**UNX510/DPS918 - Unix BASH Shell Scripting**

**Lecture 5 - Regular Expressions; grep; sed; awk**

**Regular Expressions**

* many Unix utilities use regular expressions: grep, sed, awk, vi, perl, Tcl
* shell filename matches are **not** regular expressions (eg. \*.c)
* examples in this section will use the grep utility and the file [cars](http://czegel.com/seneca/unx510-dps918/lectures/cars.txt)
* regular expressions are used to search for or match text:
  + literal text can be used to search for that text
    - grep "chevy" cars   
      - notice that "Chevy nova ..." didn't get displayed, because of the uppercase 'C'
  + . matches any character (similar to ? wildcard)
    - grep ".c" cars   
      - notice that "chevy ..." didn't get displayed, there is no character before the 'c' to match the '.'
    - grep "5..." cars
  + [ ] called a character class, matches any character within the square brackets (similar to [ ] wildcard)
    - grep "[cC]hevy" cars
    - grep "[0-9][0-9][0-9][0-9][0-9]" cars
  + [^ ] matches any character **not** within the square brackets (similar to [! ] wildcard)
    - grep "[^f]ord" cars   
      - ^ only has this special meaning if it's the first character in the square brackets
  + ^ matches beginning of line
    - grep "^f" cars   
      - ^ only has this special meaning if it's the first character in the regular expression
  + $ matches end of line
    - grep " [0-9][0-9][0-9]$" cars   
      - $ only has this special meaning if it's the last character in the regular expression
  + \* following any character denotes **zero or more** occurrences of that character
    - grep "ford.\*83" cars   
      - . means any one single character, so .\* means any number of any characters
    - grep "^[^ ][^ ]\*  \*[^ ][^ ]\*  \*65" cars   
      - this will match '65' only in the third field
  + \ inhibits meaning of special characters
    - grep ' [0-9][0-9][0-9]\$' cars   
      - of course, there are no records ending with a $ sign
* regular expressions may or may not need delimiters - varies from program to program
  + eg. grep and egrep don't use delimiters,sed and awk use delimiters, usually / (forward slash)

**Extended Regular Expressions**

* extended regular expressions are not recognized directly by grep, can use egrep or grep -E:
  + {num} following any character matches "num" occurrences of that character
    - egrep "[0-9]{5}" cars
  + {min, max} following any character matches "min" to "max" occurrences of that character - "max" is optional
    - egrep " [0-9]{3,4}$" cars
  + + following any character denotes one or more occurrences of that character - same as {1,}
    - egrep "^[^ ]+ +[^ ]+ +65" cars
  + ? following any character denotes zero or one occurrence of that character - same as {0, 1}
    - egrep "ch?e" cars
  + ( reg-exp ) parentheses used for grouping
    - egrep "^([^ ]+ +){2}65" cars
  + | means OR, matches reg-exp on either side of the vertical bar
    - egrep "ford|chevy" cars
    - egrep "(ford|chevy) +[^ ]+ +65" cars
* extended regular expression characters may or may not need to be escaped - varies from program to program
  + { } + ? ( ) |   need to be escaped for use with "grep" and "sed"
  + { } + ? ( ) |   do not need to be escaped for use with "grep -E", "egrep", "sed -r", and "awk"
* for example, the following statements have identical output:
  + grep "ford\|chevy" cars
  + egrep "ford|chevy" cars
  + grep -E "ford|chevy" cars
  + sed -n "/ford\|chevy/ p" cars
  + sed -nr "/ford|chevy/ p" cars
  + awk "/ford|chevy/" cars
* other examples of regular expressions   
     (Mr|Mrs) Smith   - match either "Mr Smith" or "Mrs Smith"   
     Mrs? Smith       - match either "Mr Smith" or "Mrs Smith"   
     [a-zA-Z]+        - match one or more letters   
     ^[a-zA-Z]\*$      - match lines with only letters   
     [^0-9]+          - match string not containing digits   
     [+-]?([0-9]+[.]?[0-9]\*|[.][0-9]+)([eE][+-]?[0-9]+)?        - match valid "C" programming numbers

