# UNX510/DPS918 - Unix BASH Shell Scripting

## Lecture 6 - More On Regular Expressions; Pathname Expansion; Named Character Classes

### More On Regular Expressions

### awk Arithmetic

* examples, using the file [cars](http://czegel.com/seneca/unx510-dps918/lectures/cars.txt)
  + awk '{inventory+=$5} END {print "Our total inventory has a value of $" inventory}' cars
  + awk '{inventory+=$5} END {print "The average cost of a car on our lot is $" inventory/NR}' cars
  + note that awk handles decimal arithmetic and format specifications, with a default of 2 decimal places:   
    awk '{inventory+=$5} END {printf "The average cost of a car on our lot is $%10.3f\n", inventory/NR}' cars
  + awk '$5 > price {price=$5} END {print "Our most expensive car has a price of $" price}' cars
  + awk '$5 <='$price' {quantity++} END {print "We have " quantity " cars under $"'$price'}' cars
  + sort -rnk5 cars | awk 'NR==1 {price=$5} price==$5'    - display all records with maximum 5th field
  + awk '$5 > price {price=$5} END {print price}' cars | xargs -Ixxx awk '$5 == xxx' cars
  + awk '$5 == '$(awk '$5 > price {price=$5} END {print price}' cars) cars

### Comparing sed And awk

* equivalent examples, using the file [cars](http://czegel.com/seneca/unx510-dps918/lectures/cars.txt)
  + sed -r 's/([^ ]+ +)([^ ]+ +)/\2\1/' cars  - swap first two fields
  + awk '{printf "%-8s%-8s%-8s%-8s%-8s\n", $2, $1, $3, $4, $5}' cars
  + sed -r 's/([^ ]+ +)([^ ]+ +)([^ ]+ +)([^ ]+ +)([^ ]+)/We have a \1 \2 at only $\5/' cars
  + awk '{printf "We have a %-8s %-8s at only $%s\n", $1, $2, $5}' cars
  + sed -nr '/ford/ s/[^ ]+ +([^ ]+) +[^ ]+ +[^ ]+ +([^ ]+)/We have an amazing \1 for the low price of $\2! What a steal!/ p' cars
  + awk '/ford/ {print "We have an amazing " $2 " for the low price of $" $5 "! What a steal!"}' cars
  + sed -nr "/$1/"' s/[^ ]+ +([^ ]+) +[^ ]+ +[^ ]+ +([^ ]+)/We have an amazing \1 for the low price of $\2! What a steal!/ p' cars
  + awk "/$1/"' {print "We have an amazing " $2 " for the low price of $" $5 "! What a steal!"}' cars
  + sed -r 's/([^ ]+) \*([^ ]+) \*([^ ]+).\*/\3 \2 \1/' cars       - display 3rd, 2nd, and 1st fields
  + awk '{print $3, $2, $1}' cars
  + sed 's/a/A/g' cars                                          - capitalize all letter a's
  + awk '{ for (i = 1; i <= length($0); i++) {   
              c = substr($0, i, 1);   
              if (c == "a")   
                 printf("A");         # or: printf "A";   
              else   
                 printf("%c", c)      # or: printf c   
           }   
           printf "\n"   
         }' cars
* awk is better for field manipulation and arithmetic, sed is better for character manipulation and editing

### Pathname Expansion

* also called globbing, ambiguous file references, metacharacters, wild card characters, and filename generation characters
* used to find filenames that match a pattern
* globbing is performed by the shell, not by commands, so globbing may be used with any command
* globbing does not match a dot at the beginning of a filename (hidden file) or a slash (directory level), by default
* if a glob doesn't match a filename it remains unchanged, by default
* run these commands to try the examples in this section:   
  mkdir testdir   
  cd testdir   
  touch .file .file{1..10} .pic{1..5}.gif file file{1..100} pic.jpg pic{1..20}.jpg pic.gif pic{21..40}.gif pic.jpeg pic{41..60}.jpeg video{1..40}.mpeg pic27 pic38
* ? matches any single character   
  ls file?2   
  ls pic??.gif
* \* matches any number of characters, including none   
  ls file\*   
  ls \*10\*
* a leading period (hidden file) must be explicitly specified   
  ls .file\*   
  ls .\*10\*
* [ ] matches any single character in included list   
  ls file[135]   
  ls file[135][123]
* - within [ ] between two characters represents a range   
  ls file[0-47-9]   
  ls p\*[1-3]\*[d-g]
* if ! is first character within [ ], then any character not in list is matched   
  ls file[!0-47-9]   
  ls p\*[1-3]\*[!d-g]

