Sed and awk

Kameswari Chebrolu

EVOLUTION OF LANGUAGE THROUGH THE AGES.

6000 B.C.

UNGH. GRRF. BOOGA.
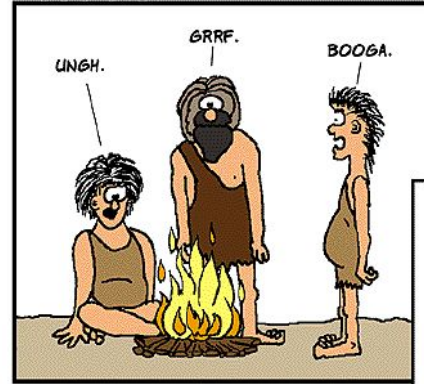
2000 A.D.

GREP. AWK. SED.
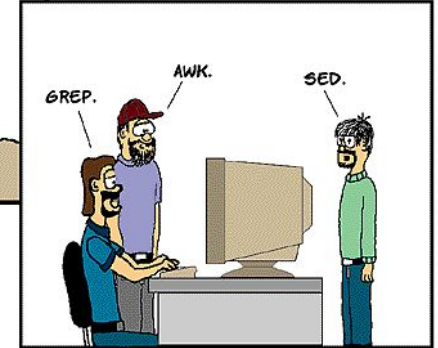
COPYRIGHT (C) 1999 ILLIAD   HTTP://WWW.USERFRIENDLY.ORG/

Sed/Awk

- Powerful text processing utilities
- sed: a non-interactive text editor
- awk: a field-based pattern processing language with a C-style syntax
- Both use
  - regular expressions
  - read input from stdin/files, and output to stdout

Sed (stream editor)

sed: Stream-oriented, Non-Interactive, Text Editor
- Stream: Look one line at a time
- Non-interactive: editing commands come in as script
- Text editor: change lines of a file
  - Sed is more a filter
  - Original input file is unchanged
  - Results sent to standard output (can be redirected to a file)

Sed Control Flow
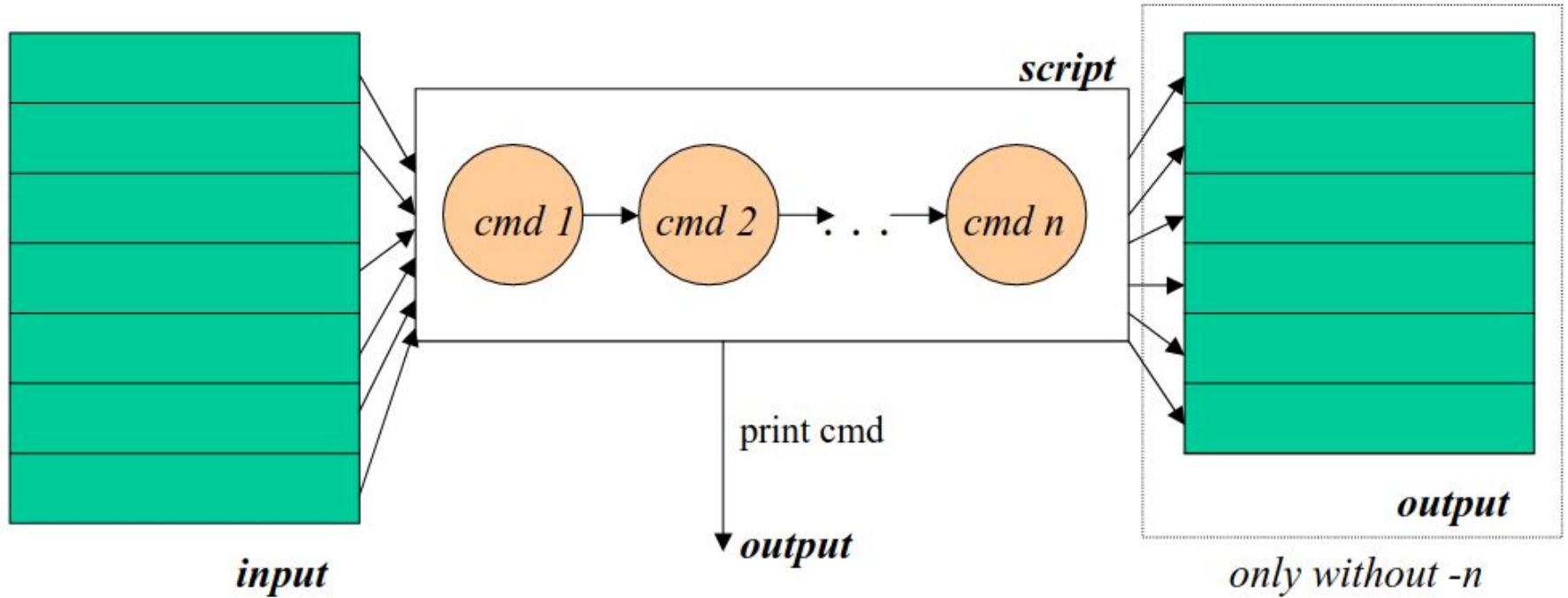
- A script is a file made of commands
- Commands also specify
  - regular expression to match a pattern (and/or)
  - an address range (line nos in a file)
- Single quotes (' ') are used to delimit the script or command being executed

