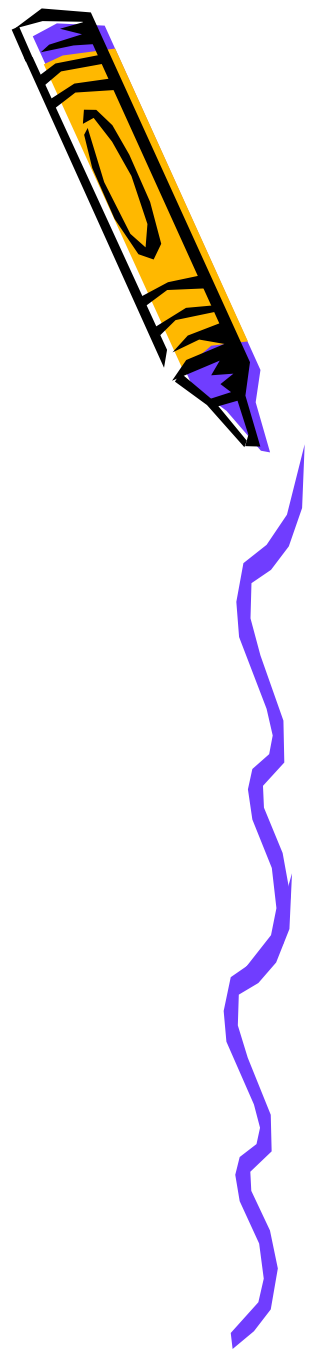
- `gzip txt.tar`
- `gzip -d txt.tar.gz = gunzip txt.tar.gz`

- `bzip2 txt.tar`
- `bzip2 -d txt.tar.bz2`

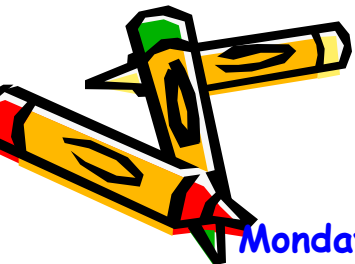
- `zip txt.tar.z txt.tar`
- `unzip txt.tar.z`
- `zcat txt.tar.z`



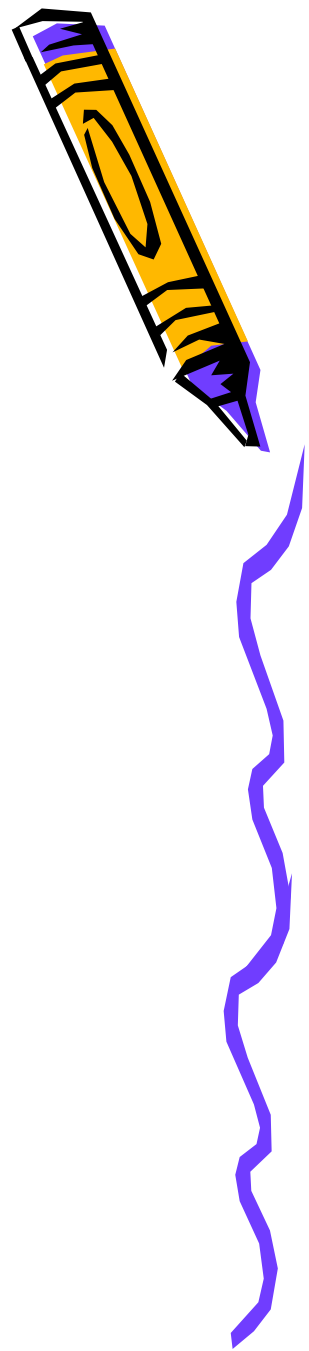
Setting Aliases with alias



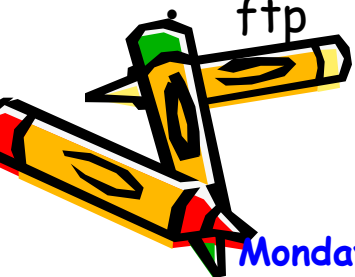
- `$ alias`
- `$ grep alias ~/.bash* ~/.profile /etc/bashrc`
- `$ vi .bashrc`
- `$ alias quit="logout"`
- `$ alias dir="ls -l"`
- `$ alias rm="rm -i"`
- `$ alias`
- `$ rm 1.txt`
- `$ unalias rm`
- `$ rm 1.txt`
- `$ quit`



