LING 408/508: Programming for Linguists

Lecture 6

Today's Topics

- Shell arithmetic
- bc command
- comparison in the shell
- positional parameters for shell scripts
- making shell scripts directly executable (chmod)
- Homework 3
  - due next Monday midnight

Last Time: `cat` command

- See http://www.linfo.org/cat.html

1.	cat *file1*		(print contents of file1)
2.	cat *file1* > *file2*		(‘>’ = redirect output to file2)
3.	cat *file2* | more		(‘|’ = pipe output to command more)
4.	more *file1*	– *easier*	(stops at end of screen, hit space to show more)
5.	less *file1*	– *easier*	(allows page by page display)
6.	cat > *file1*		(create file1 with input from terminal until **Control-D** EOF)
7.	cat		(input from terminal goes to terminal)
8.	cat >> *file1*		(append input from terminal to file file1)
9.	cat *file1* > *file2*		(file copy)
10.	cp *file1 file2* – *easier*		(cp = copy)
11.	cat file1 file2 file3		(prints all 3 files)
12.	cat *file1 file2 file3* > *file4*		(prints all 3 files to file4)
13.	cat *file1 file2 file3* | sort > *file4*		(3 files sorted alphabetically to file4)
14.	cat – *file5* > *file6*		(‘-’ = input from terminal)
15.	cat *file7* - > *file8*		

Shell Arithmetic

- at the shell prompt:
  - expr 1 + 3 (Need spaces cf. expr 1+3)
  - expr 2 '*' 2 (cf. expr 2 * 2)
  - echo `expr 1 + 3`
  - i=2 (NO SPACES! cf. i = 2)
  - expr $i + 1

  - let x=1+3 (cf. let x=1 + 3)
  - echo $x
  - let i=$i+1 (also ok let i=i+1)
  - echo $i

  - ((x = 1+ 3)) (spaces not significant)
  - echo $x
  - echo $((1+3))
  - ((i=i+1)) (also ok let i=$i+1)

Shell Arithmetic: use command `bc` instead

```
bc(1)                                                          bc(1)

NAME
       bc - An arbitrary precision calculator language

SYNTAX
       bc [ -hlwsqv ] [long-options] [  file ... ]

VERSION
       This man page documents GNU bc version 1.06.

DESCRIPTION
       bc  is a language that supports arbitrary precision numbers with inter-
       active execution of statements.  There are  some  similarities  in  the
       syntax  to  the  C  programming  language.   A standard math library is
       available by command line option.  If requested, the  math  library  is
```

`man bc` command brings up this page

- `bc` runs interactively
- `bc -l` loads the math library first

command `bc`

- Examples:
  - we know $\tan(\pi/4) = 1$, so $\tan^{-1}(1) = \pi/4$ ($\pi/4$ in radians = 45°)
  - function `a(`*radians*`)` computes arctan when `bc -l` is used

```
Machine$ bc -l
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
a(1)
.78539816339744830961
a(1)*4
3.14159265358979323844
^DMachine$
```

command `bc`

- Examples:
  - we know tan(π/4) = 1, so $\tan^{-1}(1) = \pi/4$ (π/4 in radians = 45°)
  - function `a(`*radians*`)` computes arctan when `bc -l` is used
- In man bc:
  - *the following will assign the value of "pi" to the shell variable* ***pi****.*
  - `pi=$(echo "scale=10; 4*a(1)" | bc -l)`

command bc

https://www.gnu.org/software/bash/manual/html_node/Command-Substitution.html#Command-Substitution

3.5.4 Command Substitution

Command substitution allows the output of a command to replace the command itself.
enclosed as follows:

