

Lecture 6

*408/508 Computational  
Techniques for Linguists*

# Today's Topics

- cat command
- Shell scripting
  - *writing a program using an editor and running it!*
- Homework 3 (**due Sunday midnight**)
  - *Step-by-step Bash shell exercises*

# 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)
  - *easier* (stops at screen bottom, *space* to show more)
  - *easier* (allows page up/down keys)
4. `more file1`
5. `less file1`
6. `cat > file1` (create file1, input from terminal until **Control-D EOF**)
7. `cat` (weird! 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 in order)
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 program

{1..10..2} means range from 1 to 10 incrementing by 2  
; (semicolon) or newline terminates/separates statements

```
2022 Ubuntu [Running]
Activities Terminal Sep 2 15:41
sandikway@sandikway-VirtualBox: ~
~$ nano ~/.bashrc
~$ echo $SHELL
/bin/bash
~$ for i in {1..10}
> do
> echo "$i time through the loop"
> done
1 time through the loop
2 time through the loop
3 time through the loop
4 time through the loop
5 time through the loop
6 time through the loop
7 time through the loop
8 time through the loop
9 time through the loop
10 time through the loop
~$
```

← echo means print

```
2022 Ubuntu
Activities Terminal
sandikway@sandikway-Vi
~$ for i in {1..10..2}; do echo "$i"; done
1
3
5
7
9
~$
```

```
~$ for ((i=1; i<=10; i=i+2)); do echo "$i by 2"; done
1 by 2
3 by 2
5 by 2
7 by 2
9 by 2
```

macOS

# Input

- At a terminal:
  - `read -p "Name: " name`
  - `read -p "Enter X and Y: " x y`
  - `echo $x`
  - `echo $y`

# Shell script

- Use nano to create a new file named script.sh (convention .sh for script filetype):
  - nano script.sh

- Enter:

```
GNU nano 6.2 script.sh
#!/bin/bash
read -p "First number: " x
read -p "Second number: " y
((z= x+y))
((xy = x*y))
echo "$x + $y = $z, $x * $y = $xy"
```

- Run it!

```
~$ nano script.sh
~$ chmod u+x script.sh
~$ ./script.sh
First number: 5
Second number: 13
5 + 13 = 18, 5 * 13 = 65
~$
```

- ./script.sh means run the script in file script.sh in the current directory (.)
- chmod u+x *filename* means add (+) execute (x) permission for user (u) to *filename*

# Comparison operators

- Format:  
if [ \$x OP \$y ]; then  
...  
(else/elif...)  
fi
- [ .... ] 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*

- Examples:
  - echo \$x \$i  
2 5
  - test \$x -le \$i
  - echo \$? *(exit status)*  
0
  - test \$x -le \$i -a \$i -lt \$x
  - echo \$?  
1

# Shell script 2

Note: typo, this should read  
#!/bin/bash

Note: not shown here  
chmod u+x script2.sh

- using the same approach as in the first shell script ...

```
GNU nano 6.2 script2.sh
#!/bin/bash
read -p "Number 1: " x
read -p "Number 2: " y
if [ $x -lt $y ]; then
    echo "$x smaller than $y"
elif [ $x -eq $y ]; then
    echo "$x = $y"
else
    echo "$x bigger than $y"
fi
```

```
~$ nano script2.sh
~$ ./script2.sh
Number 1: 3
Number 2: 7
3 smaller than 7
~$ ./script2.sh
Number 1: 4
Number 2: 4
4 = 4
~$ ./script2.sh
Number 1: 5
Number 2: 1
5 bigger than 1
~$
```

- Beware of execute permissions!

```
~$ ./script3.sh
bash: ./script3.sh: Permission denied
~$ chmod u+x script3.sh
~$ █
```



# Shell script 2

Compare these two files:

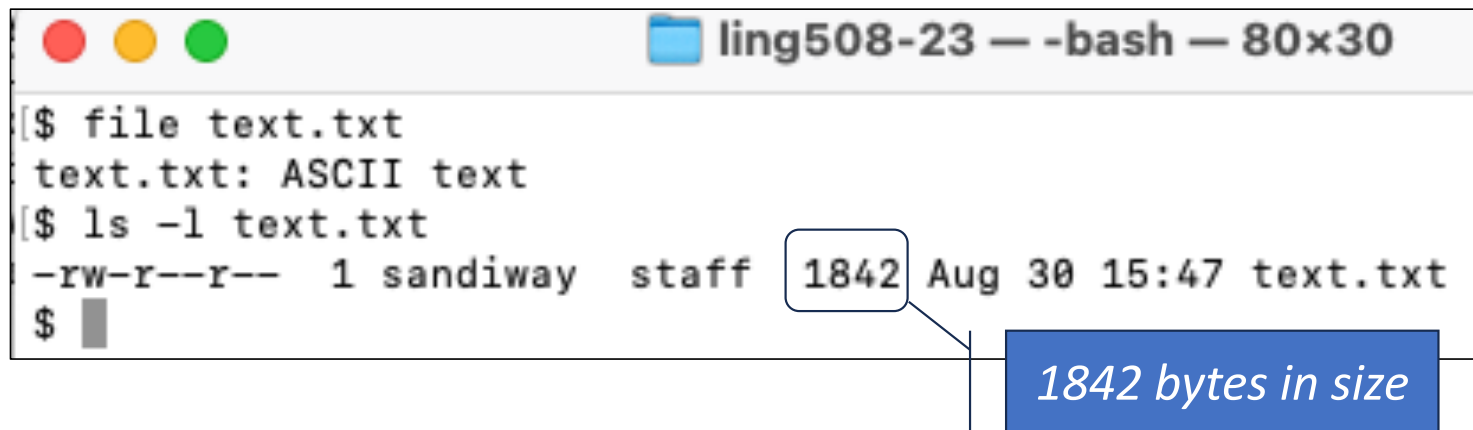
- newlines sometimes matter
- ; separator must be used in some cases
  - if condition; then
- vs.
- if *condition*
- then
- see also ; fi

```
GNU nano 6.2 script3.sh
#!/bin/bash
read -p "Number 1: " x
read -p "Number 2: " y
if [ $x -lt $y ]; then echo "$x smaller than $y"
elif [ $x -eq $y ]; then echo "$x = $y"
else echo "$x bigger than $y"; fi
```

```
GNU nano 6.2 script2.sh
#!/bin/bash
read -p "Number 1: " x
read -p "Number 2: " y
if [ $x -lt $y ]; then
    echo "$x smaller than $y"
elif [ $x -eq $y ]; then
    echo "$x = $y"
else
    echo "$x bigger than $y"
fi
```

# Homework 3: Exercise 1

1. Download file `text.txt` from the course website
  - Use browser and save file as plain text
2. Check to see the file exists in your directory in the Terminal

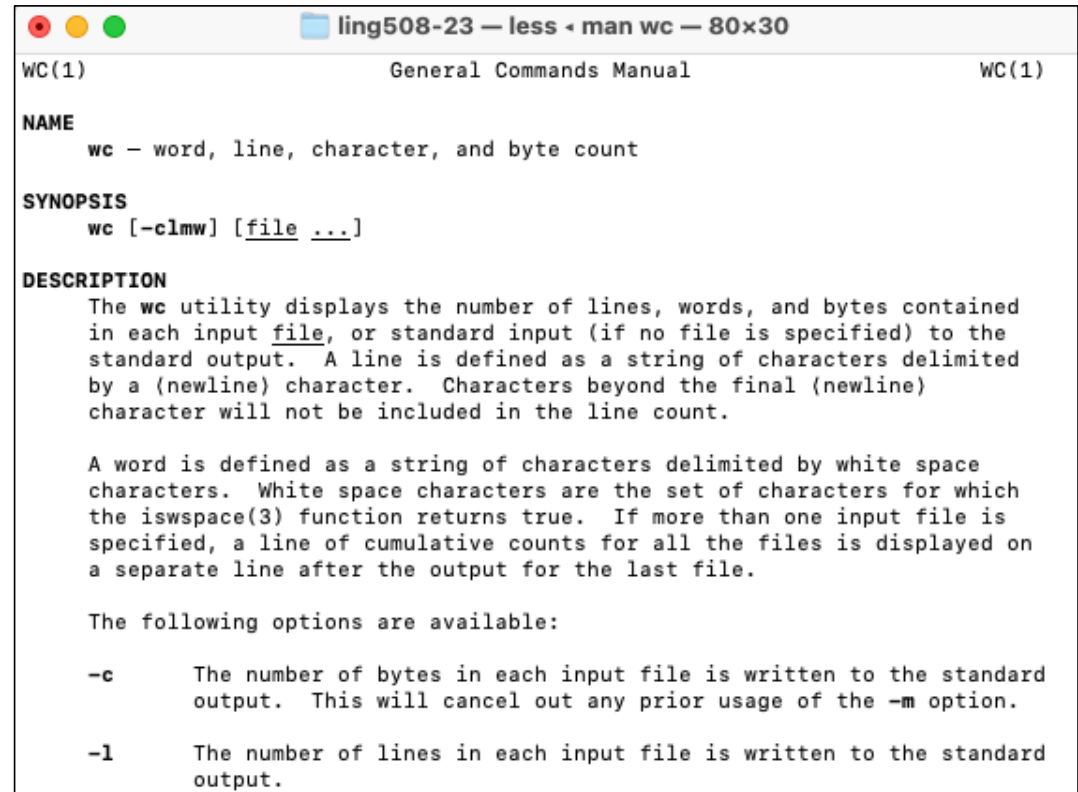


```
ling508-23 — -bash — 80x30
[$ file text.txt
text.txt: ASCII text
[$ ls -l text.txt
-rw-r--r--  1 sandiway  staff  1842 Aug 30 15:47 text.txt
$
```