Accessing the Internet

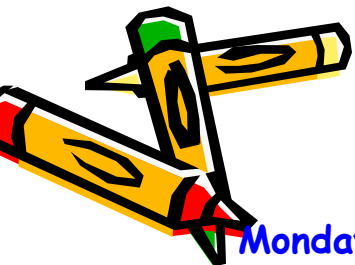


- write
- talk
- **mesg**
- telnet
- ssh
- links `www.dlmu.edu.cn`
- lynx `www.dlmu.edu.cn`
- w3m `www.dlmu.edu.cn`
- wget `www.dlmu.edu.cn`
- links `index.html`
- wget `--recursive --level=2 english.dlmu.edu.cn`
- wget `-r` (download the whole thing)
- wget `ftp://ftp.example.com`
- ftp



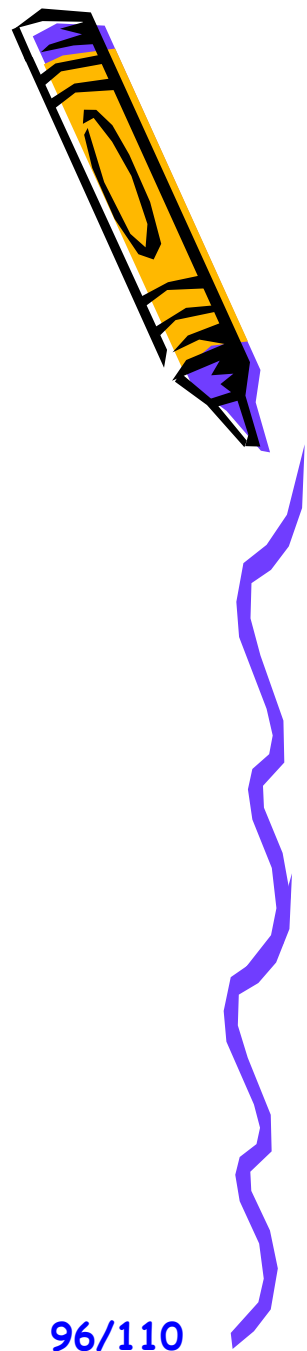
Tracing and checking Connections

- `ping` //send ICMP ECHO_REQUEST to network hosts
- `tracert www.dlmu.edu.cn` //traces path to a network host
- `tracert -d www.dlmu.edu.cn` //traces path to a network host discovering MTU along this path
- `ip route/addr/-s link/neighbor` // show or manipulate routing, devices, policy routing and tunnels
- `nslookup www.dlmu.edu.cn` //query Internet name servers interactively
- `host www.dlmu.edu.cn`
- `mtr www.dlmu.edu.cn` //a network diagnostic tool
- `ifconfig` //configure a network interface
- `ifup/ifdown` // bring a network interface up/down
- `dig www.dlmu.edu.cn` //DNS lookup utility
- `netstat -s`



