

## Summer 2013 Handout 3

### He must have been using vi

*The Moving Finger writes; and, having writ,  
Moves on: nor all thy Piety nor Wit  
Shall lure it back to cancel half a Line,  
Nor all thy Tears wash out a Word of it.*

—*Rubáiyát of Omar Khayyám*, quatrain 51

### How to prevent [Using open mode] in vi

If you get the message [Using open mode] when you enter **vi**, type **:quit!** and press **RETURN** to exit from **vi**. Then put the word **vt100** into the environment variable **\$TERM** if it is not already there. It should be there: we put it there in Handout 2, p. 13, lines 32–33.

```
1$ echo $TERM           See the contents of the variable $TERM.
2$ export TERM=vt100   Put vee tee one hundred into $TERM. No space around equal sign.
3$ echo $TERM           The contents of $TERM should now be vt100.
vt100
```

and try **vi** again.

### Before you enter vi

Because it's easier to practice **vi** on an existing file than to create a new one, copy the practice file **cremation** to your home directory and name it **junk**. Then launch **vi**.

```
1$ cd                   Go to your home directory.
2$ pwd                 Make sure you arrived there.

3$ echo $$S45          You created this variable in Handout 2, p. 13, lines 29–30
/home1/m/mm64/public_html/INFO1-CE9545/src

4$ cp $$S45/cremation junk
5$ ls -l junk           Make sure you created junk.
-r----- 1 abc1234 users 4524 May 28 15:18 junk

6$ chmod 600 junk      Give yourself permission to write as well as read it.
7$ ls -l junk           Make sure the chmod worked.
-rw----- 1 abc1234 users 4524 May 28 15:18 junk

8$ vi junk
```

**vi** should now display the first 23 lines of **junk**. If not, or if you see [Using open mode], see the note above.

### Using vi

Now that you're in **vi**, do not press **RETURN** (or **Enter**) after the following commands unless indicated. When you do have to press **RETURN**, do not type the space immediately before the **RETURN**. The space is for legibility only.


When in doubt in **vi**, press **ESC**. It could never hurt.

### Move the cursor

The four arrow keys should move the cursor. If not, use

```
h      left
j      down
k      up
l      right
```

Hold the keys down to repeat. Other convenient keys are

```
-      up (the minus sign)
RETURN down
BACKSPACE left (or try )
space bar right
```

To move the cursor to a given line number, type

```
15G      Go to line 15; the G must be uppercase.
1G      Go to the first line.
G      Go to the last line.
control-g See what line number the cursor is at.
```

```
:set number RETURN      Display the line numbers; no space before the RETURN.
:set nonumber RETURN     Don't display the line numbers.
```

The **w** and **b** keys move the cursor to the next or previous word (“word” and “back”). The **^** and **\$** keys move the cursor to the start or end of the line.

To move the cursor to the next occurrence of a string (e.g., “trail”), type

```
/trail RETURN
```

If you press **n** after this, **vi** will repeat the search and find the next occurrence of the same string. Similarly,

```
?trail RETURN
```

will find the previous **trail**, and **n** will repeat the search upwards.

### Scroll

If the file is too big to fit on the screen at once, use these lowercase control keys to scroll:

```
control-u Scroll the picture upwards half a screen; don't type the dash
control-d Scroll downwards half a screen.

control-y Scroll upwards one line.
control-e Scroll downwards one line.
```

### Delete

To delete a character, aim at it (i.e., place the cursor on it) and press **x**. This key repeats: it will suck in text from the right as you hold it down. (Nature abhors a vacuum.) **xxxx** or **4x** will delete four characters.

To delete a word, aim at the first letter of the word and type **dw** for “delete word”. A period will repeat any **vi** command. Or type **4dw** to delete four words.

To delete a line, aim at any point on the line and type **dd**. **4dd** will delete four lines.

To undo a deletion (or anything else), press **u** immediately thereafter.

### Re-insert the deleted text somewhere else

A *buffer* is a holding area or waiting room. Each chunk of deleted text will sit in the *unnamed buffer* until it is displaced by the next chunk. To insert this text back into your file somewhere else, aim at the desired point and press **p**. Deleted word(s) will reappear after the cursor, and deleted line(s) will reappear below the cursor.

An uppercase **P** will make the deleted text appear before or above the cursor. You must use lowercase **p** to make the deleted text appear after the last line in the file, and uppercase **P** to make the deleted text appear before the first line.

If you delete text and plan to re-insert it somewhere else, you must insert it before you perform another deletion. You can, however, insert the same text any number of times.

### Duplicate existing line(s)

To duplicate one line, aim at any point on the line and press **yy** (for “yank”). Then insert a copy of the line wherever you wish by using the **p** or **P** commands as above. **4yy** will duplicate four lines.

### Create new line(s)

Aim at any point on the line immediately above the place where you want to create the new line(s). Press a lowercase letter **o** and an empty line will appear below the cursor. You are now in *insert mode* (also known as *open mode* or *append mode*). Type whatever text you want to insert onto the new line(s), pressing **RETURN** whenever you wish to begin another new line. Then press **ESC** to get out of insert mode.

You can also press an uppercase letter **O** to enter insert mode and make the empty line appear *above* the current line. Lowercase **o** must be used to make the empty line appear after the last line in the file, and uppercase **O** to make the empty line appear before the first line.

### Insert new text in the middle of a line

Aim at the insertion point and press **a**. You are now in insert mode. Type whatever text you want to insert, pressing **RETURN** whenever you wish to create another new line. Then press **ESC** to get out of insert mode.

The above procedure inserts the new text to the right of where the cursor was just before you gave the **a** command: i.e., it inserts between the character you aimed at and the following character.

To enter insert mode and insert new text to the left of the cursor, press **i** instead of **a**. **a** must be used to insert new characters after the last character on the line, and **i** to insert new characters before the first character.

### Insert mode

The letters **o**, **O**, **a**, and **i** will make you enter insert mode. When in insert mode, the **DELETE** or **BACKSPACE** keys will erase typing errors, although the erased characters do not disappear until you type new ones or exit from insert mode. The four arrow keys do not work in insert mode. *You must always press **ESC** to exit from insert mode before doing anything else.* To make the words **INSERT MODE** automatically appear on the bottom line of the screen whenever you’re in insert mode,

```
:set showmode RETURN
```

If you put a **vi** command into your **EXINIT** environment variable, it will be executed automatically whenever you run **vi**. See Handout 2, p. 13, lines 17–18.

### Join and split lines

To join a pair of consecutive lines into one long line, aim at any point on the first line and press **J**. To split a long line into two lines, aim at the character you want to be at the end of the first line, press **a** to enter insert mode, press **RETURN** to insert a newline, and press **ESC** to exit from insert mode.

