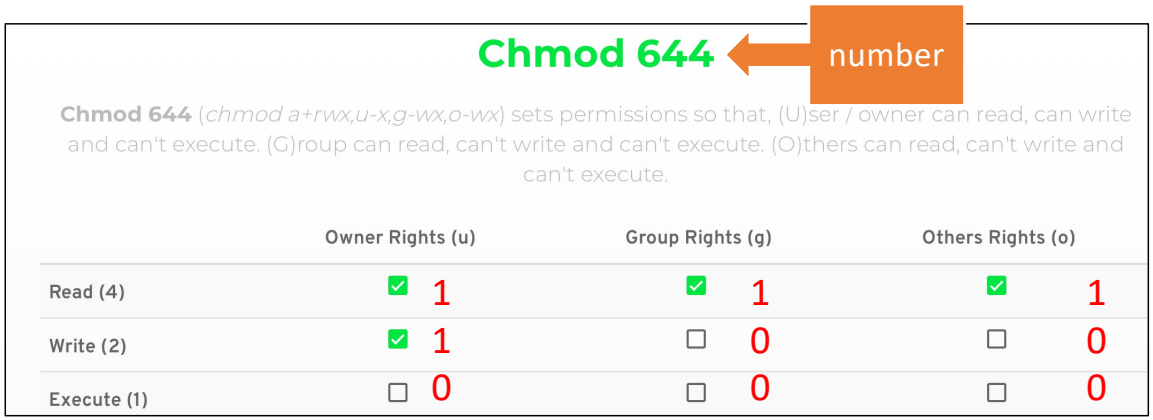
Lecture 8

408/508 *Computational Techniques for Linguists*

Today's Topics

1. a note on file permissions
2. bc (command)
3. a note on positional parameters
4. Homework 4
   - *a shell script program for you to write*

Running shell scripts



Command:

- `chmod` *`permissions filename`*
- *`permissions`*: e.g. u+x (*user add execute*) or a number

Recall everything is binary:

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

Shell Arithmetic: use command `bc` instead

- shell arithmetic, e.g. `((z= x+y))`, is integer only.
- What if you needed floating point numbers?

```
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(π/4) = 1, so $\tan^{-1}(1) = \pi/4$ (π/4 in radians = 45°)
  - function `a(`*radians*`)` computes arctan when `bc -l` is used
  - Control-D (EOF) to exit `bc`

```
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$
```

  - using echo and pipe into bc