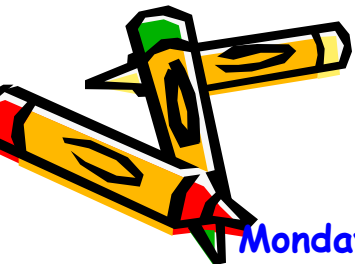
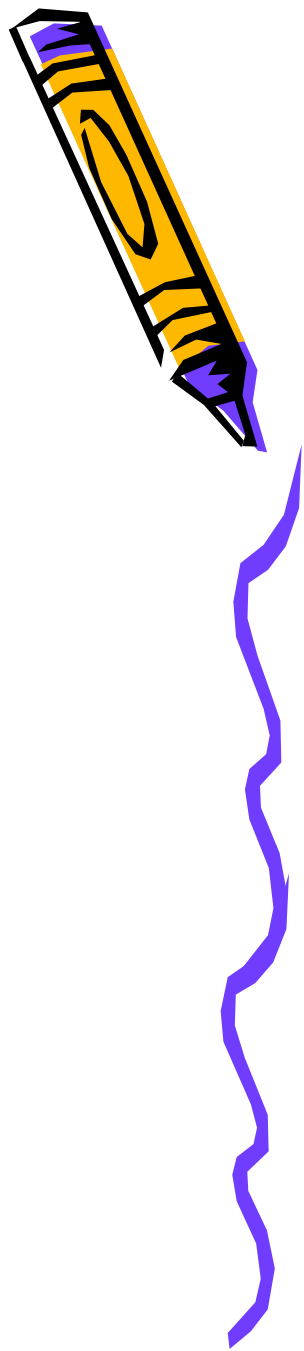
Calendaring and time

- cal
- ncal
- cal -3
- cal -j
- cal year
- ncal |grep
- calendar -B 2
- time ls



bc :Base Conversion

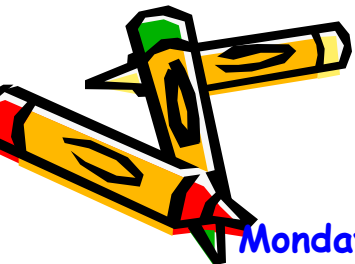
- bc
- + - * / % ...
- ctrl+d (quit the bc)
- bc bc.txt



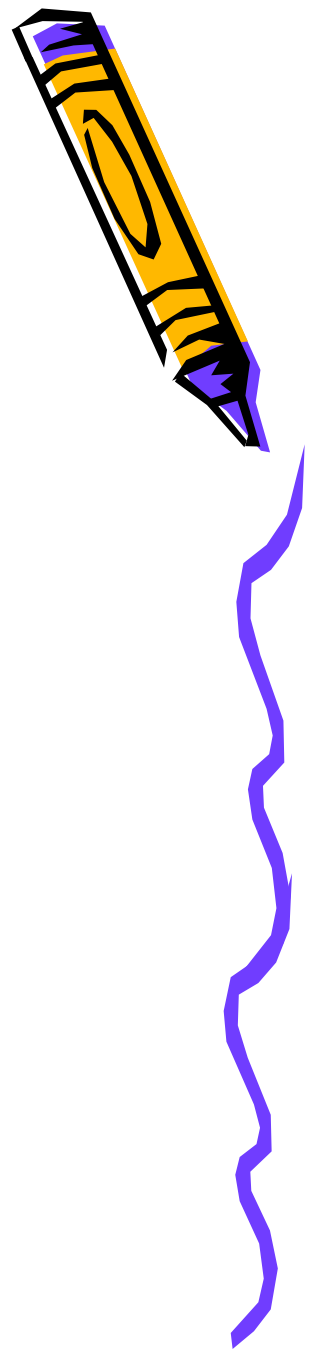
factor: Factorizing Numbers



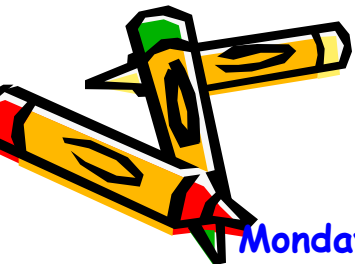
```
buyhorse@ubuntu-server-1804:~/shell$ factor 15
15: 3 5
buyhorse@ubuntu-server-1804:~/shell$ factor
15
15: 3 5
36
36: 2 2 3 3
```