**Global substitution**

To see every line in the file that contains the word “trail”, type

```
:g/trail/p RETURN  
RETURN
```

*p. 325 for g, 321 for p*  
*A second RETURN makes the lines go away.*

To change all the “gold”s to “silver”s, type

```
:1,$s/gold/silver/g RETURN
```

*p. 321 for \$, 323 for s and g*

To make this change only on lines 10 through 20, type

```
:10,20s/gold/silver/g RETURN
```

To add the word “silver” to the start of lines 10 through 20, type

```
:10,20s/^/silver/ RETURN
```

*^ means start of line, p. 324*

To add the word “silver” to the end of lines 10 through 20, type

```
:10,20s/$/silver/ RETURN
```

*\$ means end of line, p. 324*

Finally, to remove every word “gold” from lines 10 through 20, type

```
:10,20s/gold//g RETURN
```

*no space between the slashes*

A leading colon in a **vi** command makes the cursor jump down to the bottom line of the screen. At the end of the command, press **RETURN** to make the cursor go back up.

**Write your changes back to the disk**

When you enter **vi**, it makes a copy of the file you are editing and displays the copy on the screen. All of your editing commands are applied only to the copy. The original version of the file on the disk remains unchanged until you give the “write” command:

```
:w RETURN
```

*p. 327*

This command writes the copy back onto the disk, overwriting the original version of the file. Get into the habit of writing every 10 or 15 minutes just in case the machine crashes or your connection is cut off.

If you say

```
:w otherfilename RETURN
```

the copy will be written into another file, creating the other file if it does not already exist. I often save a snapshot every 10 minutes.

```
:w otherfilename1 RETURN  
:w otherfilename2 RETURN  
:w otherfilename3 RETURN
```

**Exit from vi**

To quit from **vi**, type

```
:q RETURN
```

*p. 320*

For your own safety, **vi** will obey the **:q** command only if you have previously given a **:w** command since your last editing change. If you really do want to quit without writing your editing changes (i.e., if you realize while editing that you’ve only made the file worse), type

```
:quit! RETURN
```

*with an exclamation point*

**Create a new file**

```

1$ cd                Go to your home directory.
2$ pwd              Make sure you arrived there.

3$ vi neuman        Create a new file named neuman.
iCall me Alfred.    Enter insert mode; don't press RETURN after the i;
                    type anything; then press ESC.

```

The column of tildes ~ fills the part of the screen not occupied by your file.

**What can go wrong**

The **o**, **O**, **a**, and **i** commands put you into insert mode. You must not be in this mode when giving any of the commands in this Handout. If you are in insert mode, pressing **ESC** will always get you out of it. If you are not in insert mode, **ESC** gives you a reassuring beep and does nothing else. When in doubt, or if **vi** seems to be confused or unresponsive., press **ESC**. It couldn't hurt.

If the **ESC** key doesn't work, try **control-[]**.

If every key responds in a way that is wildly wrong, make sure you haven't pressed **Shift** or **Caps Lock** by mistake.

If you suspect that the screen is messed up, press **ESC control-l** (lowercase L) to clear and redraw the screen. This is called "refreshing the screen".

Although it is possible, do not use **vi** to create a line longer than 80 characters. With a 90-character line, most would terminals display the last ten characters below the first 80, but for all other purposes the 90 characters count as a single line. For example, our **lpr** program will print only the first 80 characters of each line; a longer line will be truncated on paper.

**vi shortcuts**

Here are faster ways to do some of the above tasks. As always, you must not be in insert mode when you give the following commands.

- r** To replace one character by another without bothering to enter and exit from insert mode, aim at the character you want to replace. Type **r** and *one* replacement character.
- s** **s** (for "substitute") means **xi**. In other words, delete the character that the cursor is pointing to and then enter insert mode. **4s** will delete four characters and then enter insert mode.
- cw** "Change word" means **dwi**. In other words, delete the word that the cursor is pointing to and then enter insert mode. **4cw** will change four words.
- D** Delete all the characters from the cursor to the end of the line. This is faster than typing **x** repeatedly.
- C** **C** (for "change line") means **Da**. In other words, delete the rest of the line and then enter insert mode.
- %** Point to one of a pair of (parentheses), [square brackets], or {curly brackets} and press **%**. The cursor will jump to the corresponding character, or will beep if there is no match. Press another **%** to jump back to where you started.

**vi in a GUI**

To run **vi** in a window, you must first launch an X Window server.

(1) On Mac OS X, launch an X Window server such as **X11.app**. In the **xterm** that **X11.app** opens for you, log into **i5.nyu.edu** by saying

```
1$ ssh -Y abc1234@i5.nyu.edu
```

(2) On a PC, you can download a free X Window server if you don't already have one. For example, download and install **Xming-fonts** and then **Xming** from

<http://sourceforge.net/projects/xming>

Launch **Xming** after the fonts are installed. Then launch **putty.exe**.

Putty Release 0.57: **PUtTY Configuration** → **Category** → **Connection** → **SSH** → **Tunnels**

Putty Release 0.60: **PUtTY Configuration** → **Category** → **Connection** → **SSH** → **X11**

and check “Enable X11 forwarding”. Then log into i5.nyu.edu.

(3) On either platform, launch **vim** (improved vi, <http://www.vim.org/>) in a GUI:

```
2$ echo $DISPLAY
3$ vim -g -geometry=80x36 filename
4$ alias v='vim -g -geometry=80x36'
```

You can select text with the mouse, delete it with **d**, and re-insert it with **p** or **P**.

### vi bibliography

Print yourself a copy of the **vi** manual page at

<http://i5.nyu.edu/~mm64/man/>

Also see *Learning the vi Editor, 6th ed.*, by Arnold Robbins and Linda Lamb.

<http://www.oreilly.com/catalog/vi6/>

### ▼ Homework 3.1: make a .plan and a .project (not to be handed in)

Finger yourself and finger me:

```
1$ finger | more           no argument: finger everybody who is logged in
2$ finger abc1234         your login name
3$ finger mm64           my login name
i4% finger mm64@i5.nyu.edu from another host (currently not allowed)
```

Then create two files named **.plan** and **.project** in your home directory. Their names must begin with a dot; you’ll have to say **ls -la** to see them. The **.plan** file should say that <http://i5.nyu.edu/~abc1234/> is your World Wide Web home page. The **.project** file must be no more than a single line.

Turn on all three **r** bits of the **.plan** and **.project** files if they are not already on. Turn on all three **x** bits of your home directory if they are not already on (Homework 1.2). Then **finger** yourself again.

▲

### ▼ Homework 3.2: ✉ customize the mailx program (not to be handed in)

Create a file in your home directory named **.mailrc** containing these commands. See **mailx(1)**.

<b>set crt</b>	<i>Press space bar, b, or q when reading a long letter.</i>
<b>set hold</b>	<i>Don’t delete any letters unless I say so.</i>
<b>set keepsave</b>	<i>Don’t delete any letters unless I say so.</i>

▲

### ▼ Homework 3.3: customize emacs (for emacs users only; not to be handed in)

In your home directory, create a file named **.emacs** if you don’t already have one, containing these commands. They have parentheses because they are written in the language Lisp.

```
(setq-default auto-save-interval 600)
(setq-default require-final-newline 1)
```



## The World Wide Web

The World Wide Web holds millions of files of information, called *pages*, stored on computers all around the world. You can display a page on the screen by running a program called a *web browser*. Popular browsers include Apple Safari, Mozilla Firefox, and Microsoft Internet Explorer. Great browsers of the past included Mosaic and Netscape.

