CS2042 - Unix Tools

Fall 2010

Lecture 4

Hussam Abu-Libdeh

September 17, 2010

Counting

- How many lines of code are in my new awesome program?
- How many words are in this document?
- Good for bragging rights

A trip to the `wc` .. (get it? :-p)

- `wc -l` : count the number of lines
- `wc -w` : count the number of words
- `wc -m` : count the number of characters
- `wc -c` : count the number of bytes

Looking for things

- `find` : Searching for files/directories by name or attributes
- `grep` : Search contents of files

`find`

- used to locate files or directories
- search any set of directories for files that match a criteria
- search by name, owner, group, type, permissions, last modification date, and other criteria
- search is recursive (will search all subdirectories too)

Syntax looks like this:

```
find [where to look] criteria [what to do]
```

Simple usage

- display pathnames of all files in current directory and subdirectories

  ```
  find .  -print
  find -print
  find .
  ```

  (all equivalent)

- search for a file by name

  ```
  find .  -name my_awesome_file.txt
  ```

Find options

- `-name` : name of file or directory to look for
- `-maxdepth num` : descend at most *num* levels of directories while searching
- `-mindepth num` : descend at least *num* levels of directories while searching
- `-amin n` : file last access was *n* minutes ago
- `-atime n` : file last access was *n* days ago
- `-group name` : file belongs to group *name*
- `-path pattern` : file name matches shell pattern *pattern*
- `-perm mode` : file permission bits are set to *mode*

... for more: `man find`

More on find

- normally all modifiers for `find` are evaluated in conjunction (i.e. AND). We can find files matching a pattern *OR* another by using the `-o` flag.
- can execute a command on found files by using the `-exec command '{}' +` flag.

Find examples

Find all files accessed at most 10 minutes ago

```
find .  -amin -10
```

Find all files accessed at least 10 minutes ago

```
find .  -amin +10
```

Display all the contents of files accessed in the last 10 minutes

```
find .  -amin -10 -exec cat '{}' +
```

grep

The purpose of grep is to print the lines that match a particular pattern.

grep

grep <string> [file]

- searches file for all lines containing <string>
- grep stands for global / regular expression / print

Examples:

grep password file

- prints all lines that contain the word password in the file file.

What lines contain the word monster in Frankenstein?

grep 'monster' Frankenstein.txt

More Simple Examples

Two simple ways to use grep are on a file and on piped input:

grep on a file

```
grep "chromium" /var/log/dpkg.log
```

- Shows when I have updated chromium-browser

grep piped input

```
history | grep grep
```

- When have I used grep recently?

Grep options

- `grep -i` - ignores case
- `grep -A 20 -B 10` - prints the 10 lines before and 20 lines after each match
- `grep -v` - inverts the match
- `grep -o` - shows only the matched substring
- `grep -n` - displays the line number

Example:

```
grep -v # bashscript
```

- Prints all noncommented lines

Regular Expression

`grep` like many programs takes in a **regular expression** as its input. Pattern matching with regular expressions is more sophisticated than shell expansion and also uses different syntax.

More precisely, a regular expression is a set of strings; these strings **match** the expression.

When we use regular expressions, it is (usually) best to enclose them in single quotes to stop the shell from expanding it before passing it to `grep` or other tools.

Regular Expression Notes:

Regular Expressions are used all over the place. We've already seen `grep`, which takes RegExp search strings. Later we'll see some other fun commands which will use them.

- search documents in emacs/vi
- write scripts in Perl/Python/Ruby...

Unfortunetly, Regular Expressions use different syntax than shell expansion (sorry!)

Regular Expression Rules

Some RegExp patterns perform the same tasks as our earlier wildcards

Single Characters

Wild card: ? RegExp: .

- Matches any single character

Wild card: [a-z] RegExp: [a-z]

- Matches one of the indicated characters
- Don't separate multiple characters with commas in RegExp form (e.x. [a,b,q-v] becomes [abq-v]).

Example:

`grep 't.a'` - prints lines with things like tea, taa, and steap

A Note On Ranges

Like shell wildcards, RegExps are case-sensitive. What if you want to match any letter, regardless of case?

- What will [a-Z] match?

Character Sorting

Different types of programs sort characters differently. In the C language, characters A-Z are assigned numbers from 65-90, while a-z are 97-122. Thus, the range [a-Z] would equate to [122-65]. Though this is bad enough, there are non-alphabet characters within that range. To specify all letters safely we would use [a-zA-Z].