**grep**

* uses regular expression for pattern, eg. **grep** 'reg-exp' filename, then prints matched lines
* gives 0 exit status if pattern matched
* options:
  + -c - counts matched lines instead of printing them
  + -i - ignores case
  + -n - precedes each line with a line number
  + -v - reverses sense of test, eg. finds lines not matching pattern
* examples, using the file [cars](http://czegel.com/seneca/unx510-dps918/lectures/cars.txt)
  + grep 'chevy' cars        - display only lines containing the string "chevy"
  + grep -c 'chevy' cars     - display count of lines containing the string "chevy"
  + grep -i 'chevy' cars     - display only lines containing the string "chevy", ignoring case
  + grep -ic 'chevy' cars    - display count of lines containing the string "chevy", ignoring case
  + grep -v 'chevy' cars     - display only lines **not** containing the string "chevy"
  + grep -ivc 'chevy' cars   - display count of lines **not** containing the string "chevy", ignoring case
  + grep -n 'chevy' cars     - display only lines containing the string "chevy", with line numbers

**sed**

* stream editor
  + sed 'address instruction' filename
  + checks for address match, one line at a time, and performs instruction if address matched
  + prints lines to standard output by default (supressed by -n option)
* addresses
  + can use a line number, to select a specific line (for example: 5)
  + can specify a range of line numbers (for example: 5,7)
  + can specify a regular expression to select all lines that match
  + default address (if none is specified) will match every line
* instructions
  + p - print line(s) that match the address (usually used with -n option)
  + d - delete line(s) that match the address
  + q - quit processing at the first line that matches the address
  + s - substitute text to replace a matched regular expression, similar to vi substitution
* examples, using the file [cars](http://czegel.com/seneca/unx510-dps918/lectures/cars.txt)
  + sed '3,6 p' cars                         - display lines 3 through 6 (these lines will be doubled, since all lines printed by default)
  + sed -n '3,6 p' cars                       - display only lines 3 through 6
  + sed '5 d' cars                            - display all lines except the 5th
  + sed '5,8 d' cars                          - display all lines except the 5th through 8th
  + sed '5 q' cars                            - display first 5 lines then quit, same as head -5 cars
  + sed -n '/chevy/ p' cars                   - display only lines matching regular expression, same as grep 'chevy' cars
  + sed '/chevy/ d' cars                      - delete all matching lines, same as grep -v 'chevy' cars
  + sed '/chevy/ q' cars                      - display to first line matching regular expression
  + sed 's/chevy/gm   /' cars                 - substitute "chevy" with "gm   " on each matching line
  + sed 's/[0-9]/\*/' cars                     - substitute first occurrence of a digit on each line with an asterisk
  + sed 's/[0-9]/\*/g' cars                    - substitute every occurrence of a digit on each line with an asterisk
  + sed -r '5,8 s/[0-9]+/\*\*\*/' cars           - substitute only on lines 5 to 8
  + sed -nr '5,8 s/[0-9]+/\*\*\*/ p' cars        - substitute only on lines 5 to 8, print only matched lines
  + sed -r '/ford/ s/[0-9]+/\*\*\*/' cars        - substitute only on lines containing "ford"
  + sed -nr '/ford/ s/[0-9]+/\*\*\*/ p' cars     - substitute only on lines containing "ford", print only matched lines
  + sed -r 's/[0-9]+/\*\*\* & \*\*\*/' cars         - & is the value of the string matched by reg-exp
  + sed -r 's/[^0-9]\*([0-9]+).\*/The first number is \1/' cars     - \1 is the string matched within the first group
  + sed -r 's/([^ ]+ +)([^ ]+ +)/\2\1/' cars  - swap first two fields
  + sed -r 's/([^ ]+ +)([^ ]+ +)([^ ]+ +)([^ ]+ +)([^ ]+)/We have a \1 \2 at only $\5/' cars
  + sed -nr '/ford/ s/[^ ]+ +([^ ]+) +[^ ]+ +[^ ]+ +([^ ]+)/We have an amazing \1 for the low price of $\2! What a steal!/ p' cars
  + sed -nr "/$1/"' s/[^ ]+ +([^ ]+) +[^ ]+ +[^ ]+ +([^ ]+)/We have an amazing \1 for the low price of $\2! What a steal!/ p' cars
* when using multiple commands, the following statements have identical output:
  + sed 's/ford/Ford/' cars | sed 's/chevy/Chevy/'
  + sed -e 's/ford/Ford/' -e 's/chevy/Chevy/' cars
  + sed 's/ford/Ford/; s/chevy/Chevy/' cars