`$(command)`

```
pi=$(echo "scale=10; 4*a(1)" | bc -l)
```

or

`` `command` ``

```
Machine$ pi=$(echo "scale=10; 4*a(1)" | bc -l)
Machine$ echo $pi
3.1415926532
Machine$
```

```
pi=`echo "scale=10; 4*a(1)" | bc -l`
```

command bc

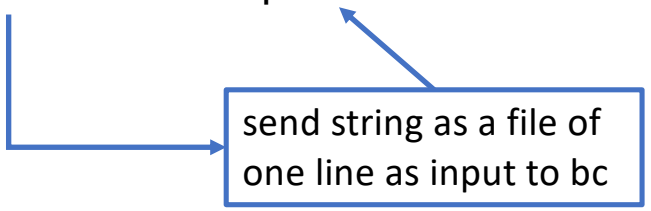
- https://www.gnu.org/software/bash/manual/html_node/Bash-Builtins.html#index-echo

echo

```
echo [-neE] [arg …]
```

Output the *arg*s, separated by spaces, terminated with a newline.

```
pi=$(echo "scale=10; 4*a(1)" | bc -l)
```



command bc

```
[Machine$ pi=$(echo "scale=10; 4*a(1)" | bc -l)
[Machine$ echo $pi
 3.1415926532
```

- pi is a bash shell variable here

```
[Machine$ echo "scale=10; 4*a(1)" | bc -l > pi.txt
[Machine$ more pi.txt
3.1415926532
pi.txt (END)
```

- spacebar to get out of `more`

command bc

- scale

```
There are four special variables, scale, ibase, obase, and last.  scale
defines  how  some  operations use digits after the decimal point.  The
default value of scale is 0. ibase and obase define the conversion base
for input and output numbers.  The default for both input and output is
base 10.  last (an extension) is a variable that has the value  of  the
last  printed  number.  These will be discussed in further detail where
```

command bc

- scale

```
[^DMachine$ bc -l
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
[scale = 100
[a(1)*4
3.1415926535897932384626433832795028841971693993751058209749445923078\
16406286208998628034825342117067982148086513282306647093844609550582
[scale = 1000
[a(1)*4
3.1415926535897932384626433832795028841971693993751058209749445923078\
16406286208998628034825342117067982148086513282306647093844609550582\
23172535940812848111745028410270193852110555964462294895493038196442\
88109756659334461284756482337867831652712019091456485669234603486104\
54326648213393607260249141273724587006606315588174881520920962829254\
09171536436789259036001133053054882046652138414695194151160943305727\
03657595919530921861173819326117931051185480744623799627495673518857\
52724891227938183011949129833673362440656643086021394946395224737190\
70217986094370277053921717629317675238467481846766940513200056812714\
52635608277857713427577896091736371787214684409012249534301465495853\
71050792279689258923542019956112129021960864034418159813629774771309\
96051870721134999999837297804995105973173281609631859502445946534690\
83026425223082533446850352619311881710100031378387528865875332083814\
20617177669147303598253490428755468731159562863882353787593751957781\
85778053217122680661300192787661119590921642019881
```

command bc

```
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
obase=2
7
111
254
11111110
obase=16
255
FF
15
F
13
D
```

- obase=2 (binary)
- obase=16 (hexadecimal) 0..9,A..F

test

- https://www.gnu.org/software/bash/manual/html_node/Bourne-Shell-Builtins.html#index-test

```
test expr
```

Evaluate a conditional expression *expr* and return a status of 0 (true) or 1 (false). Each operator and operand must be a separate argument. Expressions are composed of the primaries described below in Bash Conditional Expressions. `test` does not accept any options, nor does it accept and ignore an argument of -- as signifying the end of options.

`! expr`

True if *expr* is false.

`( expr )`

Returns the value of *expr*. This may be used to override the normal precedence of operators.

`expr1 -a expr2`

True if both *expr1* and *expr2* are true.

`expr1 -o expr2`

True if either *expr1* or *expr2* is true.

Comparison operators

- Format:

```
if [ $x OP $y ]; then
…
(else/elif…)
fi
```

  - spacing is crucial
  - [….] is known as *test*
  - OP:
    - -eq *equals*
    - -ne *not equals*
    - -gt *greater than*
    - -ge *greater than or equals*
    - -lt *less than*
    - -le *less than or equals*

    … and more

- Shell variable:
  - $? (*exit status from previous command*)
  - value 0 (success; true)

```
Machine$ test 4 -lt 5; echo $?
0
Machine$ test 4 -gt 5; echo $?
1
```

```
Machine$ i=2
Machine$ x=4
Machine$ test $x -gt $i -a $i -lt $x; echo $?
0
Machine$ test $x -gt $i -a $i -gt $x; echo $?
1
```

https://www.gnu.org/software/bash/manual/html_node/Bash-Conditional-Expressions.html#Bash-Conditional-Expressions

