Lecture 9

408/508 *Computational Techniques for Linguists*

Today's Topics

- Homework 4 review
- A bit more shell scripting (*we'll finish this week*)
  - *remember you can test everything on the command line*!
  - deleting suffixes and prefixes, string manipulation
  - loops
  - cp (*copy*)
  - positional parameters: $0, $1, $2 …
  - filename expansion: **globbing**

Homework 4 review

```
1#!/bin/bash
2# usage: weight height kg/lbs
3if [ $# -ne 3 ]; then
4    read -p "weight in kg (lbs): " weight
5    read -p "height in cm (in): " height
6    read -n 1 -p "units kg/lbs: " units
7    echo
8else
9    weight=$1
10   height=$2
11   units=${3:0:1}
12fi
```

$3 substring (1st character)

Notice variables `weight`, `height` and `units`
are set no matter whether the inputs come from command line arguments or the Terminal

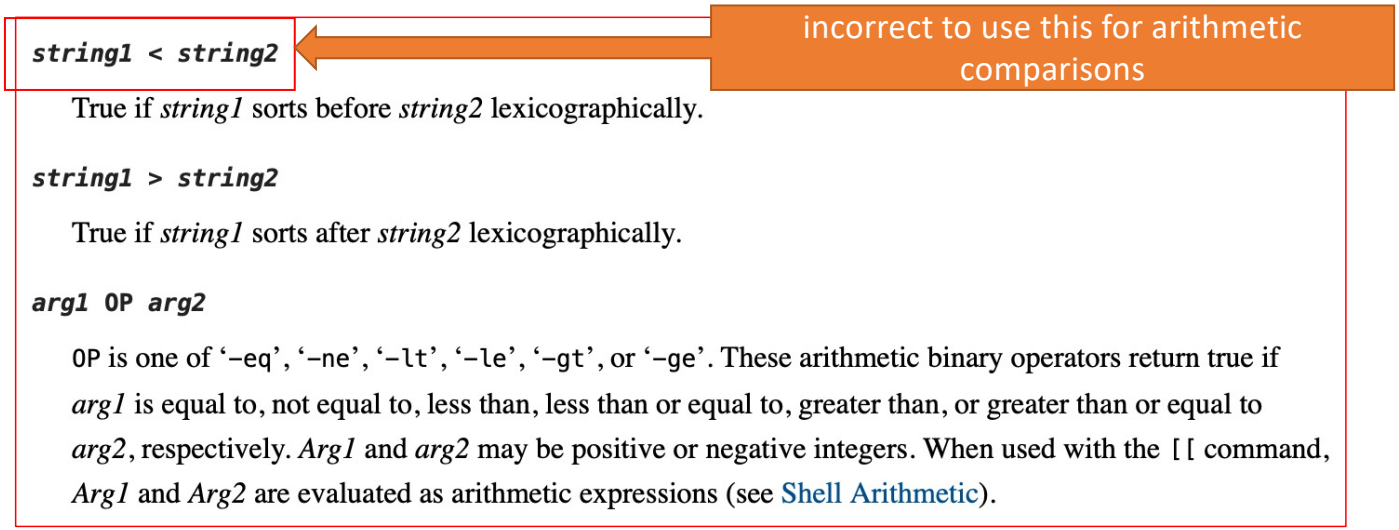
Homework 4 review

```
13 if [ $units = "k" ]; then¶
14     bmi=$(echo "scale=1; $weight*10000/($height*$height)" | bc)¶
15 else¶
16     bmi=$(echo "scale=1; $weight*703/($height*$height)" | bc)¶
17 fi¶
18 echo $bmi¶
19 bmi=$(echo "$bmi*10/1" | bc)¶
20 if [[ $bmi -lt 185 ]]; then¶
21     echo "underweight"¶
22 elif [[ $bmi -lt 250 ]]; then¶
23     echo "normal"¶
24 elif [[ $bmi -lt 300 ]]; then¶
25     echo "overweight"¶
26 else¶
27     echo "obese"¶
28 fi¶
```

Recall Bash doesn't do floating point numbers, so scale by ten. Then 18.9 becomes 189.0. Then *trick* divide by 1 to get 189

Homework 4 review

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



string1 < string2

True if *string1* sorts before *string2* lexicographically.

string1 > string2

True if *string1* sorts after *string2* lexicographically.

arg1 OP arg2

`OP` is one of '`-eq`', '`-ne`', '`-lt`', '`-le`', '`-gt`', or '`-ge`'. These arithmetic binary operators return true if *arg1* is equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to *arg2*, respectively. *Arg1* and *arg2* may be positive or negative integers. When used with the `[[` command, *Arg1* and *Arg2* are evaluated as arithmetic expressions (see Shell Arithmetic).

