CSCI: 4500/6500 Programming Languages

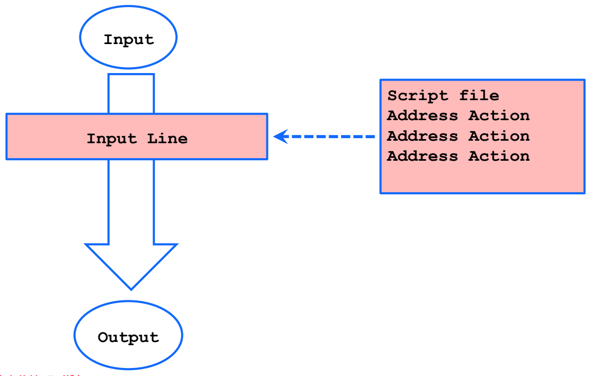
SED & AWK

sed: Stream Oriented, Non-Interactive, Text Editor

- Line-oriented tool for pattern matching and replacement (stream editor)
  - » Looks for patterns one line at a time, like grep
  - » *"Change"* lines of the file (but acts as a filter)
    - – Filter, i.e., does not modify input file
  - » There is an interactive editor *ed* that accepts the same commands
  - » UNIX philosophy – edit a stream, a stream flowing through a pipe
- Sed is not really a programming language (but AWK is)

- Basic Syntax
- [address(es)]s/pattern/replacement/[flags]

Sed Architecture



```
Script file
Address Action
Address Action
Address Action
```

Awful Syntax

- `sed [-n] [-e] ['command'] [file…]`
- `sed [-n] [-f scriptfile] [file…]`
  - » *-n* – supress output of input lines
  - » *-f scriptfile* - next argument is a filename containing editing commands
  - » *-e command* - the next argument is an editing command rather than a filename, useful if multiple commands are specified
    - – s the ultimate substitution command :
      - `sed s/day/night/ < old > new`
      - `sed s/day/night/ old > new`

Command! (function)

- `sed [-n] [-e] ['command'] [file…]`
- Command Details:
  - s – substitution
  - – [address(es)]s/pattern/replacement/[flags]
  - – `sed s/day/night/`
  - – flags – example 'g' for global, 'n' which occurrence of pattern should be replaced
  - » d – delete
  - » And more: y-transform, p-print

More Warm-up Examples

- s/Tom/Dick/2
  - » Substitutes Dick for the second occurrence of Tom in the pattern space
- s/wood/plastic/p
  - » Substitutes plastic for the first occurrence of wood and outputs (prints) pattern space

Constraining matches by *addressing*

- Commands can be constrained to accept only single line addresses or ranges of address (or a pattern).

- **Diving In Example:**
  - » echo “The UNIX operating system” | sed 's/.NI./wonderful &/’
- **Ouch!**
  - » Special replacement/patterns – characters
  - » & - replaced by the entire string matched in the regular expression for pattern

Another Example

- *sed [-n] [-e] [‘command’] [file…]*
- Escape : \
- . On character

```
{saffron} cat first.txt
first:second
one:two
{saffron} sed 's/\(.*\):\(.*\)/\2:\1/' test1
```

Another Example

- *sed [-n] [-e] [‘command’] [file…]*
- *Escape,*
- *Marking patterns (up to 9): “\(“, \)”*

```
{saffron} cat test1.txt
first:second
one:two
{saffron} sed 's/\(.*\):\(.*\)/\2:\1/' test1
second:first
two:one
```

Address Example

- *Address:*
  - » delete lines 1-10: `sed -e '1,10d‘`

```
{h70-33-107-14:ingrid:919} sed -e ‘5,14d' afile.txt
1
2
3
4
{h70-33-107-14:ingrid:920}
```

More examples

- Convert unix to dos characters.
  - » `sed -e 's/$/\r/' myunix.txt > mydos.txt`
- Transform with y (by character position)
  - `echo “maria hybinette” |  sed -e ’y/aie/xyz/’`
- `s/Tom/Dick/2`
  - » Substitutes Dick for the second occurrence of Tom in the *pattern space*
- `s/wood/plastic/p`
  - » Substitutes plastic for the first occurrence of wood and outputs (prints) *pattern space*

Append, Insert, and Change

Syntax for these commands is a little strange because they must be specified on multiple lines

append	*[address]a*
	text
insert	*[address]i*
	text
change	*[address(es)]c*
	text

- append/insert for single lines only, not range

Change Examples

- Remove mail headers, ie; the address specifies a range of lines beginning with a line that begins with From until the first blank line.
  - » The first example replaces all lines with a single occurrence of <Mail Header Removed>.
  - » The second example replaces each line with <Mail Header Removed>

```
/^From: /,/^$/c\
  <Mail Headers Removed>
```

```
/^From: /,/^$/{
   s/^From //p
   c\
   <Mail Header Removed>
   }
```

Sed Advantages

- Regular expressions
- Fast
- Concise

Sed Drawbacks

- Hard to remember text from one line to another
- Not possible to go backward in the file
- No way to do forward references like `/..../+1`
- No facilities to manipulate numbers
- Cumbersome syntax

Awk

Programmable Filters

Why is it called AWK?