1842 bytes in size

# Homework 3: Exercise 1

`wc` is a useful command.  
Do `man wc` to see the  
manual page (manpage).



```
ling508-23 — less ◀ man wc — 80×30
WC(1)                                     General Commands Manual                                     WC(1)
NAME
  wc - word, line, character, and byte count
SYNOPSIS
  wc [-clmw] [file ...]
DESCRIPTION
  The wc utility displays the number of lines, words, and bytes contained
  in each input file, or standard input (if no file is specified) to the
  standard output. A line is defined as a string of characters delimited
  by a {newline} character. Characters beyond the final {newline}
  character will not be included in the line count.

  A word is defined as a string of characters delimited by white space
  characters. White space characters are the set of characters for which
  the iswspace(3) function returns true. If more than one input file is
  specified, a line of cumulative counts for all the files is displayed on
  a separate line after the output for the last file.

  The following options are available:

  -c      The number of bytes in each input file is written to the standard
          output. This will cancel out any prior usage of the -m option.

  -l      The number of lines in each input file is written to the standard
          output.
```

## Homework 3: Exercise 1

4. Try `wc text.txt`. Find out from the manpage what the three numbers reported mean. Screenshot it.
5. What's the `wc` option that prints the number of words only? Try it.

# Homework 3: Exercise 1

6. `nano` `text.txt`. Type Control-G in the text editor to see the help text. Scroll down and find the command for counting number of words, lines and characters? What is that command and how do you type it? What is reported? Screenshot it.
- You can type Control-X or Control-G to go back to displaying `text.txt`
  - Compare your answer with that obtained in 5.

## Homework 3: Exercise 2

- Let's use the Terminal to make a frequency list of the words in `text.txt`
  - First, look at the manpage for command `tr`.
  - Next, let's replace all the punctuation characters by spaces.
1. Observe the output of both commands below. Which command do we want?
    - `cat text.txt | tr '[:punct:]' ' '`
    - `cat text.txt | tr -d '[:punct:]'`
    - **Note 1:** pipe (`|`) sends the output of the `cat` command as input to the next command `tr`.

*note: a space here*

# Homework 3: Exercise 2

- **Note 2:** we can redirect ('>') the output of the above command into a file, e.g. `text2.txt`, as follows:

- `command > text2.txt`

2. Next, we can put each word on a separate line using:

- `tr ' ' '\n'`

- **Note 3:** `\n` stands for a newline character.

3. Combine the previous two steps (1 & 2) together.

- **Note 4:** you can use `text2.txt` (*if you saved the output*), or just chain the command onto the end, i.e. do:

- `command | tr ' ' '\n'`

## Homework 3: Exercise 2

- Next, look at the manpage for command `uniq`.
4. Let's make a table of the frequency counts for each word using:
    - `sort | uniq -c`
  - by running the above command on the output of Step 3 above.
    - **Note 5:** you can chain this command as mentioned earlier, or save the output of step 4 into another file, e.g. `text3.txt`. Example:
      - `command | sort | uniq -c`
  5. Why do we sort first? Read the `uniq` manpage to find out.



## Homework 3: Exercise 2

6. Let's put the results in sorted order of frequency (*descending*) by appending:
  - `sort -rn`
  - to our list of commands so far (step 4 above).
  - Be sure to chain it using the pipe (`|`).
    - consult the `sort` manpage to find out what the options `-r` and `-n` above do.
  - Show your output.

# Instructions

- Email to [sandiway@arizona.edu](mailto:sandiway@arizona.edu)
- By Sunday midnight (*will be graded on Monday*)
- SUBJECT: 408/508 Homework 3: *YOUR NAME*
- PDF file please, screenshots should be inside, not separate attachments
- (do not submit Word .docx or .doc files)