Delete Word Suffix

```
(base) ling508-22$ b='file.txt'
(base) ling508-22$ c=${b%.txt}
(base) ling508-22$ echo $c
file
(base) ling508-22$
```

3.5.3 Shell Parameter Expansion
https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html#Shell-Parameter-Expansion

`${parameter%word}`

`${parameter%%word}`

The *word* is expanded to produce a pattern and matched according to the rules described below (see Pattern Matching). If the pattern matches a trailing portion of the expanded value of *parameter*, then the result of the expansion is the value of *parameter* with the shortest matching pattern (the '`%`' case) or the longest matching pattern (the '`%%`' case) deleted. If *parameter* is '`@`' or '`*`', the pattern removal operation is applied to each positional parameter in turn, and the expansion is the resultant list. If *parameter* is an array variable subscripted with '`@`' or '`*`', the pattern removal operation is applied to each member of the array in turn, and the expansion is the resultant list.

Delete Word Prefix

```
[ling508-20$ b='file.txt'
[ling508-20$ echo ${b##*\.}
 txt
[ling508-20$ echo ${b#*\.}
 txt
 ling508-20$
```

${*parameter*#*word*}

${*parameter*##*word*}

The *word* is expanded to produce a pattern and matched according to the rules described below (see Pattern Matching). If the pattern matches the beginning of the expanded value of *parameter*, then the result of the expansion is the expanded value of *parameter* with the shortest matching pattern (the '#' case) or the longest matching pattern (the '##' case) deleted. If *parameter* is '@' or '*', the pattern removal operation is applied to each positional parameter in turn, and the expansion is the resultant list. If *parameter* is an array variable subscripted with '@' or '*', the pattern removal operation is applied to each member of the array in turn, and the expansion is the resultant list.

More string manipulation

- String length:
  - `${#var}`
- Substring:
  - `${var:position}`
  - `${var:position:length}`
- Delete prefix:
  - `${var#substring}`
  - `${var##substring}`
- Delete suffix:
  - `${var%substring}`
  - `${var%%substring}`
- Substring substitution:
  - `${var/substring/replacement}`
  - `${var//substring/replacement}`
- Prefix substitution: `${var/#substring/replacement}`
- Suffix substitution: `${var/%substring/replacement}`

See **3.5.3 Shell Parameter Expansion**

starting at position (0,1,2…), (-N) from the end

```
units=${3:0:1}
```

shortest match
longest match

```
cp $file ${file%.*}.bak
```

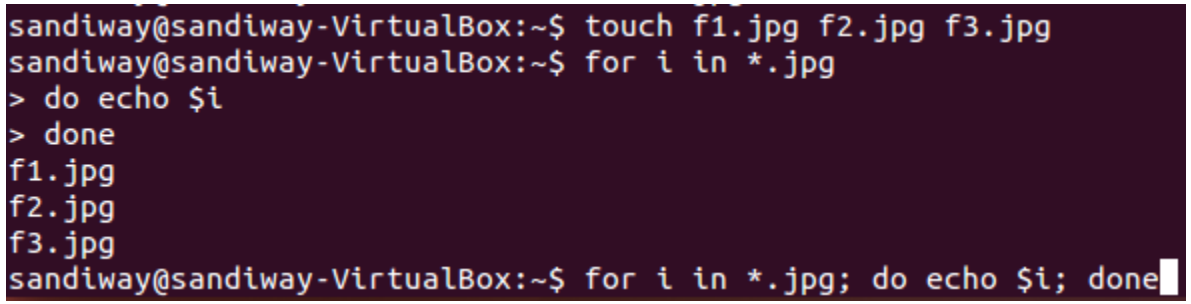
shortest match
longest match

replace first match
replace all matches

loops

- **For-loop**:
  - `for` *VAR* `[in` *LIST*`]; do …; done`
- **Example**:
  - create backup copies of files using `cp` (copy)
  - (*let's create a few empty .jpg files first using* `touch`)
  - `for i in *.jpg; do echo $i ; done`
  - `for i in *.jpg; do cp $i $i.orig ; done`

`touch f1.jpg f2.jpg f3.jpg`
Brace expansion also possible:
`touch f{1,2,3}.jpg`



A note on cp (copy)

- **man cp**

-p Cause **cp** to preserve the following attributes of each source file in the copy: modification time, access time, file flags, file mode, user ID, and group ID, as allowed by permissions.

