Lecture 04 - Unix loops

for loops in general

for variable **in** set

    code to be executed

- variable: a place holder used to temporarily hold each element in the set
- set: the entities you want to have the code within the for loop executed for

for loops in Unix

```
for variable in set
do
    code to be executed using $variable
done
```

potato example

```
for number in one two
do
  echo $number potato
done
```

```
## one potato
## two potato
```

Executing a for loop at the Unix command line

- usually the Unix command prompt is a $
- once you've begun a for loop the prompt changes to >
- this is only to remind you that you are in a loop
- you can get out of the loop or any running command with *Ctrl-c*

Reminder again

Most often the variables we will include in a set are file names, therefore you should remember. . .

- NO SPACES IN FILE NAMES!!!!!

- File extensions don't mean anything to Unix!

By the way. . .

Did anyone else get files with name stats-stats-filename when working through tutorial?

Why?

How to avoid this?

More than one shell window!!!

Holy cows, you can do this?

Remember multi-processing!

Echo and debugging

Why is `echo` useful for testing `for` loops?

Echo and debugging

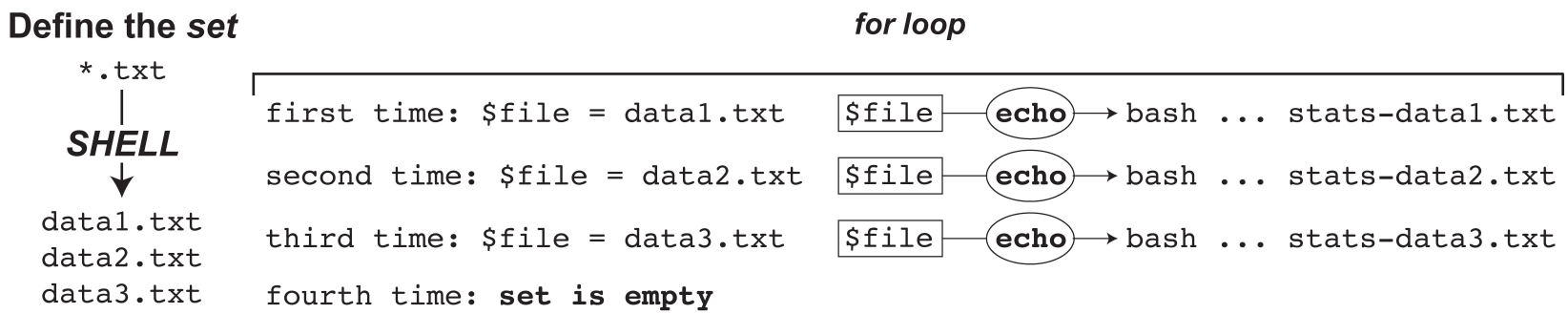
Why is `echo` useful for testing `for` loops?

```
for file in *.txt
do
  echo "bash goostats $file stats-$file"
done
```

```
## bash goostats data1.txt stats-data1.txt
## bash goostats data2.txt stats-data2.txt
## bash goostats data3.txt stats-data3.txt
```

What happened on a process level?

```
for file in *.txt
do
  echo "bash goostats $file stats-$file"
done
```



for loop application

Challenge: Use the files in `molecules/` to print each # of atoms:

```
cubane:
     16
ethane:
     8
methane:
     5
octane;
     26
pentane:
     17
propane:
     11
```

for loop application

```
for file in *.pdb
do
  echo $file | sed 's/.pdb/: /'
  cat $file | grep ATOM | wc -l
done
```

nested for loops

- As shown in the tutorial, this is useful if you have multiple directories with similar files

```
for dir in */
do
  for file in *.txt
  do
    echo "$file is in $dir"
  done
done
```

looping across files or in files?