- Commands are applied in order to each input line
  - sed reads the first command and checks address/pattern against the current input line
    - match, command is executed
    - no match, command is ignored
  - sed then repeats this action for every command in the script file
    - Note: If a command changes the input, subsequent command will be applied to the modified line
  - Reached end of the script? output the (modified?) line unless "-n" option is set

Delete Command

- Syntax: sed 'ADDRESSd' filename or sed '/PATTERN/d' filename
- Examples:
  - '6d' (deletes line 6)
 - '1, 5d' (deletes lines 1 to 5)

Substitute Command

- Syntax: sed 'address(es) s/pattern/replacement/[flags]' filename
  - Flags:
    - a number from 1 to 512 indicating address
    - g: global, replace all occurrences of pattern in pattern space
    - p: print contents
    - I is case insensitive
  - E.g.
    - sed 's/wolf/fox/' bigfile
    - sed '3 s/wolf/fox/' bigfile
    - sed 's/wolf/fox/3gI' bigfile
    - echo "Welcome To The Course CS104" | sed 's/\(\b[A-Z]\)/\(\1\)/g'

Print Command

- Syntax: sed 'ADDRESSp' filename or sed '/PATTERN/p' filename
- Used to print the matched pattern
  - Often used with the -n option
    - Without the -n option, sed automatically prints each line after applying editing commands.
    - With the -n option, sed will only print output when explicitly instructed using the p command
- Syntax: [address/pattern]p