units: Scale Conversion

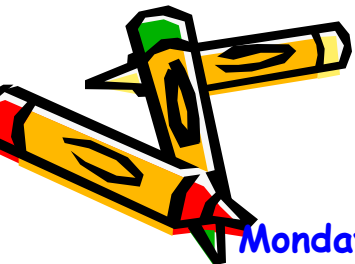
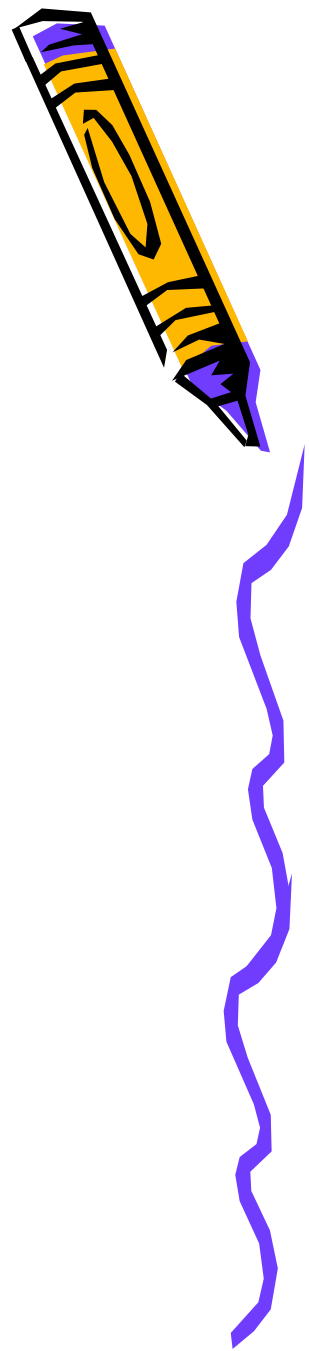


- units
 - You have: inch
 - You want: feet
 - You have: mph
 - You want: sec/mile
 - ctrl+d
- units <units.txt
- /usr/share/units/definitions.units

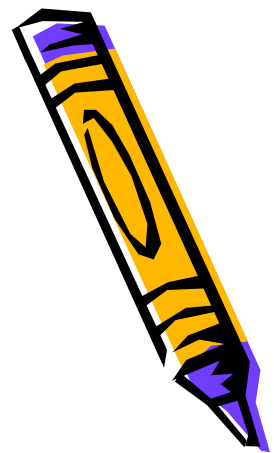


Checking Spelling

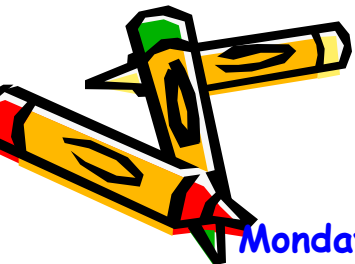
- `spell 1.txt`
- `ispell 1.txt`



look

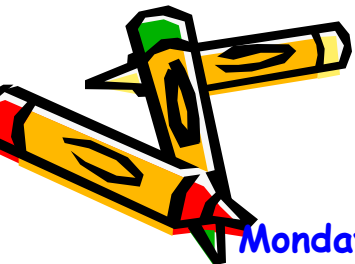
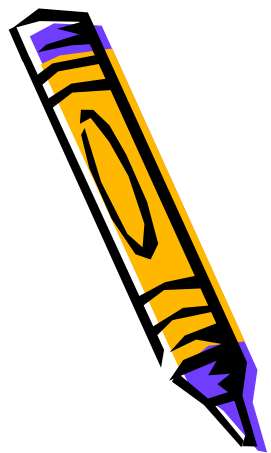