Each page contains underlined words called *hyperlinks*, or *links* for short. When you click on a link, the browser will display the page that the link leads to. You can then pursue the links on the destination page, or you can go back to the page you came from.

Instead of leading to a file of data, a link can also lead to a program called a *gateway*. A gateway can be written in any language: the shell language, C, C++, Perl, etc. When you click on a link leading to a gateway, the gateway will execute and display its output. You can also feed input into a gateway by checking checkboxes, pushing buttons, filling in the blanks, or popping up menus. A page with these widgets is called a *form*.

## Configure the Apache Web Server

When anyone in the world points their web browser at a web page stored on our host i5.nyu.edu, the program on i5.nyu.edu that sends them a copy of the page is called the *web server*. The name of this program is `/usr/apache/bin/httpd` (“Hyperterxt Transport Protocol Dæmon”). Here are excerpts from its configuration file `/etc/apache2/2.2/httpd.conf`. The documentation for the directives in this file is at

<http://www.apache.org/docs/mod/directives.html>

```
1 #If a URL specifies no filename, try these filenames by default.
2 #If the directory has no files with these name, do an ls -l of the
3 #directory instead.
4
5 DirectoryIndex index.html
6
7 #In a URL, a user's loginname with a ~ in front of it stands
8 #for the following subdirectory of that user's home directory.
9
10
11 #The following is automatically prefixed to the directory specified
12 #in a URL, unless the specified directory begins with a user's
13 #loginname with a ~ in front of it.
14
15 DocumentRoot "/var/apache2/2.2/htdocs"
16
17 #In the URL of a gateway, the following short string stands
18 #for the much longer string.
19
```

## ▼ Homework 3.4: create a World Wide Web home page

Create a file named `index.html` in the `public_html` subdirectory of your home directory. You can write your own, or copy the one in `$S45/public_html` and modify it.

```

1$ cd ~/public_html
2$ pwd

3$ cp $S45/public_html/index.html .
4$ ls -l index.html

5$ chmod 644 index.html           turn on all three r's: rw-r--r--
6$ ls -l index.html

7$ vi index.html                   Change it to describe yourself.

```

Do not write the line numbers or the blank after each line number. `chmod` your `index.html` file to `rw-r--r--`. `chmod` your home directory and its `public_html` subdirectory to `rwxr-xr-x` if they do not already have these permissions (Homeworks 1.2 and 1.3).

Then tell your friends that your World Wide Web URL is `http://i5.nyu.edu/~abc1234/` where `abc1234` is your `i5.nyu.edu` login name, and put a note to this effect in your `.plan` file. In Unix, `~abc1234` stands for `abc1234`'s home directory. In a URL, `~abc1234` stands for the `public_html` subdirectory of `abc1234`'s home directory because of the `UserDir` in `httpd.conf`. The default file-name in a URL is `index.html` because of the `DirectoryIndex` in `httpd.conf`.

If you display your home page with a browser, and then re-edit it while the browser is still running, press the browser's **Reload** button to update the display.

To see which machines have accessed your home page,

```

8$ cd /var/apache2/2.2/logs
9$ pwd

10$ ls -l access_log                How many characters?
-rw-r--r--  1 root      root      85596287 May 28 15:17 access_log

11$ wc -l access_log                How many lines? Be patient.
 855124 access_log

12$ grep abc1234 access_log | more   Be patient.

```

For the web pages of the other students in the class, see

```

http://i5.nyu.edu/~mm64/common/students.html    class roster
http://i5.nyu.edu/~mm64/INFO1-CE9545/00120132.html    class photo

```

See the 10-minute guide to HTML in the HTML home page

```
http://www.w3.org/MarkUp/
```

—On the Web at

```
http://i5.nyu.edu/~mm64/INFO1-CE9545/src/public_html/index.html
```

```

1 <HTML>
2 <HEAD>
3 <TITLE>These words are displayed in window's title bar.</TITLE>
4 </HEAD>
5
6 <BODY>
7 <H1>These words are displayed in big letters in the window itself.</H1>
8
9 <H2>Pairs of tags</H2>
10 <P>
11 Surround each tag with "less than" and "greater than" signs.

```



12 Do not write these characters for any other purpose.  
13 If you really must say "less than" and "greater than",  
14 say &lt; and &gt; and don't forget the semicolons.  
15 Write an ampersand as &amp;.  
16 </P>  
17  
18 <P>  
19 This is the start of a new paragraph.  
20 The words within the tags are case-insensitive, except if they're in quotes.  
21 Most tags come in pairs.  
22 The second tag of the pair always starts with a slash.  
23 For example,  
24 surround the entire page with a pair of HTML tags (lines 1 and 148).  
25 Every page consists of a head and a body,  
26 each enclosed by its own pair of tags.  
27 </P>  
28  
29 <P>  
30 One tag that does not come in pairs  
31 is the HR tag that draws a horizontal rule (line).  
32 <HR>  
33 <!-- A comment has an exclamation point, four dashes, and two spaces. -->  
34 </P>  
35  
36 <H2>Headers</H2>  
37 <P>  
38 The H1 tags make a big header;  
39 the H2 and H3 tags make smaller headers.  
40 The TITLE tags enclose the title of the window that displays the page.  
41 The TITLE should be shorter than the H1.  
42 Put the TITLE tags in the head,  
43 and the H1 tags in the body.  
44 </P>  
45  
46 <H2>Special fonts</H2>  
47 <P>  
48 You can apply  
49 <EM>emphasis</EM>,  
50 <STRONG>strong emphasis</STRONG>,  
51 or  
52 <FONT SIZE = 7>size</FONT>.  
53 </P>  
54  
55 <H2>Spacing</H2>  
56 <P>  
57 To make the file easier to edit,  
58 put each phrase on a line by itself  
59 and skip lines for legibility.  
60 The browser will pack the text together  
61 in civilized paragraphs when displaying it on the screen.  
62 </P>  
63  
64 <P>  
65 Use the "break" tag between lines of a poem:

```
66 </P>
67
68 <P>
69 Two households, both alike in dignity,
70 <BR>
71 In fair Verona, where we lay our scene,
72 <BR>
73 From ancient grudge break to new mutiny,
74 <BR>
75 Where civil blood makes civil hands unclean.
76 </P>
77
78 <P>
79 "Preformat" text to preserve its indentation and relative position:
80 </P>
81
82 <PRE>
83 From forth the fatal loins of these two foes
84     A pair of star-cross'd lovers take their life;
85 Whose misadventured piteous overthrows
86     Do with their death bury their parents' strife.
87 </PRE>
88
89 <H2>Lists</H2>
90 <UL>
91 <LI>Surround the entire list with a pair of UL tags.</LI>
92 <LI>Surround each item with a pair of LI tags.</LI>
93 <LI>To get numbers instead of bullets, change the UL's to OL's.</LI>
94 </UL>
95
96 <H2>Tables</H2>
97 <P>
98 Surround the entire table with a pair of TABLE tags (lines 105 and 123).
99 Within the table, surround each row with a pair of TR tags.
100 Within each row, surround each item of data with a pair of TD tags.
101 Before the first row and/or at the start of each row,
102 surround each table header with a pair of TH tags.
103 </P>
104
105 <TABLE BORDER>
106 <TR>
107 <TH></TH>
108 <TH>Column 1</TH>
109 <TH>Column 2</TH>
110 </TR>
111
112 <TR>
113 <TH>Row 1</TH>
114 <TD>row 1, col 1</TD>
115 <TD>row 1, col 2</TD>
116 </TR>
117
118 <TR>
119 <TH>Row 2</TH>
```

```

120 <TD>row 2, col 1</TD>
121 <TD>row 2, col 2</TD>
122 </TR>
123 </TABLE>
124
125 <H2>Special Characters</H2>
126 <OL>
127 <LI>&uuml;mlaut, &cedil;edilla</LI>
128
129 <LI>Arabic: "Haza modhish!" ("It's amazing!",
130 from the Berlitz Phrasebook)
131 <SPAN LANG = "AR">&#x0647;&#x0632;&#x0627;
132 &#x0645;&#x062F;&#x0647;&#x0634;</SPAN></LI>
133
134 <LI>Hebrew: "Kosher"
135 <SPAN LANG = "HE">&#x05DB;&#x05BC;&#x05B8;&#x05E9;&#x05C1;&#x05B5;&#x05E8;</SPAN></LI>
136
137 <LI>Russian: "winter forest"
138 <SPAN LANG = "RU">&#x0437;&#x0438;&#x043C;&#x043D;&#x0438;&#x0439;
139 &#x043B;&#x0435;&#x0441;</SPAN></LI>
140
141 <LI>Chinese characters for "up", "down", "stream":
142 <SPAN LANG = "ZH">&#x4E0A; &#x4E0B; &#x5DDD;</SPAN></LI>
143 </OL>
144 </BODY>
145 </HTML>