EXAMPLES

Make a copy of file foo named bar:

```
$ cp foo bar
```

Copy a group of files to the /tmp directory:

```
$ cp *.txt /tmp
```

Copy the directory junk and all of its contents (including any subdirectories) to the /tmp directory:

```
$ cp -R junk /tmp
```

loops

- `until COND; do …; done`

- **while-loop**:
  - `while` *`COND`*`; do …; done`
  - `break` (*out of while-loop*)

```
(base) ling508-22$ chmod 744 line30.sh
(base) ling508-22$ ls -l line30.sh
-rwxr--r--  1 sandiway  staff  166 Oct  2 16:56 line30.sh
(base) ling508-22$ ./line30.sh
Next word: this
5: this
(base) ling508-22$ ./line30.sh
Next word: this
Next word: is
Next word: a
Next word: test
Next word: of
Next word: the
Next word: line
Next word: length
34: this is a test of the line length
(base) ling508-22$
```

file: `line30.sh`
${*#var*} length of string stored in *var*

```
1#!/bin/bash¶
2line=""¶
3while true¶
4do¶
5    read -p "Next word: " word¶
6    line="$line $word"¶
7    if [[ ${#line} -gt 30 ]]¶
8        then break¶
9    fi¶
10done¶
11echo ${#line}:$line¶
12exit 0¶
```