  - sed '1p' bigfile
  - sed -n '1p' bigfile

Sed Control Flow
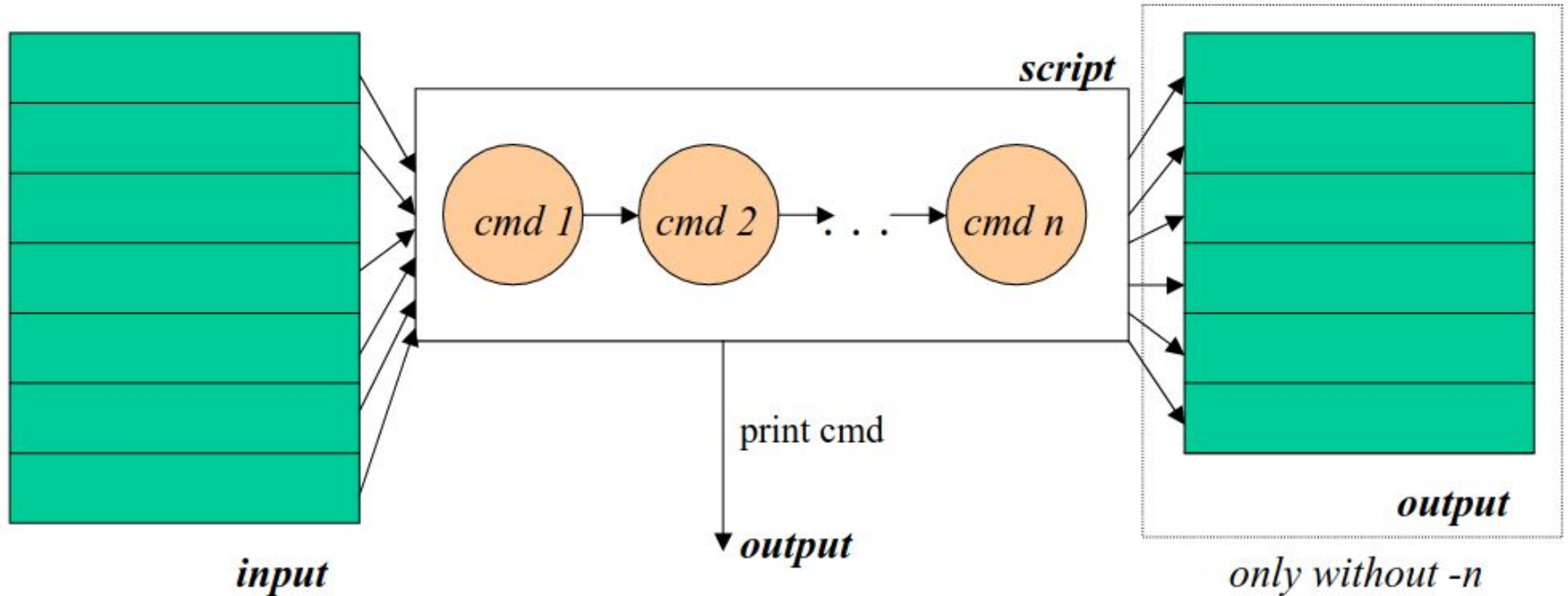
Append/Insert/Replace

- Append: [address/pattern]a file
  - Append places text after the current line in pattern space
  - sed '2a tomato' fruits
- Insert
  - Insert places text before the current line in pattern space
  - sed '2i tomato' fruits
- Replace
  - Replaces
  - sed '2c aam' fruits

quit

- Quit causes sed to stop reading new input lines
  - Once a line matches the pattern/address, the script terminated
  - Can help save time when you want to process just some portion at beginning of file
  - sed '5q' fruits (print first 5 lines and quits)

Multiple Commands

- Separate instructions with a semicolon
  - sed 's/mango/aam/; s/banana/kela/;' fruits
- Precede each instruction by -e
  - sed -e 's/mango/aam/' -e 's/banana/kela/' fruits
- Order is important!
  - echo "please fix the light bulb!" | sed 's/light/tube/g; s/bulb/light/g'
  - echo "please fix the light bulb!" | sed 's/bulb/light/g; s/light/tube/g'

Sed Control Flow
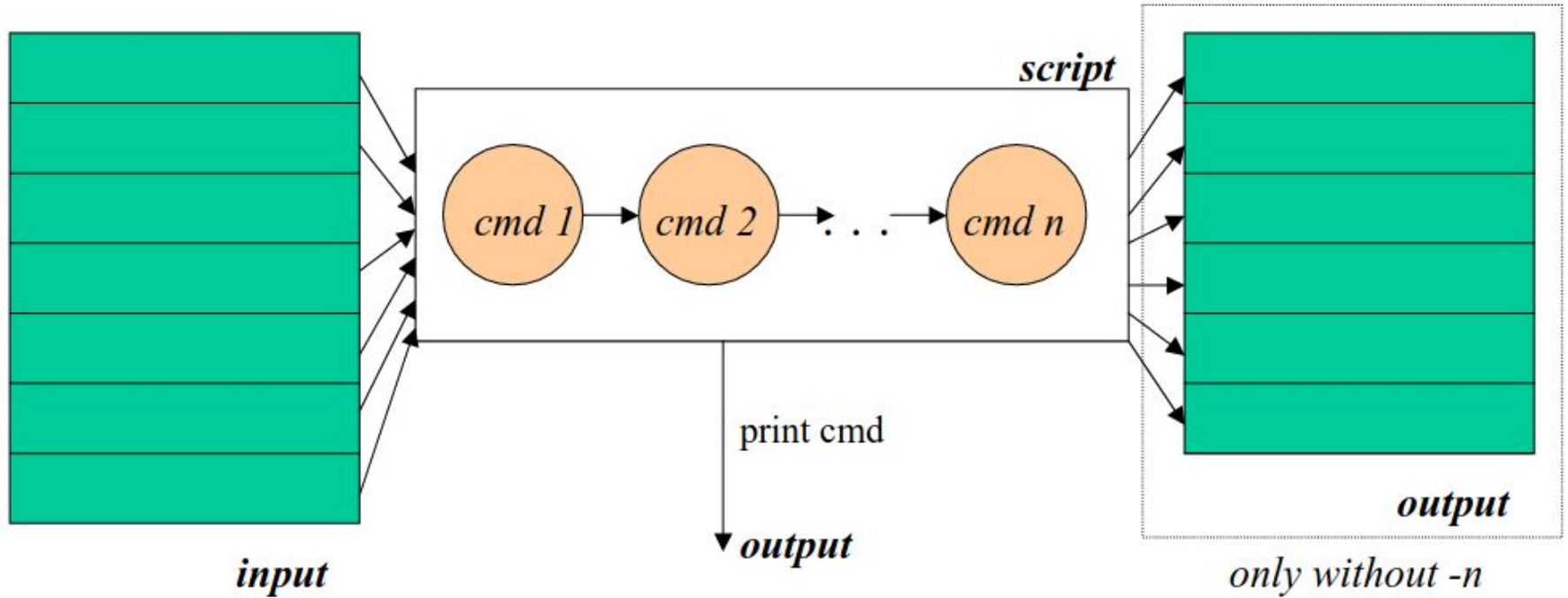
Backward References

- Can reference previously captured groups that match some regular expression pattern
  - Helps reuse parts of the pattern that you've already matched
- Represented using \1, \2, and so on, where the number corresponds to the order of the captured group
  - \1 refers to the first captured group, \2 refers to the second captured group, and so forth
- You can define a captured group using parentheses ( ) in the regular expression