Awk Introduction

- A general purpose programmable filter that handles text (strings) as easily as numbers
  - » This makes awk one of the most powerful of the Unix utilities
- awk processes *fields* while sed only processes lines
- nawk (new awk) is the new standard for awk
  - » Designed to facilitate large awk programs
  - » gawk is a free nawk clone from GNU

Awk Input

- awk gets its input from
  - » files
  - » redirection and pipes
  - » directly from standard input

AWK Highlights

- A programming language for handling common data manipulation tasks with only a few lines of code
- awk is a *pattern-action* language, like sed
- Looks like *C* but automatically handles input, field splitting, initialization, and memory management
  - » Built-in string and number data types
  - » No variable type declarations
- awk is a great prototyping language
  - » Start with a few lines and keep adding until it does what you want

Awk Features over Sed

- Convenient numeric processing
- Variables and control flow in the actions
- Convenient way of accessing fields within lines
- Flexible printing
- Built-in arithmetic and string functions
- C-like syntax

Structure of an AWK Program

- An optional BEGIN segment
  - – For processing to execute prior to reading input
- pattern - action pairs
  - – Processing for input data
  - – For each pattern matched, the corresponding action is taken
- An optional END segment
  - – Processing after end of input data

```
BEGIN {action}
pattern
{action}
pattern
{action}
   .
   .
   .
pattern
{ action}
END {action}
```

Review: What is AWK?

- Programming language used for manipulating data and generating pretty reports.
  - Job control too.

Running an AWK Program

- There are several ways to run an Awk program
  - *awk 'program' input_file(s)*
    - program and input files are provided as command-line arguments
  - *awk 'program'*
    - program is a command-line argument; input is taken from standard input (yes, awk is a filter!)
  - *awk -f program_file input_files*
    - program is read from a file

Patterns and Actions

- Search a set of files for *patterns*.
- Perform specified *actions* upon lines or fields that contain instances of patterns.
- Does not alter input files.
- Process one input line at a time
- So for this is similar to sed (except fields)

Pattern-Action Structure

- Every program statement has to have a *pattern* or an *action* or both
  - Default *pattern* is to match all lines
  - Default *action* is to print current record
- Patterns are simply listed;
  - actions are enclosed in `{ }`
- awk scans a sequence of input *lines*, or *records*, one by one, searching for lines that match the pattern
  - Meaning of match depends on the pattern

Patterns

- A selector that determines whether *action* is to be executed
- *pattern* can be:
  - the special token BEGIN or END
  - regular expression (enclosed with //)
  - relational or string match expression
  - `!` negates the match
  - arbitrary combination of the above using `&& ||`
    - `/NYU/` matches if the string “NYU” is in the record
    - `x > 0` matches if the condition is true
    - `/NYU/ && (name == "UNIX Tools")`

BEGIN and END patterns

- BEGIN and END provide a way to gain control before and after processing, for initialization and wrap-up.
  - BEGIN: actions are performed before the first input line is read.
  - END: actions are done after the last input line has been processed.

Actions

- ***Action***
  - » list of one or more C like statements
  - » arithmetic and string expressions and
  - » assignments and multiple output streams.
- ***action* is performed on every line that matches *pattern*.**
  - » If *pattern* is not provided, *action* is performed on every input line
  - » If *action* is not provided, all matching lines are sent to standard output.

An Example

```
ls | awk '
  BEGIN { print "List of html files:" }
  /\.html$/ { print }
  END { print "There you go!" }
  '
```

```
List of html files:
index.html
as1.html
as2.html
There you go!
```

Awk examples

- Add up first column, print sum and average
- {s += $1 }
- END {print “sum is”, s, “average is”, s/NR}
- awk -f awkprogram awkfile

Tutorials

- SED
  - » http://www.grymoire.com/Unix/Sed.html
    - – Great reference card available
  - » http://sed.sourceforge.net/grabbag/tutorials/
- AWK