**awk**

* pattern matching and processing
  + awk 'pattern {action}' filename
  + checks for pattern match,one line at a time, and performs action if pattern matched
* variables
  + NR is an awk variable meaning the line number of the current record
  + NF is an awk variable meaning the number of fields in the current record
  + $n are awk variables, meaning the value of the nth field (field delimiter is space or tab)
  + $0 is the entire record
  + IFS is an awk variable specifying the input field separator, defaults to space or tab
    - can also be specified using the -F option
  + OFS is an awk variable specifying the output field separator, defaults to a space
  + user-defined variables don't need to be declared, any unquoted string is assumed to be a variable
  + variables are automatically initialized, 0 if used numerically, null if used as a string
  + numeric comparison will be done if both sides are numeric (eg. $3 > 65), otherwise string comparison will be done
* pattern
  + can use a line number to select a specific line, by comparing it to NR (for example: NR == 2)
  + can specify a range of line numbers (for example: NR == 2, NR == 4)
  + can specify a regular expression, to select all lines that match
  + can compare field values to literals or variables (for example: $3 == 65)
  + can check for a regular expression match within a field by using the ~ operator (for example: $2 ~ /[0-9]/)
  + BEGIN is a special pattern that causes execution of the action before any records have been read
    - usually used to initialize variables, print header lines for reports, etc.
  + END is a special pattern that causes execution of the action after all records have been read
    - usually used to calculate totals and averages, print summary lines for reports, etc.
  + every line is selected if no pattern is specified
* instructions
  + print - print line(s) that match the pattern, or print fields within matching lines
  + print is default action if no action is specified
  + printf - a C-like version of print, with similar format specifications and no automatic new-line
  + there are many, many instruction, including just about all C statements with similar syntax
* examples, using the file [cars](http://czegel.com/seneca/unx510-dps918/lectures/cars.txt)
  + awk 'NR == 2, NR == 4' cars                     - display the 2nd through 4th lines (default action is to print entire line)
  + awk '/chevy/' cars                              - display only lines matching regular expression, same as grep 'chevy' cars
  + awk '{print $3}' cars                           - display third field of all lines
  + awk '{print $3 $1}' cars                        - display third and first field of all lines
  + awk '{print $3, $1}' cars                       - includes an output field separator (variable OFS) because of the comma
  + awk -F':' '{print $6}' /etc/passwd              - specifies that : is input field separator, default is space or tab
  + awk '/chevy/ {print $3, $1}' cars               - display third and first field of lines matching regular expression
  + awk '$3 == 65' cars                             - display only lines with a third field value of 65
  + awk '$5 <= 3000' cars                           - display only lines with a fifth field value that is less than or equal to 3000
  + awk '$5 <='$price' {print $1, $2, $5}' cars     - $price is a shell variable, not an awk variable, e.g. first execute: price=3000
  + awk '$5 > 3000 && $5 < 9000' cars               - display lines where 5th field is in range 3000 to 9000
  + awk '$5 < 3000 || $5 > 9000' cars               - display lines where 5th field is outside of range 3000 to 9000
  + awk '$2 ~ /[0-9]/' cars                         - searches for reg-exp (a digit) only in the second field
  + awk '{printf "%-30s%20s\n", $5, $1}' cars       - display 5th field left-justified in a 30 character field, 1st field right-justified in a 20 character field
  + awk '$5 >='$price' {$5 = $5 \* 0.9} {print}' cars - if field 5 >= shell variable $price then reduce field 5 by 10%, e.g. first execute: price=$5000
  + awk '$3 < "8"' cars                              - double quotes force string comparison
  + awk '$1 < "honda"' cars                - double quotes force string comparison, otherwise "honda" would be treated as a variable
  + awk 'NF != 5' cars                               - display lines without 5 fields
  + awk 'BEGIN {OFS="~"} {print $1, $2}' cars        - display 1st and 2nd field of each record, separated by ~
  + awk '{OFS="~"; print $1, $2}' cars               - same result, but much less efficient
  + awk '{printf "%-8s%-8s%-8s%-8s%-8s\n", $2, $1, $3, $4, $5}' cars            - swap first two fields
  + awk '{printf "We have a %-8s %-8s at only $%s\n", $1, $2, $5}' cars
  + awk '/ford/ {print "We have an amazing " $2 " for the low price of $" $5 "! What a steal!"}' cars
  + awk "/$1/"' {print "We have an amazing " $2 " for the low price of $" $5 "! What a steal!"}' cars