Text Handling Commands

Text

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

Searching 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 - **G**et **R**egular **Ex**p**r**ession

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 I
```


Regular Expressions

- The string "foo" is a simple pattern.
- grep actually understands more complex patterns that are described using *regular expressions*.
- We will look at regular expressions used by grep and other programs later.
- In case you can't wait - here is a sample:

```
grep "[A-Z]0{2,3}" somefile
```

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!

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!

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 join` - 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 **`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 **`tail`** command

`tail [options] [files]`

By default tail shows the last 10 lines.

Options:

- *-n* print the last *n* lines.
- *-n*c print the last *n* characters
- *+n* print starting at line number *n*
- *+n*c print starting at character number *n*

The **`tail`** command (cont.)

More Options:

Not all versions support this option!

- -r show lines in reverse order
- -f don't quit at end of file.

Examples:

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


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:

- 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.

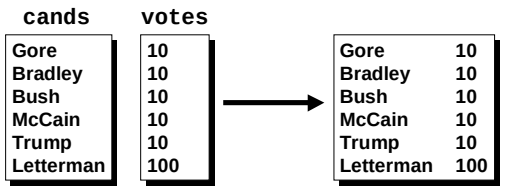
The `paste` command

- paste 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



paste options

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

you can use different c between each column.

-s merge subsequent lines from a single file.


```
paste -s -c"\t\n" records
```

```
records
```

```
Gore
10
Bradley
10
Bush
10
Letterman
100
```

```
Gore        10
Bradley     10
Bush        10
Letterman   100
```

```
paste -s -c"\t\t\n" records
```

```
Gore        10      Bradley
10          Bush    10
McCain      10      Letterman
100
```

The **`join`** command

- join combines the common lines of 2 *sorted* files.
- Useful for some text database applications, but not a very *general* command.
- Look at examples in the book if you are interested.

The **`sort`** command

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

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

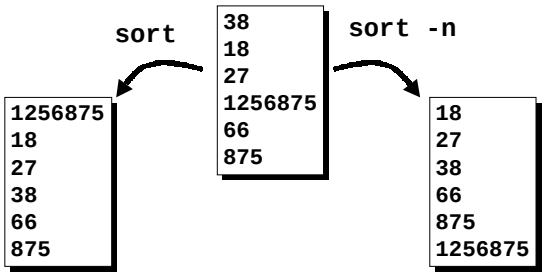

`sort` options

- **`-b`** ignore leading spaces and tabs
- **`-d`** sort in dictionary order (ignore punctuation)
- **`-n`** sort in numerical order
- **`-r`** reverse the order of the sort

tons more options!

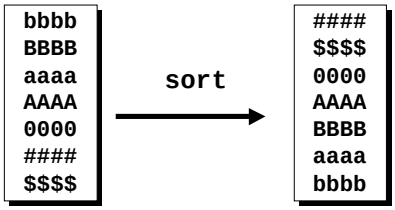
Numeric vs. Alphabetic

By default, sort uses an alphabetical ordering.



Alphabetic Ordering (uses ASCII)

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



ASCII codes

```
 32:   33:!  34:"  35:#  36:$  37:%  38:&  39:'
 40:(  41:)  42:*  43:+  44:,  45:-  46:.  47:/
 48:0  49:1  50:2  51:3  52:4  53:5  54:6  55:7
 56:8  57:9  58::  59:;  60:<  61:=  62:>  63:?
 64:@  65:A  66:B  67:C  68:D  69:E  70:F  71:G
 72:H  73:I  74:J  75:K  76:L  77:M  78:N  79:O
 80:P  81:Q  82:R  83:S  84:T  85:U  86:V  87:W
 88:X  89:Y  90:Z  91:[  92:\  93:]  94:^  95:_
 96:`  97:a  98:b  99:c 100:d 101:e 102:f 103:g
104:h 105:i 106:j 107:k 108:l 109:m 110:n 111:o
112:p 113:q 114:r 115:s 116:t 117:u 118:v 119:w
120:x 121:y 122:z 123:{ 124:| 125:} 126:~
```

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, …

```
tr [options] [string1 [string2]]
```

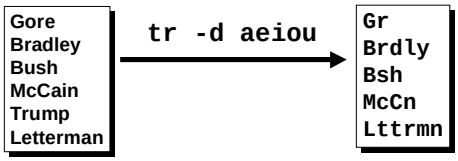
No files! Always standard input!

tr Example

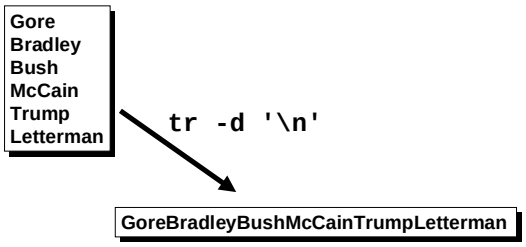
Replace 'A' with 'a', 'B' with 'b', … 'Z' with 'z'

tr can delete

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



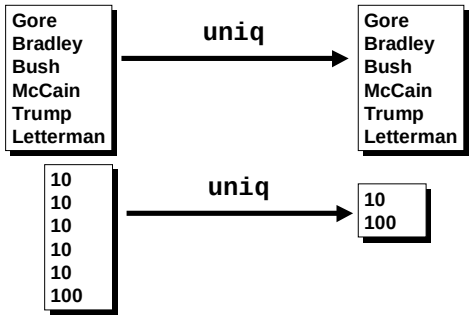
Another **tr** example - remove newlines



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



Exercises

- Convert a text file to all uppercase.
- Replace all digits with the character '#'
- sort the file /etc/passwd
- extract usernames from /etc/passwd
- find all files in your home directory that end in ".html".
- find all the lines in /etc/passwd that contain the number 10 (100 is OK, so is 710).