CSE2031 Software Tools - UNIX introduction

Summer 2010

Przemyslaw Pawluk

Department of Computer Science and Engineering
York University
Toronto

June 29, 2010

Notes

Table of contents

Notes

The AWK Programming Language

- AWK (pron. auk) can be used to manipulate text and numerical values.
- Usually, simple short programs (could be just one line).
- The program could be in a file, or could be entered with the command
- The name AWK is derived from the family names of its authors Alfred Aho, Peter Weinberger, and Brian Kernighan
- Consider the following example

Notes

AWK structure

- The structure of an AWK program
- Each AWK program is a sequence of one or more pattern-action statement
- Searches the input file looking for any lines that are matched by any of the patterns and the action is applied

Notes

Shell built-in variables

- $# The number of arguments
- $* All arguments to shell
- $- Options supplied to shell
- $? return value of the last command executed
- $$ process ID of the shell
- $ process ID of the last command started with &

Notes

Shell pattern Matching Rules

- `*` Any string, including the null string
- `?` Any single character
- `[ccc]` Any of the characters in ccc `[a-d0-3]` is equivalent to `[abcd0123]`
- `"..."` Matches exactly, the quotes are to protect special characters
- `\c` c literally; if `\*` it matches the "*" char
- `a|b` In case expression only, matches a or b

Notes

Example

You have a file called in.txt that contains list of cars with distances and rates. We want to print all cars with distance grater than 0.

Notes

Structure of the program

- Sequence of one or more pattern-action statements
- Search the input file looking for any lines that are matched by any of the patterns
- if found action is applied
- if there is no pattern action is executed on every line
- expression separated by commas in print are separated by single space when printed
- You can use printf function as in C

```
pattern {action}
pattern {action}
```

Notes

How to run a program?

```
awk 'program' file1 file2
```

Program in file

```
awk -f progfile file1 file2
```

Notes

Combination of Patterns

- patterns can be combined with `&& (AND), ||(OR) or !(NOT)`
- `/name/` matches with name in the line

```
NF != 3 {print $0, "Number of fields is not 3"}
$2 <8.75 || $2>100 {print %0, "Overflow"}
$2 >20 {print $0, "rate more than $20 dollars"}
!$3 >0 {print $0, "Negative rate"}
```

Notes

Begin and End

- The special pattern BEGIN matches before the first line of the first input file
- The special pattern END matches after the last line of the last input file

Notes

Example

```
BEGIN{print "NAME RATE HOURS"; print}
{print}
{total = total + $2 * $3}
END{print "The total is ", total}
```

Notes

String manipulation

You can use functions such as length(str) to get length of the string str

```
{ names = names $1 "␣"}
END {print names}
{last = $0
END {print last}
{ nc = nc +length($0) +1
nw = nw + NF}
END {print NR, "lines␣", nw,
"words ,␣nc,␣"characters }
```

Notes

Control flow

- You can use `if-else` and `while` statements as in C

```
$2 > 6 {n=n+1; rate = rate + $2 * $3}
END { if(n > 0)
print n, cars, total rate is , rate,
 average rate is  , rate/n
else
print No cars are making more than $6
}
{ i=1
while (i <= $3) {
printf( \t%.2f\n , $1 *(1 + $2) ^i)
i=i+1
}
}
```

Notes

Arrays

AWK allows for arrays

- The index of the arrays need not be integer.
- No need for declaration
- Initialized to 0 or `""`
- For example, you can say `Ar1[$1] = $2`

```
# print the input in a reverse order
{line[NR] = $0}
END { i=NR
while(i > 0) {
print line[i]
i=i-1
}
}
```

Notes

Arrays

```
{ar[$1]=$2}
END {
for (x in ar) print x, ar[x]
}
```

The order of stepping in the array is implementation dependent!

Notes

Patterns

- Rule = Pattern+Action
- BEGIN statement is executed befor any input is read
- END statement is executed after all inputs are read
- Pattern statement is executed when Pattern is true (satisfied)
- /regular expression/ statement is executed when line contains string that matches expression
- Compound pattern statement is executed at any line that satisfies pattern
- pattern1, pattern2 statement – is a range pattern that matches each line from line matched by pattern1 to the next line mached by pattern2

Notes

Matching Strings

- `/regexpr/` matches when the current input line contains a substring matched by regexpr
- Expression `~/regexpr/` Matches if the string value of the expression contains a substring matched by regespr.
- Expression `!~/regexpr/` matches if the string value of expression does not contain a substring matched by regexpr

```
/Asia/ # short hand for $0 ~ /Asia/
$4 ~ /Asia/
$3 !~ /Asia/
```

Notes

Regular expressions

- A non metacharacter that matches itself A, b, D,
- Escape sequence that matches a special symbol `\t, \*`
- ^ beginning of a string
- $ End of a string
- . Any single character
- `[ABC]` matches any of A,B,C
- `[A-Za-z]` matches any character
- `[^0-9]` any character except a digit

Notes

Regular expressions - combinations

- Alternation: `A|B` matches A or B
- Concatenation: `AB` matches A followed by B
- Closure: `A*` matches zero or more A
- Positive closure `A+` matches 1 or more A
- Zero or one: `A?` matches the null string or A
- Parenthesis: `(r)` matches the same string as r

Notes

Regular expressions - string matching

- `^C` matches C at the beginning of a string
- `C$` matches C at the end of a string
- `^C$` matches the string consists of the single character C
- `^.$` any string with exactly one character
- `...` matches any three consecutive characters
- `\.$` matches a string that ends with period
- `^[ABC]` A, B, or C at the beginning of a string
- `^[^ABC]` any character at the beginning of a string except A,B, or C
- `[^ABC]` any character other than A,B, or C
- `^[^a-z]$` any single character string except a lower case character

Notes

Built-in variables

- ARGC Number of command lines arguments
- ARGV arra of command line arguments
- FILENAME Name of current input file
- FNR Record number in current file
- FS Input field separator
- NF Number of field in the current record
- NR Number of records red so far
- OFS Output field separator
- ORS Output record separatot
- RLENGTH Length of string matched by matching function
- RS Input record separator

Notes

Reading from a File

- getline function can be used to read input from a file, splits the record and sets NF, NR, and FNR
- It returns 1 if there was a record, 0 for end of file, and -1 for error
- `getline < "File"`
- `getline x <"File"` gets the next line and stores it in x (no splitting) NF, NR, and FNR not modified

Notes

Notes