1. AWK / GAWK

2. More Branching
Some Logistics

• HW2 is online...officially!
• Subtle changes to README.md, none that are important except:
  • (OH Yesterday): I am giving sample files.
  • Lecture 08 demo will be updated soon: using different separators in sed.
  • Excellent Piazza question: why is read behaving this way?
  • Directory structure sort of changed, but only in that you get more files. No changes to instructions.
• Challenge task at end.
• You are FORBIDDEN from using today's lecture in HW2, except for the gandalfify_extreme.sh challenge question.
• (Poll) should I even cover Python?
AWK / GAWK
• **awk** is a programming language designed for processing text-based data.
  • Allows easy operation on fields rather than full lines.
  • Works in a *pattern-action* manner, like *sed*.
  • Supports numerical types (and operations).
  • Supports control-flow (e.g. *if-else* statements).

• Created at Bell Labs in the 1970s.
  • Alfred Aho, Peter Weinberger, and Brian Kernighan.
  • An ancestor of *perl*, a *cousin* of *sed*.

• Very powerful.
  • It's *Turing Complete*!
• **gawk** is the GNU implementation of the **awk** programming language.
• On BSD/OSX, it is just called **awk**.
• On GNU, it is technically **gawk**. But should reliably be "symlinked" as **awk**.
• **awk** allows us to setup filters to handle text as easily as numbers.
• The basic structure of an **awk** program is:
  ```plaintext
  pattern1 { commands }
  pattern2 { commands }
  ...
  ```
• Patterns can be regular expressions!
  • Proceeds line by line, checking each pattern one by one, executing **commands** if **pattern** is found.
Why use `awk` over `sed`?

- Convenient numerical processing.
- Variables and control flow in the actions.
- Convenient way of accessing fields within lines.
- Flexible printing.
- Built-in arithmetic and string functions.
Simple Examples

`awk '/[Mm]onster/ {print}' frankenstein.txt`
- Print all lines containing Monster or monster.

`awk '/[Mm]onster/' frankenstein.txt`
- If no action specified, default is to print the whole line.

`awk '/[Mm]onster/ {print $0}' frankenstein.txt`
- The $0 variable in awk refers to the whole line.

`awk '/[Mm]onster/ {print $1}' frankenstein.txt`
- The first item. Can be delimited by something other than whitespace, just like sed.

- awk understand extended regular expressions by default :) 
  - We don't need to escape +, ?, etc!
• **awk** allows blocks of code to be executed only once, at the beginning / end.

• With the script `monstrosity.awk` and `frankenstein.txt` in current directory:

```bash
#!/usr/bin/awk -f
BEGIN { print "Starting search for monster..." }
/[Mm]onster/{ count++ }
END { print "Found " count " monsters in the book." }
```

• Use the `-f` in conjunction with shebang to cheat **awk** (it uses the script itself).

```bash
>>> ./monstrosity.awk # hangs...
>>> ./monstrosity.awk frankenstein.txt # yay!
>>> awk -f monstrosity.awk frankenstein.txt # yay!
```
• **NF**: the number of fields in the current line.
  • You cannot change NF or NR.

• **NR**: the number of lines read so far.

• **FILENAME**: the name of the input file.

• **FS**: the field separator.
  • Change FS="," for a csv.
  • Can also specify the -F flag for the FS.
Matching and awk

- **awk** can match any of the following pattern types:
  - `/regular expression/`
  - relational expression
  - `pattern && pattern`
  - `pattern || pattern`
  - `pattern1 ? pattern2: pattern3`
    - If `pattern1`, then match `pattern2`. Otherwise, match `pattern3`.
  - `(pattern)`: parenthesis to group / change order of operations.
  - `! pattern` to invert.
  - `pattern1, pattern2`: match `pattern1`, work on every line until it matches `pattern2`.
    - Cannot combine this...
Much Much More...

- Regular expression usage / comparisons:

- More comparison operations:

- Powerful built-in functions:
  - `toupper()`
  - `tolower()`
  - `exp(x)`: exponential of `x`
  - `rand()`: random number between 0 and 1
  - `length(x)`: length of `x`
  - `log(x)`: returns the log of `x`
  - `sin(x)`: returns the sin of `x`
  - `int(x)`: convert to integer
  - etc

More Branching
Lecture slides...PART II!
Previous cornell cs 2043 course slides.