- Note: not everything treats sorting like C. For example, a dictionary program might sort its characters aAbBcC...

Fortunately We Can Get Around This Easily

Fortunately there are shortcuts for many ranges of characters we typically run into:

POSIX character classes

- `[:alnum:]` - alphanumeric characters
- `[:alpha:]` - alphabetic characters
- `[:digit:]` - digits
- `[:punct:]` - punctuation characters
- `[:lower:]` - lowercase letters
- `[:upper:]` - uppercase letters
- `[:space:]` - whitespace characters

Example:

```
ls | grep [[:digit:]]
```

- list all files with numbers in the filename

The Not Operator

We can also negate ranges of characters:

Not

- `[^abc]` - matches any character that is not a b or c
- `[^a-z]` - matches any non lowercase letter

Matching Any Number

Regular Expressions gain much of their power in their handling of repeated expressions. A RegExp followed by one of these repetition operators defines how many times that pattern should be matched:

- `*` - matches 0 or more occurences of the expression
- `\?` - matches 0 or 1 occurrences of the expression
- `\+` - matches 1 or more occurrences of the expression

Examples:

- `grep 't*a'` - matches things like `aste`, `taste`, `ttaste`, `tttaste`
- `grep '[[:alpha:]]\+a'` - matches the letter a only when it is preceeded by at least one letter.
- `grep '"\?Hello World"\?'` - matches Hellow World with or without quotes.

Beginning and End

Another thing RegExp can do is match the beginning and end of a line

Positional Operators

- `^` matches the beginning of a line
- `$` matches the end of a line

Examples:

`grep '^o$'`

- matches lines ending with "o"

`grep '^[A-Z]'`

- matches lines beginning with a capital letter

`ls -l | grep '^l'`

- prints all files that are links

Matching A Range of Repetitions

- `\{n\}` - preceeding item is repeated exactly n times
- `\{n,\}` - preceeding item is repeated at least n times
- `\{i,j\}` matches between `i` and `j` occurrences of strings that match `e`.

`grep -o '[0-9]\{3\}-\{0,1\}[0-9]\{2\}-\{0,1\}[0-9]\{4\}'` prints all social security numbers in a file (both `111-11-1111` and `111111111`)

Grouping Expressions

Grouping Expressions

`\(expr\)` : matches expr

- useful for grouping expressions together

Examples:

`a\(boat\)*` finds a, aboat, aboatboat, etc.

Regular Expression Rules

And a few more:

- `c1\|c2` matches the expression c1 or the expression c2.
- `\<` matches the beginning of a word
- `\>` matches the end of a word

`grep '\(left\)\|\(right\)'` matches left or right.

`grep 'top\{3\}'` searches for `toppp`.

`grep '[0-5]\{2\}\|[6-9]\{2\}'` searches for things like 12, 15, 68, 97, but not 19, 61.

A word about extended regular expressions

With extended regular expressions you do not need to escape special characters such as ?, +, () and {}. To use extended regular expressions with grep use the variant egrep or grep -E.

Extended regular expressions tend to be cleaner and easier to read:

`grep '\(woo\+t\)\{2,3\}'` becomes `egrep '(woo+t){2,3}'`.

There is also `fgrep` which does not understand any regular expressions (fastest).

Why we quote regular expressions

Suppose we have a directory with the following files in it:

```
num, num2, test
```

Now suppose we want to search the file test for the regular expression `nu*`. If we don't quote,

```
grep nu* test
```

gets expanded to

```
grep num num2 test
```

, which searches num2 and test for the string num.

Regular Expression Examples

How would you match any word that begins with `c` and ends with `d`?

Regular Expression Examples

How would you match any word that begins with `c` and ends with `d`?

```
grep '\<c[A-Za-z]*d\>'
```

```
grep '\<c.*d\>'
```

Regular Expression Examples

How would you match any word that begins with c and ends with d?

```
grep '\<c[A-Za-z]*d\>'
```

```
grep '\<c.*d\>'
```

If we just want 5 letter words beginning with c and ending with d:

```
grep -o '\<c...d\>' /usr/share/dict/words | uniq
```

```
caged
caked
caned
caped
cared
⋮
```

Great for crosswords!