```

	Column 1	Column 2
Row 1	row 1, col 1	row 1, col 2
Row 2	row 2, col 1	row 2, col 2

**Accent marks, special characters, other character sets**

For the ümlaut, çedilla, etc. (line 127), don't forget the leading ampersand and trailing semicolon. All the special characters are at

<http://www.w3.org/TR/REC-html40/sgml/entities.html>

Look up the four hexadecimal digit code for each foreign character in the code charts at <http://www.unicode.org/charts/>.



**▼ Homework 3.5: create a file using vi (not to be handed in)**

Go to your home directory and use **vi** to create a file whose name is your last name (not your login name) in all lowercase. Here is what goes on each line of the file:

- Line 1: your login name
- Line 2: your real name
- Lines 3 and 4: your imaginary home address
- Line 5: your imaginary home phone
- Line 6: your imaginary work phone
- Line 7: the section number (1 for Wednesday)
- Line 8: the URL of your home page (if you have one outside of this class), or, as a last resort, your home page that you created in this class
- Line 9: your version of Unix at home or at work, or, as a last resort, here on **i5.nyu.edu**

Line 10: your version of the shell at home or at work, or, as a last resort, here on **i5.nyu.edu**

Line 11: the editor you use at home or at work, or, as a last resort, here on **i5.nyu.edu: ed, vi, emacs, pico**

Line 12: the programming languages you know

If you know no languages, create an empty line 12: the file must be at least 12 lines long. You can write whatever you want on subsequent lines. Don't type the line numbers or the space after each line number:

```

1 abc1234
2 Alfred E. Neuman
3 1700 Broadway           Don't type your real address.
4 New York, NY 10019-5905
5 (914) 234-5678         Don't type your real phone numbers.
6 (212) 876-5432
7 1
8 http://www.madmagazine.com/people/aen
9 Linux
10 Korn shell
11 vi
12 C, C++, Java, Perl
13 I have also taken the following courses ...
14 I saw the Grateful Dead twice ...
15 Robert Moses was a genius, but his ...

```

The letters in your login name must be lowercase. Be sure that the first line contains nothing except the characters of your login name: no blanks or tabs. Be sure that the seventh line contains nothing except one digit: no blanks, tabs, or other characters. Do not write the words "Login name:" or "Section number:".

Be sure that your login name is on line 1: many people accidentally create a blank line 1 and put the login name on line 2. If this happens, press **ESC** to make sure you're not in insert mode, type **1G** to move the cursor to line 1, and type **dd** to delete that line.

Make some deliberate errors while you're typing and then fix them. After you have exited from **vi**, give the command

```
1$ cat -n neuman
```

to make sure all 12 pieces of information are on the correct lines.

Then **mv** your file into the directory **~mm64/public\_html/INFO1-CE9545/bio**. Put nothing into this directory except your biography file: no shellscripts or other homeworks. **chmod** your biography file to **r--r--r--**. Do not hand in a printout of your biography file.

If the **mv** command gives you an error message, see if it's because the directory already contains a file whose name is your last name. If so, name your biography **neuman2** and try again.

▲

### ▼ Homework 3.6: split lines that are longer than 80 characters.

Copy the file **\$\$45/greatexpectations.txt** to your **public\_html** directory and display it on the web.

```
1$ cd ~/public_html
2$ pwd
```

```
3$ cp $$45/greatexpectations.txt .
4$ ls -l greatexpectations.txt
```

or use the **escape-** in Handout 2, p. 12

```
5$ chmod 644 greatexpectations.txt      r--r--r--
6$ ls -l greatexpectations.txt
```

You will see that two of the eight lines are longer than 80 characters.

Edit your copy of the file with **vi**:

```
7$ chmod 644 greatexpectations.txt      rw-r--r--
8$ ls -l greatexpectations.txt
```

```
9$ vi greatexpectations.txt
```

In **vi**, it may be harder to see the overlong lines. There are three ways to identify them.

- (1) Move the cursor to the first line of the file. Then press **RETURN RETURN RETURN** to travel down the file one line at a time. If one of the **RETURN**'s skips a line, you have just passed a line that is longer than the 80 characters that would fit on a single line on the screen.
- (2) Press space bar space bar space bar to move the cursor to the end of the line. If the cursor moves to the right edge of the screen and then continues to the line below, you are on a line that is longer than the 80 characters that would fit on a single line on the screen.
- (3) Compare the printout with what **vi** displays on the screen. Our printer prints only the first 80 characters of each line.

Split each line that is longer than 80 characters into two or more lines of at most 80 characters each. You can split a line by inserting a **RETURN** character into it. Then print the file and hand it in. Do not change Dickens's spelling, punctuation, or grammar. You get no credit if the printout contains the nonsense words **wordsI** or **s**.