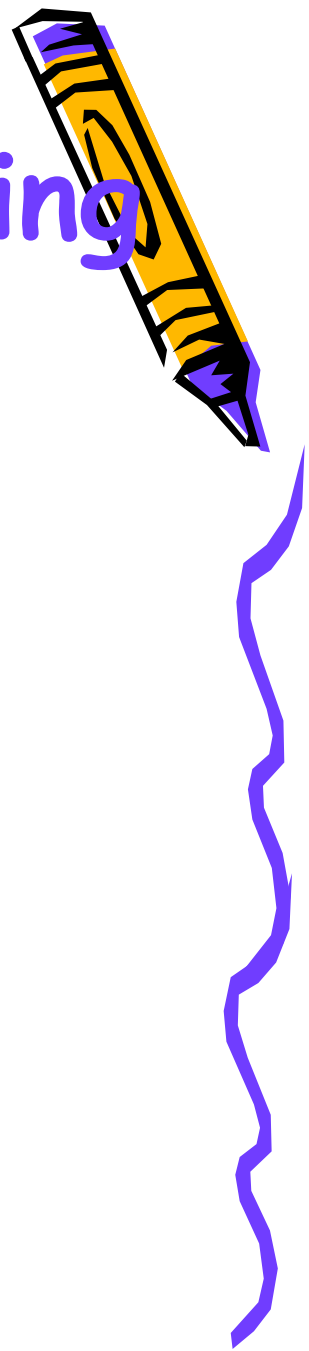
- //display lines beginning with a given string
- `$ look h 1.txt` //以h开头的
- `$ look -f h 1.txt` //忽略大小写
- `$look unfor`



Formatting with `fmt`

- `$ cat unformatted`
- `$ fmt unformatted`
- `fmt -u` to make spacing uniform: one space between words, and two spaces between sentences
- `fmt -w` to specify the width of the formatted text; for example, `w 60` would specify a 60-character-wide line
- `fmt -u -w 30 unformatted`

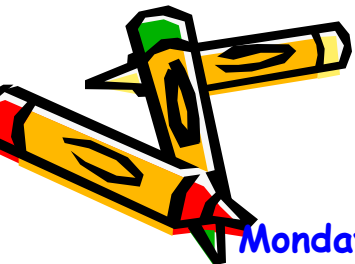




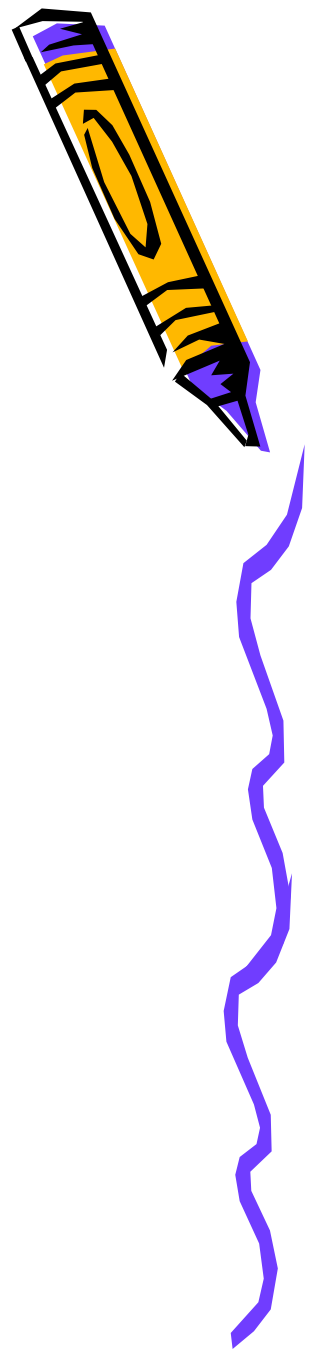
convert text files for printing

pr

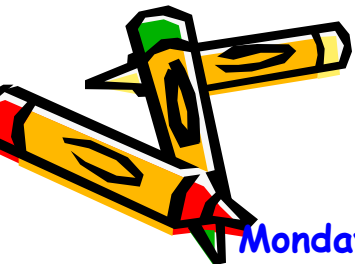
- \$ pr unformatted
- \$ cat md.c |pr -2