Comparison operators

```
[Machine$ if [ 3 -lt 4 ]; then echo "Yes"; fi
 Yes
[Machine$ if [3 -lt 4]; then echo "Yes"; fi
 -bash: [3: command not found
[Machine$ if [ 3 -lt 4 ]; then echo "Yes" fi
[> ;
 -bash: syntax error near unexpected token `;'
 Machine$
```

- spacing is crucial
- also semicolons terminating commands on the same line

Positional Parameters

- In a shell script, these variables have values:
  - $1: first parameter
  - $2: 2nd parameter and so on…
  - $#: # of parameters

- Program:

```
#!/bin/bash
echo "Number of parameters: $#"
if [ $# -eq 1 ]; then
    echo "1st parameter: $1"
fi
```

- Output:
  - `sh test.sh`

    Number of parameters: 0
  - `sh test.sh 45`

    Number of parameters: 1

    1st parameter: 45
  - `sh test.sh 45 56`

    Number of parameters: 2

Running shell scripts

- Supply program filename as a parameter to sh/bash:
- `sh` is `dash`, not `bash` anymore
  - `sh test.sh`
  - `bash test.sh`
  - `source test.sh`
  - `. test.sh`
  - (*. = source*)

- Run the program in the current directory:
  (./ needed if current directory is not in PATH)
  - `./test.sh`

    -bash: ./test.sh: Permission denied
  - `ls -l test.sh`

    -rw-r--r-- 1 sandiway staff 98 Sep 4 09:14 test.sh
  - `chmod u+x test.sh`
  - `ls -l test.sh`

    -rwxr--r-- 1 sandiway staff 98 Sep 4 09:14 test.sh
  - `./test.sh`

    Number of parameters: 0

Running shell scripts

Chmod 644

Chmod 644 (*chmod a+rwx,u-x,g-wx,o-wx*) sets permissions so that, (U)ser / owner can read, can write and can't execute. (G)roup can read, can't write and can't execute. (O)thers can read, can't write and can't execute.

	Owner Rights (u)		Group Rights (g)		Others Rights (o)	
Read (4)	☑	1	☑	1	☑	1
Write (2)	☑	1	☐	0	☐	0
Execute (1)	☐	0	☐	0	☐	0

Recall everything is binary:

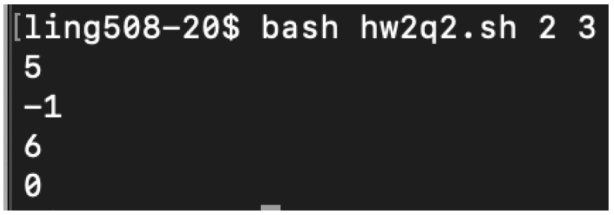
- 110 = 6, 100 = 4
- 644 = 110100100 (3 groups of binary)

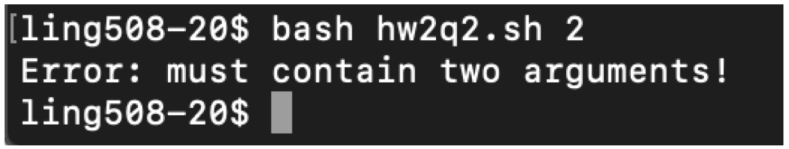
Homework 3

- submit one PDF file covering all questions
- Subject: 408/508 Homework 3 YOUR NAME
- Question 1
  - use bc to compute the value of the math constant *e* to 50 decimal places
  - https://en.wikipedia.org/wiki/E_(mathematical_constant)
  - submit screenshot
  - compare your answer to that shown
  - **BONUS CREDIT**: what's weird about the result?

Homework 3

- Question 2:
  - write a bash shell script that takes two command line parameters (two numbers), calls bc to print out the result of adding, subtracting, multiplying and dividing the two numbers. It should print an error instead if you don't submit two numbers.
  - Submit program code and screenshot
  - **Example**:





  - Notice last result above? Change your program to allow floating point results