▲

### **ls** can see where its standard output has been directed

**ls** has the exceptional ability to detect whether its standard output has been directed to a terminal; only then will it output multiple columns. See **isatty(3)** and the shell **-t** expression in **ksh(1)** p. 20.

```
1$ cd /home1
2$ pwd
```

```
3$ ls
a      e      h      l      o      s      w
b      expired i      m      p      t      x
c      f      j      mydata q      u      Y
d      g      k      n      r      v      z
```

```
4$ ls | wc -l
28
```

*indirect evidence that **ls** output 28 lines*

```

5$ ls > ~/junk
6$ head ~/junk
a
b
c
d
e
expired
f
g
h
i

```

### Build a pipeline

```

1$ cd ~mm64/public_html/INFO1-CE9545/homework
2$ pwd

```

```

3$ ls -l
total 49
-r--r--r-- 1 pqa7501 users 346 Oct 28 15:40 alevante
-r--r--r-- 1 jrb210 users 226 Oct 28 18:23 bruce
-r--r--r-- 1 cqb6429 users 404 Nov 5 12:32 burns
-r--r--r-- 1 sqc1065 users 299 Nov 20 03:50 choi
-r--r--r-- 1 jrb210 users 225 Oct 29 18:24 spruce

```

*He's telling me more and more about some useless information supposed to*

```

4$ ls -l | tail +2
-r--r--r-- 1 pqa7501 users 346 Oct 28 15:40 alevante
-r--r--r-- 1 jrb210 users 226 Oct 28 18:23 bruce
-r--r--r-- 1 cqb6429 users 404 Nov 5 12:32 burns
-r--r--r-- 1 sqc1065 users 299 Nov 20 03:50 choi
-r--r--r-- 1 jrb210 users 225 Oct 29 18:24 spruce

```

```

5$ ls -l | tail +2 | awk '{print $3}'
pqa7501
jrb210
cqb6429
sqc1065
jrb210

```

```

6$ ls -l | tail +2 | awk '{print $3}' | sort
cqb6429
jrb210
jrb210
pqa7501
sqc1065

```

```

7$ ls -l | tail +2 | awk '{print $3}' | sort | uniq -d
jrb210

```

*“duplicate”*

**What the single quotes are for: p. 75**

```

1$ rm *                               Remove all the files in the current directory.
2$ rm '*'                             Remove the file whose name is *; Handout 2, p. 5.

3$ echo hello > junk                 Output the word hello to a file named junk.
4$ echo hello '>' junk               Output the three words hello > junk to the screen.

5$ echo $HOME                         Output the full pathname of your home directory.
6$ echo '$HOME'                       Output the five characters $HOME.

```

**The bin directory that is a child of your home directory: p. 37**

Put your shellscripts and other *executables* (programs) in the **bin** directory that is a child of your home directory. **bin** stands for “binary”, because in the early days of Unix all executable files were binaries (i.e., compiled) rather than scripts (i.e., interpreted).

```

1$ echo hi there
hi there

```

```

2$ echo $PATH                          See ksh(1) pp. 13, 21.
/home1/a/abc1234/bin:/usr/bin:/local/bin:/usr/local/bin:/usr/sfw/bin:/usr/ucb:.

```

Write the ASCII code (Handout 2, p. 1) of the newline character as octal **12** with a zero in front of it (p. 43).

```

3$ echo $PATH | tr : '\012' | cat -n | more
 1 /home1/a/abc1234/bin
 2 /usr/bin                binary. /bin is a symbolic link to this directory.
 3 /local/bin
 4 /usr/local/bin
 5 /usr/sfw/bin           Sun Freeware: http://www.sunfreeware.com/
 6 /usr/ucb              University of California at Berkeley
 7 .                      your current directory: Handout 1, p. 9

```

If you run a program that is in one of the directories listed in your **\$PATH**, you do not have to specify the program’s full pathname. For example, you can get away with

```

4$ ls -l

```

instead of

```

5$ /bin/ls -l

```

because the directory **/bin** is in your **\$PATH**. It’s a symbolic link to **/usr/bin**.

```

6$ ls -ld /bin
lrwxrwxrwx  1 root    root          9 Jul  5  2012 /bin -> ./usr/bin

```

On the other hand, if you run a program that is not in one of these directories, you have to specify the program’s full pathname:

```

7$ moon                               Handout 2, p. 14, line 88
ksh: line 1: moon: not found

```

```

8$ ~mm64/bin/moon

```

```

9$ ./a.out                            for dot, Handout 1, p. 9; for a.out, Handout 3, p. 22

```

### Environment variables

The *environment* variables are the ones shared by all your programs. See `environ(5)`. The `env` command will output the name and contents of all your environment variables, including `$PATH` and `$HOME`. To see the environment variables in Windows, right-click on **My Computer**; then **Properties** → **Advanced** → **Environment Variables**.

The `-d` option tells `sort` to ignore underscores, apostrophes, and other punctuation marks; `-f` tells `sort` to ignore case (upper vs. lower). Combine them to `-df` as in Handout 1, p. 3, line 61.

```
1$ env | sort -df | more
```

### which: pp. 141–143

The `which` command outputs the full pathname of the file that would execute if you were to run the specified command.

- (1) `which` does not search every directory, but only the directories in your `$PATH` from left to right in the order listed.
- (2) It does not even look at every file in these directories, but only at the files which you have permission to execute—those whose `x` bits are turned on.
- (3) `which` stops as soon as it finds the first executable file with the required name: it doesn't necessarily search through all the directories in your `$PATH`. See `find(1)` (Handout 10, p. 1) for a general-purpose searching program.

```
1$ which ls
/bin/ls
```

```
2$ which whoami
/bin/whoami
```

*Handout 1, p. 1, line 2*

```
3$ which gcc
/bin/gcc
```

### Create a shellscript: pp. 80–82

Go to the `bin` subdirectory of your home directory; say `pwd` to make sure you got there. Do not go to the `/bin` directory at the top of the tree. Never name a shellscript (or any other program) `test`: a program with that name already exists. In fact, always use `which` to see if there already is a program with the name you plan to use. Other names that must not be used are that of an `alias` in your `.profile`, or of a special command in pp. 27–38 of `ksh(1)`.

```
1$ cd ~/bin
2$ pwd
/home1/a/abc1234/bin
3$ which ksh
/bin/ksh
4$ ls -ld /bin
lrwxrwxrwx  1 root  root           9 Jul  5  2012 /bin -> ./usr/bin
5$ vi own2
```

Use `vi` (or any other editor: `emacs`, `ed`, etc.) to create a file named `own2` containing commands and a comment saying what they do. You get credit for a shellscript only if it contains the comment. Each shellscript that you write for this course will be so simple that a single sentence will suffice. Use the imperative tense for brevity:



*Indicative:* This shellsript outputs the login name of everyone who owns more than one file in the current directory.

*Imperative:* Output the login name of everyone who owns more than one file in the current directory.

```

http://i5.nyu.edu/~mm64/INFO1-CE9545/src/own2
#!/bin/ksh
#Output the login name of everyone who owns more than one file in the
#current directory.
#Sample use: own2

ls -l | tail +2 | awk '{print $3}' | sort | uniq -d

```

You can download a file from the web with the rudimentary web browser **lynx** (with a lowercase L).

```

6$ cd ~/bin
7$ pwd

8$ lynx -source http://i5.nyu.edu/~mm64/INFO1-CE9545/src/own2 > own2
9$ ls -l own2

```

The “pound exclamation point” line is not a comment, although it looks like one. The **#!** must be the first two characters of the first line. Nothing can come before them, not even a blank or a comment. Nothing can come above them. Put no space between the **#!** and the **/bin/ksh**. Put nothing to the right of the **#!/bin/ksh**. See **exec(2)**.