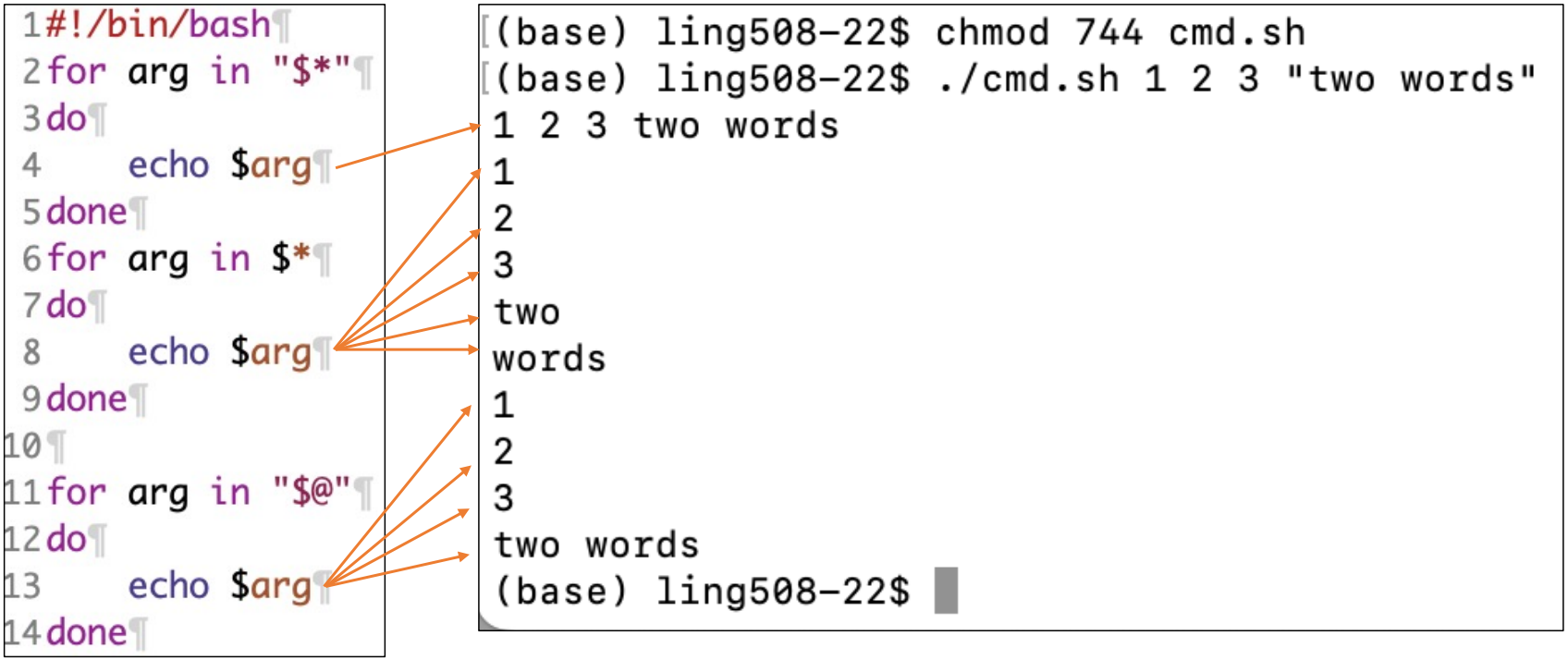
Shell positional parameters

- command line to script (or function):
  - `$0, $1, $2` *etc.* — `$1` = 1st parameter and so on, `$0` = name of script
  - `$#` — number of parameters passed
  - `$*` — all parameters as a single word: `a "b c" => a b c`
  - `$@` — each parameter is a quoted string
  - `shift` — removes one parameter (use with $#)
  - ("…" *quoted variables below just in case there are spaces in the values*)

```
for arg in "$*"
do
  echo $arg
done
```

```
for arg in "$@"
do
  echo $arg
done
```

Positional Parameters



Expansion: different kinds

- Arithmetic expansion:
  - $((… expression ..))
    - `x=$(($x+1))`
- Pathname expansion (aka ***globbing***):
  - similar (but not the same) as regular expressions
    - * (*wild card string*)
    - ? (*wild card character*)
    - [ab] (*a or b*)
    - [^ab] (*not* (*a or b*))
- (curly) Brace expansion:
    - `mkdir ~/class/examples/{ex1,ex2}`
    - `echo x{1,2,3,4}`

Use command `ls`
in conjunction with globbing

Expansion

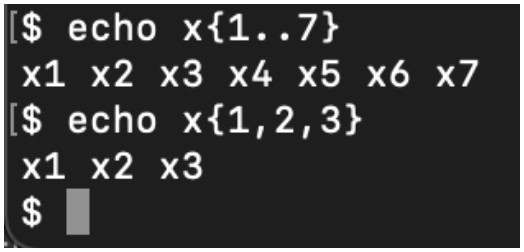
- Pathname expansion (aka ***globbing***):
  - similar (*but not the same*) as regular expressions
    - `*` (*wild card string*)
    - `?` (*wild card character*)
    - `[ab]` (*a or b*)
    - `[^ab]` (*not* (*a or b*))
- Examples:
  - `ls f[23]*.jpg`
  - `ls f[^4]*.jpg`

Expansion

- (curly) Brace expansion:
  - `mkdir ~/class/examples/{ex1,ex2}`

  shorthand for:
  - `mkdir ~/class/examples/ex1 ~/class/examples/ex2`

  - `echo x{1,2,3,4}` (or `echo x{1..4}`)
  - shorthand for:
  - `echo x1 x2 x3 x4`



File extension renaming

Script (`rmext.sh`):

```
#!/bin/bash
if [[ $# -ne 2 ]]; then
echo “usage: ext1 ext2”
exit 1
fi
for filename in *.$1
do
  mv "$filename" "${filename%$1}$2"
done
exit 0
```

Exercise: create a subdirectory with some .JPG files, rename to .jpg

```
$ chmod 744 rmext.sh
$ mkdir tmp
$ cd tmp
$ touch f{1..7}.JPG
$ ls
f1.JPG f2.JPG f3.JPG f4.JPG f5.JPG f6.JPG f7.JPG
$ ../rmext.sh JPG jpg
$ ls
f1.jpg f2.jpg f3.jpg f4.jpg f5.jpg f6.jpg f7.jpg
```

⇑ delete suffix: `${string%substring}`

- "… " just in case there are spaces in the filenames

File renaming

- Example:
  - *append a suffix* –1 *to all jpg files*
  - `for f in *.jpg; do mv $f ${f/./-1.}; done`

Substring substitution:
`${string/substring/replacement}`

- Levels of quoting:

```
$ echo text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
text /home/me/ls-output.txt a b foo 4 me
$ echo "text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER"
text ~/*.txt {a,b} foo 4 me
$ echo 'text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER'
text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
```

http://linuxcommand.org