### Globbing Shell Options (bash only)

* shell options may be set using shopt -s and unset using shopt -u
* without the -s or -u options, shopt will show if the option is on or off
* nullglob - non-matching globs are removed, instead of preserved   
  echo [0-9]   
  shopt -s nullglob   
  echo [0-9]
* failglob - non-matching globs cause an error, command is not executed   
  echo [0-9]   
  shopt -s failglob   
  echo [0-9]
* nocaseglob - matches are done ignoring case   
  touch FiLe25   
  echo file\*5   
  shopt -s nocaseglob   
  echo file\*5
* dotglob - wildcards will match hidden filenames   
  echo \*5   
  shopt -s dotglob   
  echo \*5

### Extended Globbing (bash only)

* extended globbing may be enabled via a shell option: shopt -s extglob, but is on by default
* a pattern-list is a list of items separated by a vertical bar
* ?(pattern-list) - matches zero or one occurrence of the given patterns   
  ls pic\*.jp?(e)g   
  ls file4?(3|5)   
  echo pic?([0-9]).\*
* \*(pattern-list) - matches zero or more occurrences of the given patterns   
  ls pic\*(3).\*   
  ls file\*(1|3|5)
* +(pattern-list) - matches one or more occurrences of the given patterns   
  ls pic+(3).\*   
  ls file+(1|3|5)
* @(pattern-list) - matches one of the given patterns   
  ls pic\*@(jpg|gif)   
  ls pic@(1|2|33|66).\*
* !(pattern-list) - matches anything except one of the given patterns   
  ls pic!(\*jpg|\*gif)   
  ls pic\*!(jpg|gif)   
  - does NOT work, because "!(jpg|gif)" matches a null at the end of the matched string

### Named Character Classes

* named character classes are useful, ensuring that collating sequences are correct regardless of the locale
* [:alnum:] - alphanumeric - same as [:alpha:] and [:digit:]
* [:alpha:] - alphabetic - same as [:lower:] and [:upper:]
* [:blank:] - spaces and tabs
* [:cntrl:] - control characters
* [:digit:] - digits 0 to 9
* [:graph:] - alphanumerics and punctuation - same as [:alnum:] and [:punct:]
* [:lower:] - lower-case alphabetic
* [:print:] - printable characters - same as [:alnum:], [:punct:], and spaces
* [:punct:] - punctuation - eg. ! " # $ % & ' ( ) \* + , - . / : ; < = > ? @ [ \ ] ^ \_ ` { | } ~
* [:space:] - space characters - eg. tab, newline, vertical tab, form feed, carriage return, and space
* [:upper:] - upper-case alphabetic
* [:xdigit:] - hex digits - 0 to 9, a to f, A to F
* can be used with "tr" command:   
  tr "[:lower:]" "[:upper:]" < cars
* can be used within regular expressions, including within the "[[ ... ]]" structure (must be enclosed within a second set of square brackets):   
  echo $1 | grep "^[[:digit:]]\*$" >/dev/null || { echo "First argument must be numeric" >&2; exit 2; }   
  echo $1 | grep "[^[:digit:]]" >/dev/null && { echo "First argument must be numeric" >&2; exit 2; } || exit 0   
  [[ $1 =~ [^[:digit:]] ]] && { echo "First argument must be numeric" >&2; exit 2; } || exit 0
* can be used within globs, including within the "[[ ... ]]" structure, extended globbing does NOT need to be enabled (must be enclosed within a second set of square brackets):   
  ls pic[[:digit:]].\*   
  [[ $1 = \*[^[:digit:]]\* ]] && { echo "First argument must be numeric" >&2; exit 2; } || exit 0