Use **chmod** to turn on your shellsript’s “read” and “execute” bits.

```

10$ pwd
/home1/a/abc1234/bin

11$ ls -l own2
-rw-----  1 abc1234  users          181 May 28 10:09 own2

12$ which own2
no own2 in any of the directories in your $PATH
Make sure there isn't already another own2

13$ chmod 755 own2
You can say a+rx instead of 755.
14$ ls -l own2
-rwxr-xr-x  1 abc1234  users          181 May 28 10:09 own2

15$ which own2
/home1/a/abc1234/bin/own2

16$ cd ~mm64/public_html/INFO1-CE9545/homework
17$ pwd
Okay, let's try it.

18$ own2
jrb210
19$
Don't need to say "run" or "exec".

```

You can use this shellsript in all of the following ways because you did not specify in the shellsript where the **uniq -d** should direct its standard output:

```

20$ own2                                to the screen of your terminal
21$ own2 | wc -l
22$ own2 | lpr                            $PRINTER eliminated the need for the -P option of lpr.

23$ own2 > outfile                        Create the file in the current directory, whose w bit must be on.
24$ own2 >> outfile                        If the file outfile already exists, its w bit must be on.

25$ own2 > /another/directory/outfile
26$ own2 > /dev/pts/10                    to the screen of someone else's terminal
27$ own2 > /dev/null                      throw away the standard output of own2
    
```

“The symbolic codes are difficult to explain”—p. 56

<i>read</i>	<i>write</i>	<i>execute</i>	
chmod a+r own2	chmod a+w own2	chmod a+x own2	<i>all three</i>
chmod a-r own2	chmod a-w own2	chmod a-x own2	
chmod u+r own2	chmod u+w own2	chmod u+x own2	<i>user (i.e., the owner)</i>
chmod u-r own2	chmod u-w own2	chmod u-x own2	
chmod g+r own2	chmod g+w own2	chmod g+x own2	<i>group</i>
chmod g-r own2	chmod g-w own2	chmod g-x own2	
chmod o+r own2	chmod o+w own2	chmod o+x own2	<i>other (i.e., everybody else)</i>
chmod o-r own2	chmod o-w own2	chmod o-x own2	

▼ Homework 3.7: search for duplicates

Hand in exactly one of the following three shellscripts. You get no credit for this assignment if you hand in more than one, or if you fail to hand in the shellscript’s output on the same page. You also get no credit if you use **tail +2** unnecessarily, or fail to use it when necessary. Output nothing except what I ask for: no titles, explanatory sentences, etc.

(1) Write a one-line shellscript named **logs** that will output the login name of everyone who is logged in on more than one terminal. Use the method shown above. **who** does not output a header line, so do not filter the output of **who** through **tail +2**. Instead, filter the output of **who** through an **awk** command to eliminate everything except the column of login names. You get no credit for **logs** if you use **ps**, **finger**, or **w**.

(2) Write a one-line shellscript named **irons** that will output the login name of everyone who is running more than one process. Remember to use **tail +2** to chop off the first line output by **ps -Af** (Handout 2, p. 15). Then eliminate everything except the column of login names.

(3) Write a one-line shellscript named **populars** that will output the name of every program of which more than one copy is running. For example, chances are that many people are running the Korn Shell right now. Use **ps -A -o comm** to make **ps** output only the names of the programs without their command line arguments. Remember to use **tail +2** to chop off the first line output by **ps**. After you have printed the shellscript and its output, write a checkmark ✓ by hand next to each program that is a daemon. The name of a daemon ends with a lowercase **d**; look up each candidate in **man** to make sure.



Print a shellscript and its output all on one page

You get no credit for any shellscript unless you print the shellscript and its output on the same page.

```

1$ cd ~/bin                                Go to your personal bin directory.
2$ pwd
    
```

Instead of writing the word **bigfile.txt** over and over again, use the **escape-** in Handout 2, p. 11.

```
3$ cp own2 bigfile.txt      Create a copy of own2 named bigfile.txt.
4$ ls -l bigfile.txt       See how many bytes are in bigfile.txt.
5$ own2 >> bigfile.txt     Append the output of own2 to bigfile.txt.
6$ ls -l bigfile.txt       Did bigfile.txt get bigger?

7$ vi bigfile.txt          Doctor bigfile.txt if necessary.
```

Then print **bigfile.txt**. You can print it from your web browser as in Handout 1, p. 15:

```
8$ chmod 644 bigfile.txt    Turn on all three r's: rw-r--r--
9$ ls -l bigfile.txt

10$ mv bigfile.txt ~/public_html
11$ cd ~/public_html
12$ pwd
13$ ls -l bigfile.txt
```

Then point your browser at

```
http://i5.nyu.edu/~abc1234/bigfile.txt
```

After you print it, read-protect it so that no one can copy it:

```
14$ chmod 600 bigfile.txt   Turn off the last two r's: rw-----
```

Or you can print **bigfile.txt** at an NYU computer center like this:

```
15$ lpr bigfile.txt         $PRINTER eliminated the need for the -P option of lpr.
16$ lpq | more
17$ rm bigfile.txt
```

(1) If you are handing in more than one homework, staple all of them together in order of ascending homework number. Each computer center has a stapler. Print your real name, login name, and section number legibly on the first sheet. Better yet, write them in the comments.

(2) Write the homework number on each shellscript.

(3) Do not hand in cover sheets or folders. Put the cover sheets in the recycling bin near the printer.

(4) Mark a non-working shellscript as “non-working”.

### Change the data by hand to test a pipeline

If the output of the first program in a pipeline (e.g., **who**, **ls -l**, **ps -Af**) does not adequately test the subsequent programs, doctor the output by hand before passing it on.

```
1$ prog1 | prog2 | prog3 | prog4

2$ prog1 > tempfile
3$ vi tempfile
4$ prog2 < tempfile | prog3 | prog4
5$ rm tempfile
```

### Tee connections: pp. 72–73

```
1$ prog1 | prog2 | prog3 | prog4
```

```
2$ prog1 | tee tempfile | prog2 | prog3 | prog4
3$ cat tempfile
4$ rm tempfile
```

**tee** lets us have more than one alternative in the diagram in Handout 2, p. 14. Direct the standard output to a file and to the screen:

```
5$ prog | tee outfile
```

Direct the standard output to two files but not to the screen:

```
6$ prog | tee outfile1 > outfile2
```

Direct the standard output to three files and to the screen:

```
7$ prog | tee outfile1 | tee outfile2 | tee outfile3
```

Direct the standard output to three other screens and to your own:

```
8$ prog | tee /dev/pts/10 | tee /dev/pts/20 | tee /dev/pts/30
```

## wall

```
1$ awk '71176 <= NR && NR <= 71177' $S45/Shakespeare.complete
DEMETRIUS It is the wittiest partition that ever I heard
discourse, my lord.
```

```
2$ cd /usr/sbin
```

```
3$ ls -l wall                                     "write all"; see set-uid bit in Handout 9, p. 11
-r-xr-sr-x  1 root      tty          22476 Jul  5  2012 wall
```

**which wall** doesn't find **wall** because it looks only in the directories in your **\$PATH**. **whereis** will look in a few other directories as well. **which** will output at most only one full pathname; **whereis** will output every full pathname it finds. **find** can search every direction that you have permission to search.

