COS 126 Lecture 14: Pattern Matching

Essential and characteristic UNIX tools

grep
awk
sed
more
emacs

Learn to use these!

References:

UNIX Programming Environment
(Kernighan and Pike)

AWK Programming Language
(Aho, Weinberger, and Kernighan)

"man"

Not C programming,
though the tools are (as?/more?) powerful

Useful for variety of practical applications

Directly related to
fundamental tenets of computer science

Context for second half of COS126

Most of us "know"

- how to program in C (but need practice)
- how machine works (but want to know more)

C programming

- deemphasize (specific knowledge in manual)
- reemphasize (context for advanced concepts)
- build efficient programs and systems

Learn to use and exploit new systems

- get small examples working
- go to the documentation
- understand limitations

Appreciate other programming languages

- ex: grep, awk, sh, perl, c++, java, maple
- differ from C (and each other) in "details"
- applications-oriented abstract mechanisms
- cult following

Fundamental limits on computation

Systems programs and programming environments

Applications

General questions

that come up when facing a new problem to
solve using a computer:

What primitive objects are important?
    Ex. numbers, files, pictures, text, programs
    [Could always do it in C.]
    [Does another tool allow direct manipulation?]
How long will it take me to do this task?
    [Depends on what tool I use.]
Have I done something like this before?
    [If so, maybe I should use the same tool.]
    [Maybe I have some code laying around.]
    [Does it still work?]
Will I be doing something like this again?
    [If not, quick hack may be OK.]
Will I be doing something like this *frequently*?
    [Worthwhile to learn a new tool?]
    [Worthwhile to *create* a new tool?]
Has *someone else* done something like this?
    [May be some code laying around.]
Will *someone else* be doing this in the future?
    [Document the code?]
    [Make it portable?]

No easy answers: need to consider alternatives
with an *open mind*

grep

general regular expression pattern matching
    filter
    stdout gets only those lines from stdin
        that "match" argument string

Elementary examples:

Does a file contain a string?

```
grep Smith classlist
```

Which file contains a string?

```
grep grep *.sl
```

Just give me the data of interest...

```
a.out | grep -v DEBUG
```

Dictionary

Crossword puzzle or Scrabble too time consuming?

usr/dict/words is list of words in dictionary
    25,486 words

Grep and similar tools can be effective in
    ``finding'' words

grep hh /usr/dict/words

```
beachhead
highhanded
withheld
withhold
```

grep .a.a.a   usr/dict/words | wc

```
34
```

grep .u.u.u   usr/dict/words

```
cumulus
```

Ex: Do spell checking by specifying what you know

egrep 'n(ie|ei)ther' /usr/dict/words

"man grep" excerpts

Name

    grep - search file for regular expression

Syntax

    grep [option...] expression [file...]

Description

    Commands of the grep family search the
    input files (standard input default) for
    lines matching a pattern. Normally, each
    line found is copied to the standard output.

    Take care when using the characters
    $ * [^ | ( ) and in the expression because
    they are also meaningful to the Shell. It
    is safest to enclose the entire expression
    argument in single quotes ' '.

Options

    -c Produces count of matching lines only.
    -i Considers upper and lowercase letter identical
    -n Precedes each matching line with its line number.
    -v Displays all lines that do not match.

Restrictions

    Lines are limited to 256 chars;
        longer lines are truncated.

See Also

    ex(1), sed(1), sh(1)

Regular Expressions

conventions for grep:

```
c        any non-special char matches itself
^        beginning of line
$        end of line
.        any single character
[...]    any character in [a-z]
[^...]   any character not in [a-z]
r*       zero or more occurrences of r
```

"extended" regular expressions (egrep)

```
(r)
r1 | r2
```

specific technical meaning in theoretical CS

[stay tuned, next lecture]

More Examples

Find all references to Java

```
grep 'java|Java' text
grep '[Jj]ava' text
```

Find all big files

```
du -a | egrep "^[1-9][0-9][0-9][0-9]"
```

Find all lines with dollar amounts on them

```
egrep '[$][0-9]+.[0-9]*' myfile
```

BUG: matches $7A46.

Ex: fix this bug

Matches involving special chars can be complex

Ex: excerpt from "man grep":

```
grep  -E  '\(  *([a-zA-Z]*|[0-9]*)  *\)'  my.txt
```

This command displays lines in my.txt such as
(783902) or (y), but not (alpha19c).

Note that with grep -E, \(and \) match parens in the text and (and) are special chars that group parts of the pattern. With grep without the -E flag, the reverse is true; use (and) to match parens and \(and \) to group chars.

Can't "escape" this problem!

Pattern Matching alternatives in UNIX

grep
egrep: extended regular expressions
fgrep: search for multiple patterns
egrep, fgrep are obsolescent
     [try "man grep", "webster obsolescent"]

more: (Try it!)

Substitution (editing), not just matching

emacs, ex (various ways)
     interactive

sed
     filter
     line-by-line editing

sed 's/apples/oranges/g' file

awk: Pattern matching "language"
     matching
     substitution
     pattern manipulation
     variables
     numeric capabilities
     control and logic

AWK

Ref:

"man AWK"

The AWK Programming Language

Aho, Kernighan, and Weinberger

Simple language

- string and data manipulation
- formatting
- selecting

Based on pattern matching

Expressive language, short programs

General-purpose programmable tool

Rapid prototyping

"Automatic"

- input
- field-splitting
- storage management
- declarations, initialization

Running awk:

```
% awk 'program' inputfile
% command | awk 'program' | command
% awk -f program inputfile
```

Patterns and Actions

AWK program: sequence of pattern-action pairs

"patterns"

- regular expression
- conditional expression

"actions"

- print
- arithmetic statement
- [full C-like program]

"variables"

- input lines ("records") are divided into "fields"
    - delimited by space char (change with -F)
- built-in variables
    - NF -- number of fields in current record
    - NR -- number of records seen so far
    - $0 -- current record
    - $i -- ith field in current record
- BEGIN -- matches first line
- END   -- matches last line

No explicit input

Basic AWK cycle

- for every input line
    - if pattern matches, perform action

Sample AWK programs

Grep

```
awk '/pattern/' file
```

Line Count

```
awk 'END {print NR}' file
```

Rearrange fields

```
awk '{ print $2, $1 }' file
```

Print long lines

```
awk 'length > 72' file
```

Add up first column, print sum and average

```
awk '{ s += $1 } END { print s, s/NR }' file
```

Compute COS126 midterm grades

```
awk '
{printf("%8s ", $1);}
{p = $2+$3+$4+$5; q = $6;}
{printf("%4d ", 100*(16*p/40.0 + 20*q/65.0)/36.0);}
{printf("%3d%3d%3d%3d%3d ",  $2, $3, $4, $5, $6);}
' scores | sort -r +2 | lpr
```