More Grep Examples

```
$ grep '^smug'
```

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '^$'
```

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '^$'
```

searches for blank lines

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '^$'
```

searches for blank lines

```
$ grep 'B[oO][bB]$'
```

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '^$'
```

searches for blank lines

```
$ grep 'B[oO][bB]$'
```

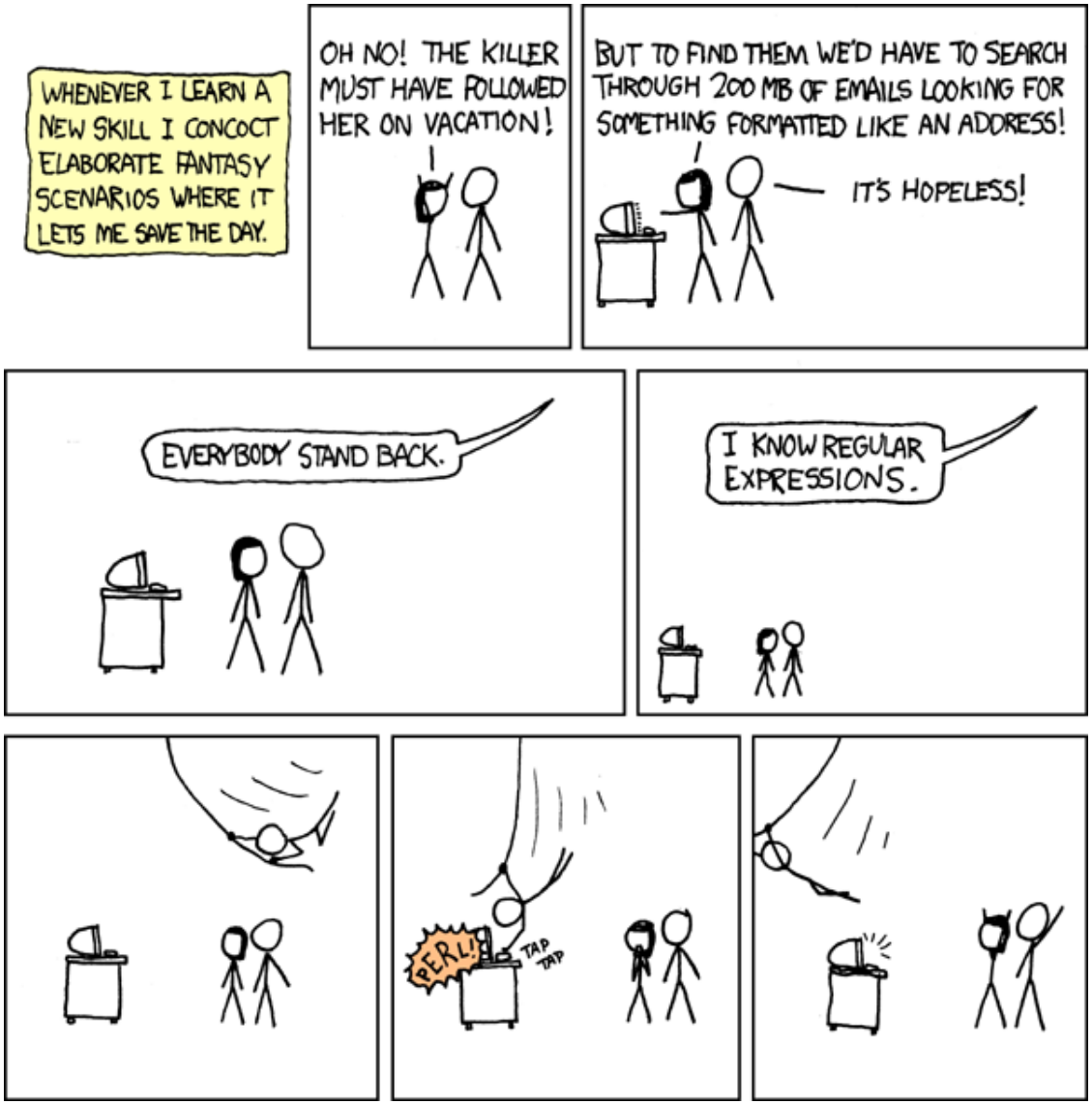
searches for lines that end in either BOB, Bob, BOb, or BoB.

Where else can we use Regular Expressions?

Example: less

When we are reading something using less, if we hit / and type a regular expression, less will highlight everywhere it occurs

- press `n` to move to the next match
- press `N` to move to the previous match


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

xkcd.com/208