## script

To make separate transcripts of all the input and output,

```
1$ tee infile | prog | tee outfile                input from keyboard, output to screen
```

To interleave the two transcripts into one big file,

```
2$ cd
```

```
3$ script bigfile
```

```
Script started, file is bigfile
```

```
4$ prog
```

*(Your dialogue with the program **prog** takes place here.)*

```
5$ exit
```

*safer than **control-d***

```
Script done, file is bigfile
```

```
6$ cd
```

```
7$ pwd
```

```
8$ ls -l bigfile
```

*Make sure you created the file **bigfile**.*

```
9$ lpr bigfile
```

```
10$ rm bigfile
```

**The -F option of awk: pp. 115–116**

The **F** (“input field separator”) must be uppercase.

```
1$ date
Tue May 28 15:18:34 EDT 2013

2$ date | awk '{print $5}'
EDT

3$ date | awk '{print $5}' | awk -F: '{print $3}'
```

List the TCP port numbers currently in use:

```
4$ netstat -an -f inet -P tcp | tail +5 | awk '{print $1}' |
  awk -F. '{print $NF}' | grep -v '^*$' | sort -n | uniq
22
25
80
111
3306
```

**Let awk draw its standard input from a file**

```
1$ more /etc/passwd
root:x:0:0:root@i5:/root:/usr/bin/bash
daemon:x:1:1::/
bin:x:2:2::/usr/bin:
```

Let’s output the loginname of everyone who has an account on our machine. The **-F:** option of **awk** tells **awk** that the fields of this file are separated by colons (pp. 115–116). **awk** doesn’t need the **<** in line 2; see Handout 2, p. 23.

```
2$ awk -F: '{print $1}' < /etc/passwd | more
3$ awk -F: '{print $1}' /etc/passwd | more
root
daemon
bin
```

How many people belong to group 15? The group number is the fourth field in each line of **/etc/passwd**, so tell **awk** to print **\$4** instead of **\$1**.

```
5$ grep 15 /etc/passwd | wc -l           would count abc15 or group 150 too
6146

6$ awk -F: '{print $4}' /etc/passwd | grep '^15$' | wc -l
6144
```

How many people have home directories that are descendants of **/home1**?

```
7$ awk -F: '{print $6}' /etc/passwd | more
/home1/v/vrc1

8$ awk -F: '{print $6}' /etc/passwd | awk -F/ '{print $2}' | more
home1
```

You can split a long command after a pipe. See pp. 107–108; Handout 2, p. 14, lines 81–83; Handout 2, p. 16, lines 25–27.

```
9$ awk -F: '{print $6}' /etc/passwd | awk -F/ '{print $2}' |
grep '^home1$' | wc -l
6144
```

**comm: p. 107**

**mail** and **mailx** store your letters in a file named `/var/mail/abc1234`; the `$MAIL` variable holds the name of this file. See Handout 2, p. 13, lines 23–24. If the file `/var/mail/abc1234` does not exist then you have no letters waiting for you.

The **roster** program outputs the loginname of everyone in the class:

```
1$ ~mm64/bin/roster 45 | more
yb610
ic297
bc1478
sum208
mm64
```

A `~` (tilde) in a shellsript is the full pathname of the home directory of the person running the shellsript.

**comm -12 file1 file2** outputs the lines that are in both **file1** and **file2**. You must always give exactly three command line arguments to **comm**. The last two will be filenames.

You can remove two or more files with a single **rm**. See Handout 2, p. 5.

```
http://i5.nyu.edu/~mm64/INFO1-CE9545/src/recipients
#!/bin/ksh
#Output the login name of everyone in the class who has mail waiting
#for them.
#Sample use: recipients

~mm64/bin/roster 45 | sort > ~/inclass
ls /var/mail > ~/hasmail

comm -12 ~/inclass ~/hasmail
rm ~/inclass ~/hasmail
```

After the following shellsript is finished, you'll still be in the same directory where you started. A **cd** in a shellsript does not drag you along with it.

```
http://i5.nyu.edu/~mm64/INFO1-CE9545/src/recipients2
#!/bin/ksh
#Output the login name of everyone in the class who has mail waiting
#for them.
#Sample use: recipients2

cd
~mm64/bin/roster 45 | sort > inclass
ls /var/mail > hasmail

comm -12 inclass hasmail
rm inclass hasmail
```

```
2$ recipients
```

```
aw1312
```

```
ic297
```

```
mm64
```

```
rz665
```

```
sum208
```

```
3$ cd /tmp
```

```
4$ pwd
```

```
5$ ls -ld
```

```
drwxrwxrwt 46 root sys 3765 May 28 15:18 .
```

```
6$ ls -ltr | head -5
```

*Handout 1, p. 10 for -t and -r*

```
total 1248
```

```
-rw----- 1 webservd webservd 40 May 15 18:34 jCGztTA8I2
```

```
-rw----- 1 webservd webservd 269257 May 15 18:34 QDa0dtlhMd
```

```
-rw----- 1 mm64 users 0 May 28 15:18 18277.ps
```

```
-rw----- 1 mm64 users 0 May 28 15:18 18293
```

```
7$ date
```

```
Tue May 28 15:18:37 EDT 2013
```

See the process id number (PID): p. 33

```
-----http://i5.nyu.edu/~mm64/INFO1-CE9545/src/pid-----
#!/bin/ksh
#Output the process id number; see p. 146.

echo $$
echo $$ #the same number
```

—On the Web at

<http://i5.nyu.edu/~mm64/INFO1-CE9545/src/pid.c>

```
1 /* This program is written in C. */
2
3 #include <stdio.h> /* declaration for printf function */
4 #include <stdlib.h> /* definition for EXIT_SUCCESS macro */
5 #include <unistd.h> /* declarations for getpid and getppid functions */
6
7 int main()
8 {
9     printf("%d\n", getpid()); /* my process id number, i.e. $$ */
10    printf("%d\n", getppid()); /* my parent's process id number, p. 34 */
11
12    return EXIT_SUCCESS;
13 }
```

To compile (i.e., translate) and run a C program, use the minus lowercase O option to tell the C compiler `gcc` (`cc` on other systems) to create executable file named `~/bin/pid`. (Without the `-o` and the following filename, the name of the executable file will default to `a.out`.)

```
1$ man gcc
2$ gcc -o ~/bin/pid pid.c
```

```
3$ ls -l ~/bin/pid          Verify that you have permission to execute the new file.
-rwx-----  1 mm64      users          7148 May 28 15:18 /home1/a/abc1234/bin/pid
```

```
4$ pid                    Execute the new file; direct its output to screen.
19419                  the PID number of the C program
19418                  the PID number of the parent of the C program
```

### ▼ Homework 3.8: find duplicate programs

We have at least two versions of **grep**:

```
1$ ls -l /bin/grep /usr/xpg4/bin/grep
-r-xr-xr-x  1 root      bin          21660 Jul  5  2012 /bin/grep
-r-xr-xr-x  3 root      bin          22348 Aug 27  2012 /usr/xpg4/bin/grep
```