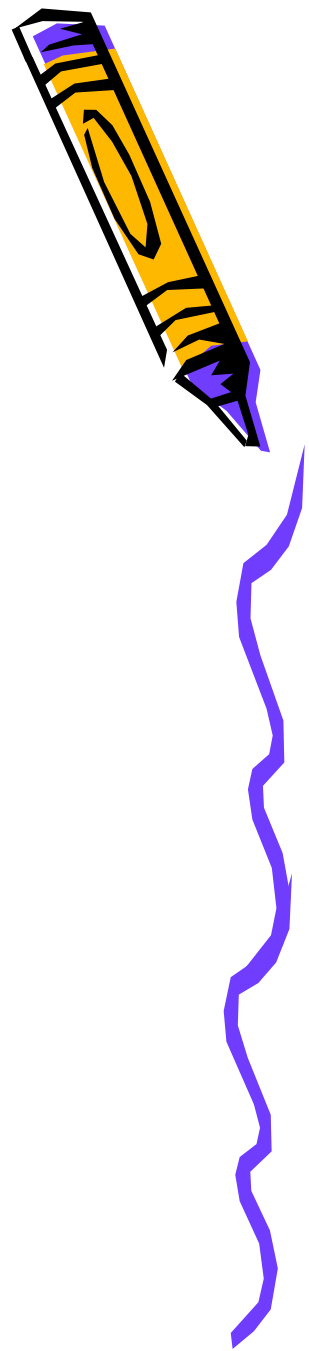
print files with lp



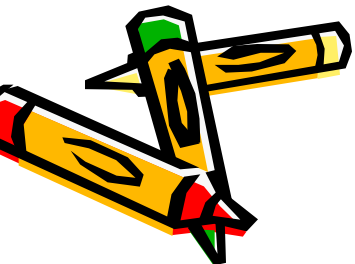
- lp filename
- lp -d otherprintername filename
- fmt unformatted | pr | lp filename
- lpq



cut: remove sections from each line of files



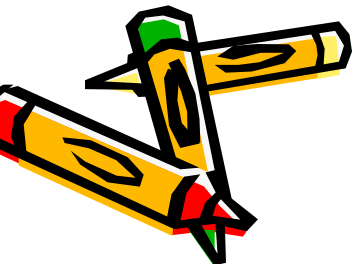
- 参数
 - -c : 以字符为单位进行分割。
 - -d : 自定义分隔符，默认为制表符。
 - -f : 与-d一起使用，指定显示哪个区域。
- `cut -c -7 filename`
- `cut -f 1 student_records`
- `cut -f 2,4 student_records`
- `sed -n l student_records`
- `cut -d: -f1,7 /etc/passwd`
- `cut -d';' -f1,3 com.txt`
- `paste p1.txt p2.txt`
- `paste -s p1.txt p2.txt`



awk - pattern scanning and processing language

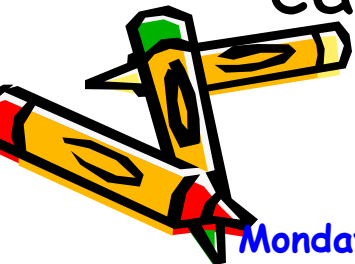
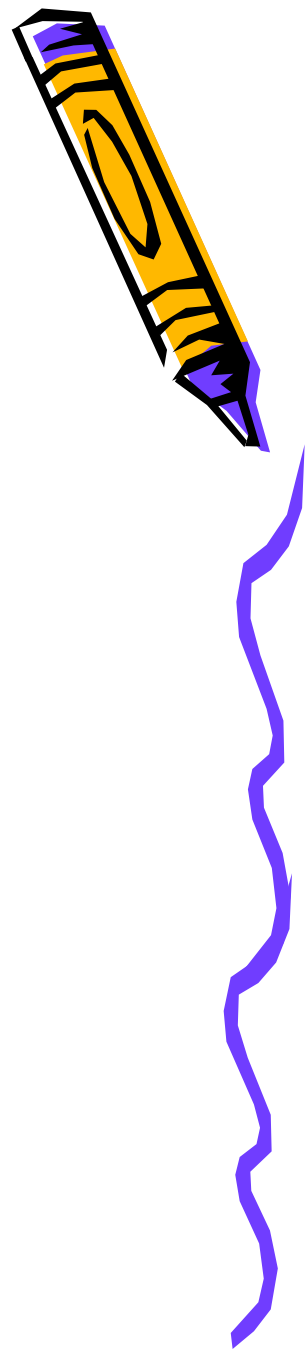