```
(base) ~$ echo "4*5" | bc
20
```

command bc

- Example:
  - 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
- From man bc:
  - *the following* [*Terminal command*] *will assign the value of "pi" to the shell variable* ***pi****.*

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

$(*command*) is a modern synonym for `*command*` (` is backtick, not ') which stands for command substitution; it means run *command* and put its output here (*see next slide*).

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` ``

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

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

command bc

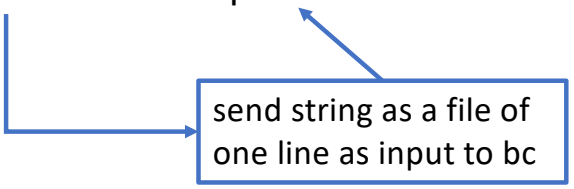
- https://www.gnu.org/software/bash/manual/html_node/Bash-Builtins.html#index-echo

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`

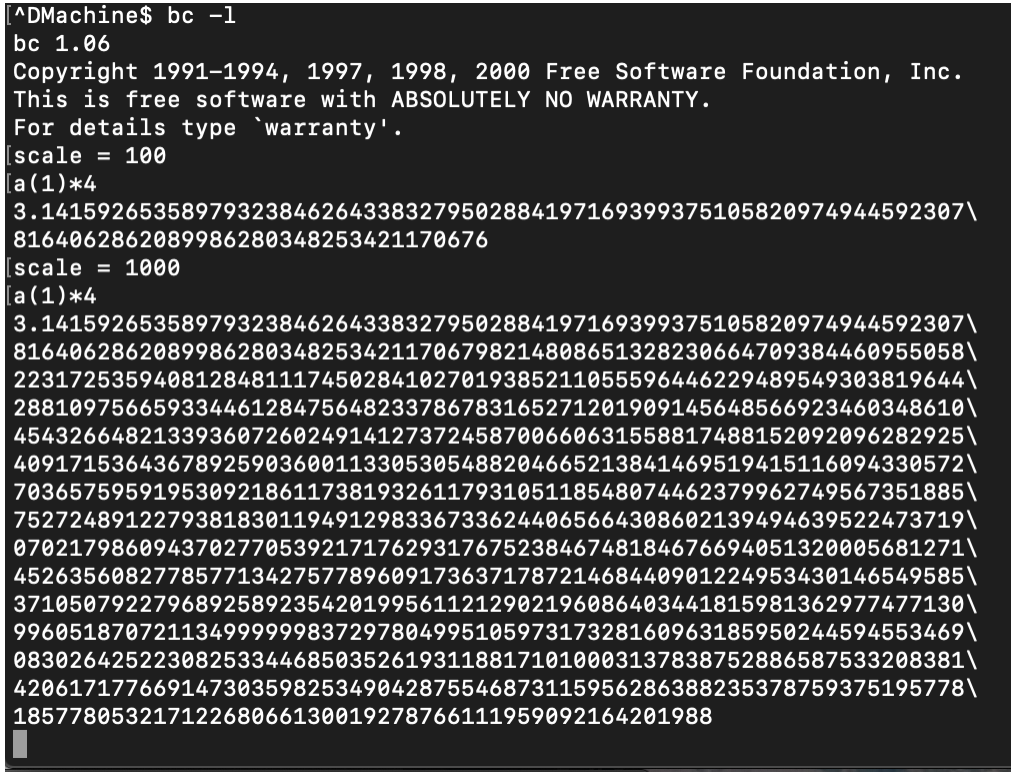
command bc

- scale variable in bc:

```
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



Math constant *e*

```
scale=50
e(1)
2.71828182845904523536028747135266249775724709369995
scale=51
e(1)
2.718281828459045235360287471352662497757247093699959
scale=52
e(1)
2.7182818284590452353602874713526624977572470936999595
```

100 Decimal Digits

https://www.mathsisfun.com/numbers/e-eulers-number.html

Here is **e** to 100 decimal digits:

2.71828182845904523536028747135266249775724709369995957
49669676277240766303535475945713821785251664274...

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
```

Output base (obase) variable:

- obase=2 (*binary*)
- obase=16 (*hexadecimal*) 0..9,A..F
- **Recall**: control-D to exit

Positional Parameters

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

- Program on webpage `test.sh`:

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

- Output:
  - `bash test.sh`

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

    Number of parameters: 1

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

    Number of parameters: 2

or do `chmod u+x test.sh`
Run using: `./test.sh`

If-test

test2.sh

```
1#!/bin/bash¶
2echo "Number of parameters: $#"¶
3if [$# -eq 1]; then¶
4    echo "1st parameter: $1"¶
5fi¶
```