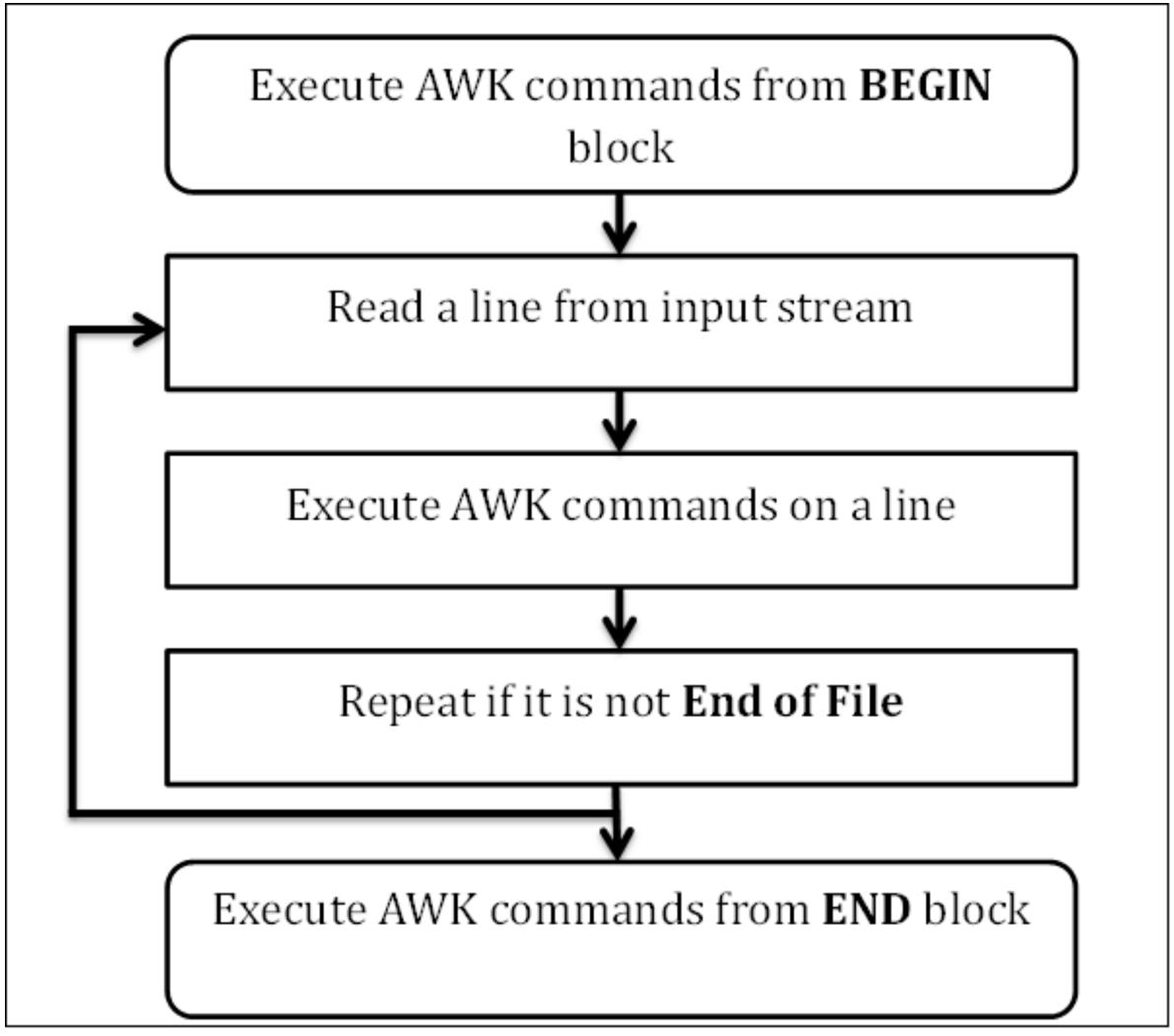
looping across files or in files?

- hopefully clear why looping across files is useful
- do we need to loop in files? why or why not?

looping through files

- generally not necessary because of pipes and filters
- to do more complex manipulations of each line in a file use `awk`, especially when a file is tabular in form

awk process


Execute AWK commands from **BEGIN** block
Read a line from input stream
Execute AWK commands on a line
Repeat if it is not **End of File**
Execute AWK commands from **END** block

awk syntax

```
awk 'BEGIN{print "START"}{print $1, $4}END{print
"STOP"}'
```

awk syntax made up of three components

1. BEGIN block can be used to print field names or set option values; this is optional
2. action block is the syntax executed for each line of the file passed to awk; often this takes the form of printing a subset of the available fields, but is very flexible
3. END block can be used to print additional components at the end of the output

awk options

- a pattern can be specified prior to the `action` block as `/pattern/`, and the `action` block will only be executed when the pattern matches a given line

- the input field separator can be specified in the `BEGIN` block with `FS="separator"`; this defaults to whitespace

- the output field separator can also be specified in the `BEGIN` block with `OFS="separator"`; this defaults to a space, and can also be done manually in the `action` block

- `awk` is quite sophisticated; for example, if-else statements can be used, etc.

awk application

Challenge: Use `awk` to print the name then date of each dragonfly sighting listed in 'insectSightings.txt' from Exercise 2. Separate the name and date on each line with a comma, and put the output in a file.

awk application

Challenge: Use `awk` to print the name then date of each dragonfly sighting listed in 'insectSightings.txt' from Exercise 2. Separate the name and date on each line with a comma, and put the output in a file.

```
cat insectSightings.txt | grep dragonfly |
awk '{print $3 "," $2} > dragonflySightings.txt
```

function review

`for`

`$`

`history`

`awk`

up arrow

Ctrl-c

Ctrl-a

Ctrl-e