Text Handling Commands

Sistemas Operativos I

Text

- There are many Unix commands that handle textual data:
  - operate on *text files*
  - operate on an *input stream*
- Operations:
  - Searching
  - Processing (manipulations)

Pattern Search Commands

- **grep, egrep, fgrep** : search files for text patterns
- **strings**: search binary files for text strings
- **find**: search for files whose name matches a pattern

grep - Get Regular Expression

grep [options] *regexp* [files]

regexp is a "regular expression" that describes some *pattern*.

files can be one or more files (if none, **grep** reads from standard input).

grep Examples

- The following command will search the files a,b and c for the string "foo". grep will print out any lines of text it finds (that contain "foo")

```
grep foo a b c
```

- Without any files specified, grep will read from *standard input*:

```
grep foo
```

grep options


```
-c   print only a count of matched lines.
-h   don't print filenames
-l   print filename but not matching line
-n   print line numbers
-v   print all lines that don't match!
```

Regular Expressions

- The string "foo" is a simple pattern where 'f' and 'o's characters are literals.

- grep actually understands more complex patterns that are described using *regular expressions*

- Their usage is much more extended than to grep command. Awk, sed and many other programs and languages also support *regex*

grep, egrep and fgrep

- All three search files (or stdin) for a text pattern.
  - **grep** supports regular expressions
  - **egrep** supports *extended regular expressions*
  - **fgrep** supports only fixed strings (nothing fancy)
- All have similar forms and options.

strings

- The **strings** command searches any kind of file (including binary data files and executable programs) for text strings, and prints out each string found.
- **strings** is typically used to search for some text in a binary file.

strings [options] files

The `find` command

- Find searches the filesystem for files whose name matches a pattern*.
- Here is a simple example:

```
find . -name unixtest -print
```

- *Actually find can do lots more!

The find command

- Execute something with the files that match:

```
find ~ -name “*.tx?” -exec dirname {} \;
find ~ -name “*.tx?” -exec basename {} \;
```

- Match atributes of the file:
  - For example, find files with size > 1 MB:

```
find /var/log -size +1M
```

Text Manipulation

- There are lots of commands that can read in text (from files or standard input) and print out a modified version of the input

- Some possible examples:
  - force all characters to lower case
  - show only the first word on each line
  - show only the first 10 lines

Common Concepts

- These commands are often used as filters, they read from standard input and send output to standard output

- Different commands for different specific functions
  - another way is to build one huge complex command that can do anything. This is not the Unix way!

Commands

head tail - show just part of a file

cut paste - deal with columns in a text file.

sort - reorders the lines in a file

tr - translate characters

uniq - find repeated or unique lines in a file.

head or tails?

- **head** shows just the "head" (beginning) of a file
- **tail** shows just the "tail" (end) of a file
- Both commands assume the file is a text file.

The `tail` command

`tail [options] [files]`

By default tail shows the last 10 lines.

Options:

- *-n* print the last *n* lines
- *-nc* print the last *n* characters
- *+n* print starting at line number *n* *(ignores the first n lines)*
- *+nc* print starting at character number *n*

The `tail` command (cont.)

More Options:

-r show lines in reverse order

-f don't quit at end of file (output appended data as the file grows)

Not all versions support this option!

Examples:

```
tail -100 somefile
tail +100 somefile
tail -r -c 100 somefile
```

The head command

head [options] [files]

By default head shows the first 10 lines.

Options: -*n* print the first *n* lines.

Example:

head -20 /etc/passwd

The cut command

- **cut** selects (and prints) columns or fields from lines of text.

cut options [files]

- You must specify an option!

cut options

-c*list* cut character positions defined in *list*.

list can be a:

- number (specifies a single character position)
- range (specifies a sequence of positions)
- comma separated list (specifies multiple positions or ranges)

cut -c examples

cut -c1 prints first char. (on each line).

cut -c1-10 prints first 10 char

cut -c1,10 prints first and 10th char.

cut -c5-10,15,20- prints
5,6,7,8,9,10,15,20,21,… char on each line.

more cut options

-f*list* cut fields identified in *list*.

A field is a sequence of text that ends at some separator character (delimiter).

You can specify the separator with the -d option.

-d*c* where *c* is the delimiter

The default delimiter is a tab

Specifying a delimiter

cut -d: -f1 prints everything before the first ":" (on each line).

What if we want to use space as the delimiter?

cut -d" " -f1

cut -f examples

cut -f1 prints everything before the first tab

cut -d: -f2,3 prints 2nd and 3rd : delimited columns

cut -d" " -f2 prints 2nd column using space as the delimiter

The paste command

- paste joins files horizonally, i.e., puts lines from one or more files together in columns and prints the result.

paste [options] files

- The combined output has columns separated by tabs.

paste cands votes

cands	votes
Gore	10
Bradley	10
Bush	10
McCain	10
Trump	10
Letterman	100

→

Gore	10
Bradley	10
Bush	10
McCain	10
Trump	10
Letterman	100

```
paste -c"\t" cands votes
```

-d*c* separate columns of output with character *c*.

The `sort` command

- **sort** reorders the lines in a file (or files) and prints out the result.

```
sort [options] [files]
```

Numeric vs. Alphabetic

By default, sort uses an alphabetical ordering

```
38
18
27
1256875
66
875
```

sort

```
1256875
18
27
38
66
875
```

sort -n

```
18
27
38
66
875
1256875
```

Alphabetic Ordering (ASCII)

'0' < '9' <'A' < 'Z'< 'a' < 'z' <

bbbb
BBBB
aaaa
AAAA
0000
####
$$$$

sort →

####
$$$$
0000
AAAA
BBBB
aaaa
bbbb

The uniq Command

- **uniq** removes duplicate adjacent lines from a file.
- **uniq** is typically used on a sorted file (which forces duplicate lines to be adjacent).
- **uniq** can also reduce multiple blank lines to a single blank line.

uniq examples

Gore
Bradley
Bush
McCain
Trump
Letterman

uniq →

Gore
Bradley
Bush
McCain
Trump
Letterman

10
10
10
10
10
100

uniq →

10
100

The tr command

- **tr** is short for *translate*.
- **tr** translates between two sets of characters.
  - replace all occurrences of the first character in set 1 with the first character in set 2, the second char in set 1 with the second char in set 2, …

No files! Always standard input!

tr [options] [string1 [string2]]

tr Example

Replace 'A' with 'a', 'B' with 'b', … 'Z' with 'z'

Gore
Bradley
Bush
McCain
Trump
Letterman

tr A-Z a-z

gore
bradley
bush
mccain
trump
letterman

tr can delete

-d option means "delete characters that are found in string1".

Gore
Bradley
Bush
McCain
Trump
Letterman

tr -d aeiou

Gr
Brdly
Bsh
McCn
Lttrmn

Another `tr` example - remove newlines

Gore
Bradley
Bush
McCain
Trump
Letterman

```
tr -d '\n'
```

GoreBradleyBushMcCainTrumpLetterman

More commands for text processing

- **wc** used to count lines, words, caracters, of a file
  - wc -l file.txt counts lines of file.txt
  - wc -w file.txt count words of a file.txt

- **sed** Stream Editor. Useful for find and replace a string within text files and input streams. Also, insert, delete words and lines. Caution: functional differences between unix flavors!

More commands for text processing

- Sed example for in-place substitution:

Substitute command

Flag for Global Replacement

```
sed -i 's/SEARCH_REGEX/REPLACEMENT/g' INPUTFILE
```

String or regular expression

String

By default output is STD.

-i option performs an in-place replacement