```
$ bash test2.sh 1 2
Number of parameters: 2
test2.sh: line 3: [2: command not found
$
```

- Spaces are important!

If-test

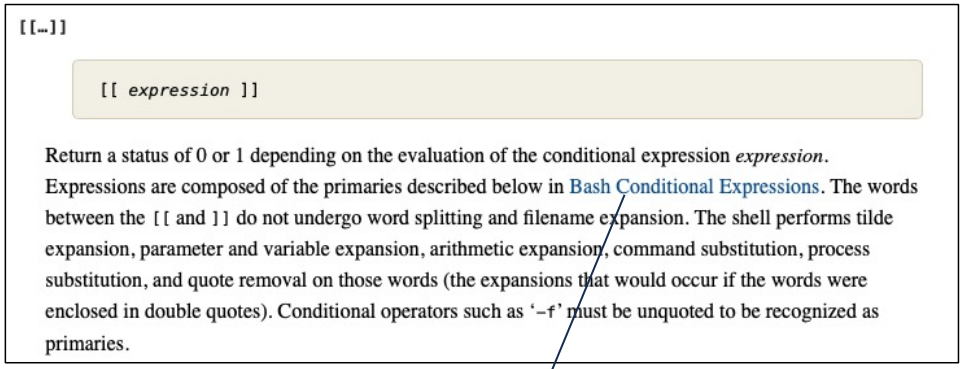
- Note the spaces between the […] !

Primary	Meaning
[-a FILE]	True if FILE exists.
[-b FILE]	True if FILE exists and is a block-special file.
[-c FILE]	True if FILE exists and is a character-special file.
[-d FILE]	True if FILE exists and is a directory.
[-e FILE]	True if FILE exists.
[-f FILE]	True if FILE exists and is a regular file.
[-g FILE]	True if FILE exists and its SGID bit is set.
[-h FILE]	True if FILE exists and is a symbolic link.
[-k FILE]	True if FILE exists and its sticky bit is set.
[-p FILE]	True if FILE exists and is a named pipe (FIFO).
[-r FILE]	True if FILE exists and is readable.
[-s FILE]	True if FILE exists and has a size greater than zero.
[-t FD]	True if file descriptor FD is open and refers to a terminal.
[-u FILE]	True if FILE exists and its SUID (set user ID) bit is set.
[-w FILE]	True if FILE exists and is writable.
[-x FILE]	True if FILE exists and is executable.
[-O FILE]	True if FILE exists and is owned by the effective user ID.
[-G FILE]	True if FILE exists and is owned by the effective group ID.
[-L FILE]	True if FILE exists and is a symbolic link.
[-N FILE]	True if FILE exists and has been modified since it was last read.

[FILE1 -nt FILE2]	True if FILE1 has been changed more recently than FILE2, or if FILE1 exists and FILE2 doe
[FILE1 -ot FILE2]	True if FILE1 is older than FILE2, or is FILE2 exists and FILE1 does not.
[FILE1 -ef FILE2]	True if FILE1 and FILE2 refer to the same device and inode numbers.
[-o OPTIONNAME]	True if shell option "OPTIONNAME" is enabled.
[-z STRING]	True of the length if "STRING" is zero.
[-n STRING] or [STRING]	True if the length of "STRING" is non-zero.
[STRING1 == STRING2]	True if the strings are equal. "=" may be used instead of "==" for strict POSIX compliance.
[STRING1 != STRING2]	True if the strings are not equal.
[STRING1 < STRING2]	True if "STRING1" sorts before "STRING2" lexicographically in the current locale.
[STRING1 > STRING2]	True if "STRING1" sorts after "STRING2" lexicographically in the current locale.
[ARG1 OP ARG2]	"OP" is one of -eq, -ne, -lt, -le, -gt or -ge. These arithmetic binary operators return true "ARG2", respectively. "ARG1" and "ARG2" are integers.

If-test

- https://www.gnu.org/software/bash/manual/bash.html
- if [[*condition*]] (*newer test*: *older* […] *suppported*)



[[…]]

```
[[ expression ]]
```

Return a status of 0 or 1 depending on the evaluation of the conditional expression *expression*. Expressions are composed of the primaries described below in Bash Conditional Expressions. The words between the `[[` and `]]` do not undergo word splitting and filename expansion. The shell performs tilde expansion, parameter and variable expansion, arithmetic expansion, command substitution, process substitution, and quote removal on those words (the expansions that would occur if the words were enclosed in double quotes). Conditional operators such as '`-f`' must be unquoted to be recognized as primaries.

see previous slide, plus pattern matching, e.g. regex =~
[[$# -eq 1 && $1 -lt 10]] vs. [$# -eq 1] && [$1 -lt 10]

```
#!/bin/bash
echo "Number of parameters: $#"
if [[ $# -eq 1 && $1 -lt 10 ]]; then
    echo "1st parameter: $1"
fi
```