- Example: ([A-Z])\1
  - ([A-Z]) is a group that captures any uppercase letter
  - \1 matches whatever was captured
  - echo "AA BB CC" | sed 's/\([A-Z]\)\1/XX/g' will result in XX XX XX
- & is used to reference the entire matched pattern
  - Example:
    - echo "I have 5 apples and 3 bananas." | sed 's/[0-9]\+/[&]/g'
    - Outputs: I have [5] apples and [3] bananas.

Script

- Not practical to enter many commands on the command line
- Create a script file that contains instruction and use -f option
  - sed -f script-file file

Drawbacks

- Not possible to go backward in the file
- No way to do forward references
- No facilities to manipulate numbers
- Cumbersome syntax

References

- https://www.gnu.org/software/sed/manual/sed.html
- https://linuxhint.com/50_sed_command_examples/

awk

- awk named after inventors: Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan.
- Like sed, stream-oriented and interprets a script of editing commands
- Unlike sed
  - Supports a programming language modeled on C Language
    - expressions, conditional statements, loops etc
  - 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

Structure

An awk program consists of:

- An optional BEGIN segment
  - To execute prior to reading input
- Pattern - action pairs
  - Processing input data
  - Action enforced in { }
- An optional END segment
  - To execute after end of input data

```
BEGIN {action}

pattern {action}

pattern {action}

    .

    .

    .

pattern { action}

END {action}
```

Simple Example

```
ls | awk '
BEGIN { print "List of jpg files:" }
/\.jpg$/ { print }
END { print "All done!" }
'
```

Records

Awk views each input line as a record
Default record separator is newline

Each word on that line is delimited by spaces or tabs or comma etc, as a field
$0 represents the entire input line
$1, $2, ... refer to the individual fields on the input line
Awk splits the input record before the script is applied

Built in Variables

NR: number of current records

RS: record separator

FS: field separator

OFS: output field separator

ORS: output record separator

Variables

- Variables need no declaration
  - variables take on numeric or string value based on context
  - By default, variables are initialized to the null string which has numerical value 0

Arithmetic

- Much better support than bash
- Examples
  - x = x + 1
  - y = y + $2 * $3
- Lot of built-in functions: sin, cos, atan, exp, int, log, rand, sqrt etc

Conditionals

- If
- If else
- If else if
- Use a script?
  - awk -f
    example.awk file

```
if (condition) {
  action-1
  action-2
  .
  .
}

if (condition) {
  action-1
  action-2
  .
  .
else
  action-a
  Action-b
  .
  .
}
```

Relation Operators

- x $<$ y True if x is less than y
- x $<=$ y True if x is less than or equal to y
- x $>$ y True if x is greater than y
- x $>=$ y True if x is greater than or equal to y
- x $==$ y True if x is equal to y
- x $!=$ y True if x is not equal to y
- x $\sim$ y True if the string x matches the regexp denoted by y
- x $!\sim$ y True if the string x does not match the regexp denoted by y

Other Operators

- = assignment operator
- == equality operator, returns TRUE if both sides are equal
- != inverse equality operator
- && logical AND
- || logical OR
- !   logical NOT

Loops

- Loops: for, while, do--while
- Break, continue and exit also possible

```
for (initialization; condition; increment/decrement)
    Action


while (condition)
    action


do
    action
while (condition)
```

Arrays

- Supports associative arrays i.e. the index need not be continuous or even numbers
  - Array index can be strings or numbers
    - E,g, arr[2]=6 or grade[ram]=AA
  - No need to declare the size of an array in advance
  - Supports one dimensional arrays

Built in functions

- Arithmetic
  - sin, cos, atan, exp, int, log, rand, sqrt
- String
  - length, substr
- Output
  - print, printf
- Special
  - system - executes a Unix command
  - system("clear") to clear the screen

sed+awk

- Can combine through pipe command

References

https://www.tutorialspoint.com/awk/index.htm