- `awk '/hello/' 1.txt`
- `awk '$4>3' student_records`
- `awk '$4>3 {print NR}' student_records`
- `awk '$4>3 {print $1}' student_records`
- `awk '/hello/ {count++;} END {print "hello was found "count" times"}' 1.txt`
- `awk 'BEGIN {total=0} {total+=$4} END {print total}' student_records`



Keeping a Record of Your Session with **script**

- script coverrecording
- ls
- pwd
- date
- exit (ctrl+d)
- ls -l coverrecording
- cat coverrecording



Monitoring and checking the messages



- top (h for help)
- top d 2 | grep Mem
- w;who;finger;last;watch last; watch -n 5 date
- dmesg
- dmesg | mail user -s "Help me understand"
- tail /var/log/kern.log
- **free** (used – buffers – cached vs. free + buffers + cached)
- vmstat
- lspci

`cat /etc/rsyslog.d/50-default.conf |grep log`

A buffer is something that has yet to be "written" to disk.

A cache is something that has been "read" from the disk and stored for later use.

`cat /proc/meminfo`



Monitoring and checking the messages -2

144115224423105799正在观看视频



- **vmstat**
- **free (used – buffers – cached vs. free + buffers + cached)**

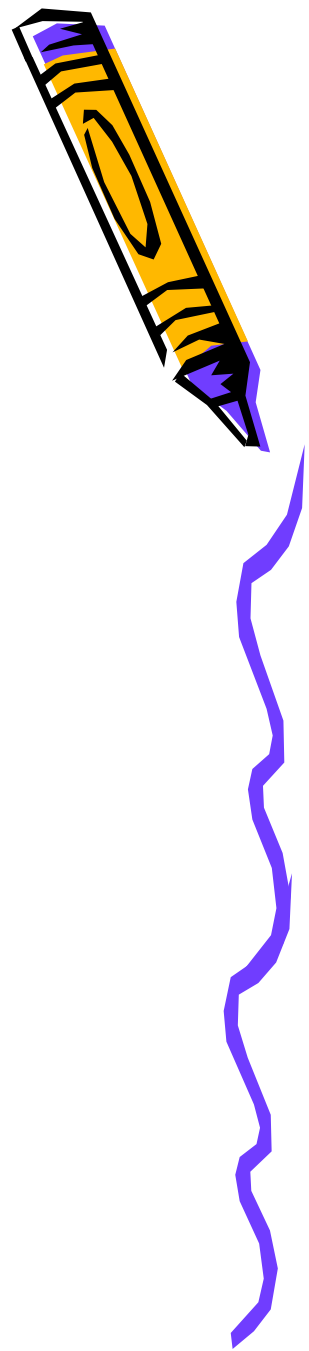
A buffer is something that has yet to be “written” to disk.

A cache is something that has been “read” from the disk and stored for later use.

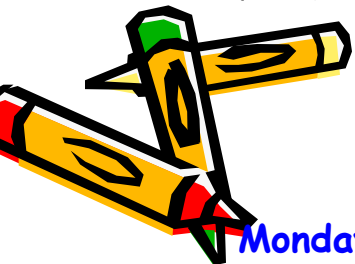
cat /proc/meminfo

```
buyhorse@ubuntu:~$ free
              total        used        free      shared    buffers     cached
Mem:      12292140    4502920    7789220    144096     520588    3107324
-/+ buffers/cache:    875008    11417132
Swap:      11718652          0    11718652
buyhorse@ubuntu:~$
```

just funny



- `sl/LS`
- `sl-h`
- `cmatrix`
- `yes Linux is funny`
- `sudo apt-get moo`
- `telnet towel.blinkenlights.nl`
- `/usr/games/`



End of shell