```
$ bash test3.sh 1
Number of parameters: 1
1st parameter: 1
$ bash test3.sh 10
Number of parameters: 1
$
```

Homework 4

- Let's write a simple shell-script BMI calculator
  - solicit input from the terminal (using read) or from the command line ($1 $2)

Measurement Units	Formula and Calculation
Kilograms and meters (or centimeters)	Formula: weight (kg) / [height (m)]^2 With the metric system, the formula for BMI is weight in kilograms divided by height in meters squared. Since height is commonly measured in centimeters, divide height in centimeters by 100 to obtain height in meters. Example: Weight = 68 kg, Height = 165 cm (1.65 m) Calculation: 68 ÷ (1.65)^2 = 24.98
Pounds and inches	Formula: weight (lb) / [height (in)]^2 x 703 Calculate BMI by dividing weight in pounds (lbs) by height in inches (in) squared and multiplying by a conversion factor of 703. Example: Weight = 150 lbs, Height = 5'5" (65") Calculation: [150 ÷ (65)^2] x 703 = 24.96

try the metric one first

Homework 4

- You can use if-test, bc (*from this lecture*), and shell scripting to build your program
- Submit your shell script and screenshots of your runs
- To get you started, let's play on the command line first:
  - *your instructor weights 72kg and is 1.72 meters tall*
  - `((bmi = 72 / (1.72 * 1.72)))`
  - `echo $bmi`
  - *won't work: why*?

Homework 4

- One approach is to scale height in cm instead of meters:
  - `((bmi = 72 / (172 * 172)))`
  - `echo $bmi`
  - *how to scale it if we use cm instead of m?*
  - *100 cm in a meter, multiply by what?*

Homework 4

- Instead of scaling to integer, we could pipe numbers to bc directly:

```
(base) ling508-22$ echo "72 / (1.72 * 1.72)" | bc
24
```

- or use variables:

```
(base) ling508-22$ weight=72
(base) ling508-22$ echo $weight
72
(base) ling508-22$ height=1.72
(base) ling508-22$ echo $height
1.72
(base) ling508-22$ echo "$weight / ($height * $height)" | bc
24
(base) ling508-22$
```

Homework 4

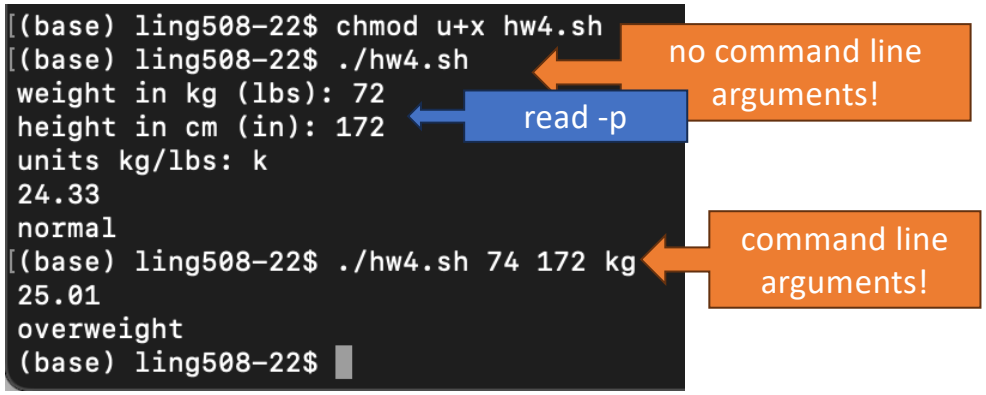
- After you figure out how to do on the command line, put it in a shell script, and try to add the following three embellishments:
  1. accept either command line arguments or read from the terminal if they're missing
  2. recall `read –p "Enter : " variablename`
     - `if [ $# –ne N ]; then`
     - *N* = number of command line arguments.

Homework 4

2. print the weight status message according to the following table:
3. modify the calculator to accept input in both metric and traditional units

- make sure you supply examples of your program working!

BMI	Weight Status
Below 18.5	Underweight
18.5 – 24.9	Normal
25.0 – 29.9	Overweight
30.0 and Above	Obese



Homework 4

- Instructions:
  - email sandiway@arizona.edu
  - submit everything in one PDF file!
  - subject of email: 408/508 Homework 4 *your name*
  - cite any discussion or source
  - due date: next Sunday by midnight