Write a shellscript named **two** that will output every name that is common to things in the two directories **/bin** and **/usr/xpg4/bin**. (In Handout 2, p. 17, we saw that two directories have versions of **tr**.) There is no reason to give the **-l** option to **ls** in this Homework. As of May 28, 2013, the correct output is shown below. To save paper, I split the shellscript's standard output into four columns by piping it into **pr -4 -i ' '1 -t** (Handout 4, pp. 1–2), but don't do that.

alias	ed	jobs	sed
ar	edit	kill	sh
at	egrep	ln	sort
awk	env	ls	stty
basename	ex	m4	tail
batch	expr	more	test
bg	fc	mv	tr
cd	fg	nice	type
chgrp	fgrep	nl	ulimit
chown	file	nm	umask
command	find	nohup	unalias
cp	getconf	od	vedit
crontab	getopts	patch	vi
ctags	grep	pfsh	view
date	hash	pr	wait
df	id	read	who
du	ipcs	rm	

▲

### Sample shellscript: delinquent

**comm -23 file1 file2** outputs the lines that are in **file1** but not **file2**. You can split a long command after a pipe: see Handout 3, p. 21, line 9.



```

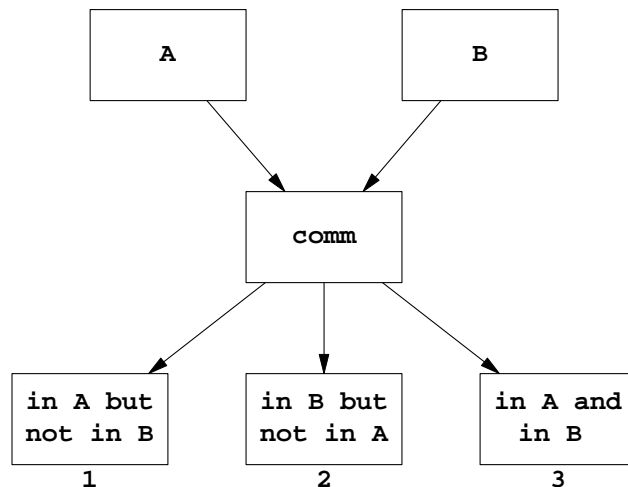
http://i5.nyu.edu/~mm64/INFO1-CE9545/src/delinquent
#!/bin/ksh
#Output the login name of everyone in the class who owns no file in
#the directory ~mm64/public_html/INFO1-CE9545/homework.
#Sample use: delinquent

~mm64/bin/roster 45 | sort > ~/inclass

ls -l ~mm64/public_html/INFO1-CE9545/homework | tail +2 |
  awk '{print $3}' | sort | uniq > ~/didhomework

comm -23 ~/inclass ~/didhomework
rm ~/inclass ~/didhomework

```



### Two requirements for every shellscript

(1) You get no credit for any shellscript in which you accidentally maim the data with an unnecessary `tail +2`, or in which you accidentally process the header as a line of data because you forgot the `tail +2`.

(2) You get no credit for any shellscript unless you can run it in your home directory `/home1/a/abc1234` as well as in the parent of your home directory `/home1/a`:

```

1$ cd
2$ pwd
3$ classmates

```

*Run `classmates` in your home directory.*

```

4$ cd ..
5$ pwd
6$ classmates

```

*Run `classmates` in the parent of your home directory.*

### ▼ Homework 3.9: classmates

Write a shellscript named `classmates` that will output the login name of everyone in this class who is currently logged in. When you run this program, it will always output at least your own login name, and possibly others. Output nothing except what I ask for: no titles, explanatory sentences, etc.

Use the method shown above. Create two temporary files: `~/inclass` will be a list of the login names of everyone in the class, and `~/loggedin` will be a list of the login names of everyone who is logged in. You get no credit if you put any name into `~/loggedin` more than once. Feed the two files to `comm` to output the lines that are in both files. Remove both of your temporary files with a single `rm`. You

get no credit if you use **finger**, **ps**, or **w**.

↘ Extra credit. Write **classmates** with only one temporary file and without **comm**. There's a way to do this using only the material we've covered so far. Your shellscript must output the correct result even if someone not in the class is using two or more terminals.

You get no credit if you use **cat** or **<** in this shellscript. Handout 2, p. 24, ★ shows how to avoid them.

↘↘ For space cadets only. Write **classmates** without any temporary file and without **comm**. Use the parentheses on p. 72, but write no more pairs of parentheses than are strictly necessary. See the warning at the end of the previous paragraph.

You get no credit for Homework 3.9 if you hand in more than one version. Therefore you can do only one of the extra credit versions, and if you do so you can not hand in the non-extra credit version. Do not use these extra credit algorithms in any other homework—you will get no credit if you do.



### head and tail: pp. 19–20

**head** and **tail** output a copy of part of their input. **-10** is the default argument. There are only three possibilities because there is no **head +10**.

```
1$ head -10           Output only the first 10 lines; 10 is the default.
2$ tail -10           Output only the last 10 lines; 10 is the default.
3$ tail +10           Output every line from line 10 onwards. Why does tail +1 do nothing?
```

```
4$ head /etc/passwd           Output the first 10 lines of the file.
```

```
5$ last abc1234 | head           When were the 10 last times abc1234 logged in?
6$ dmesg | tail                 diagnostic messages from the operating system
7$ ls -lt ~mm64/public_html/INFO1-CE9545/homework | tail -1 oldest file.
```

```
8$ cd /var/apache2/2.2/logs
```

```
9$ pwd
```

```
10$ tail access_log
```

```
11$ tail access_log           Different output; use the r or control-p in Handout 2, p. 11.
```

```
12$ grep abc1234 access_log | tail           Be patient.
```

Always give the **-n** (or **-nr**) option to **sort** when sorting numbers:

```
13$ sort           alphabetical sort by mistake
```

```
1
```

```
2
```

```
10
```

```
control-d
```

```
1
```

```
10
```

```
2
```

To see what goes wrong when you sort numbers alphabetically, try **ls -l /dev/pts | more**

Output the size in bytes of the largest file in the current directory.

```
14$ ls -l | tail +2 | awk '{print $5}' | sort -nr | head -1
```

Output the name of one of the largest files in the current directory. See p. 116 for the comma in the argument of **awk**. Change **\$9** to **\$NF** (p. 115). See pp. 19, 106, and Handout 1, p. 3, line 38 for **+4nr**. Linux **ls** has an uppercase **-S** option for size order; combine it with **-l** as **-lS**.

```
15$ ls -l | tail +2 | awk '{print $5, $9}' | sort -nr | head -1 | awk '{print $2}'
16$ ls -l | tail +2 | sort +4nr | head -1 | awk '{print $9}'
```

### ▼ Homework 3.10: chop off the header line(s) (not to be handed in)

How many lines of header material does each of the following commands output before the lines of data? Use `tail` to chop off the header without maiming the data.

```
1$ who
2$ ls
3$ ls -l
4$ ps -Af
5$ df
6$ w
```

```
7$ cal 5 2013
8$ lpq
```

```
9$ /usr/sbin/arp -a | cat -n | head           see if there are empty lines
10$ netstat -f inet -P tcp | cat -n | head
11$ nm -D /lib/libc.so | cat -n | head      shared object file containing C Standard Library
```

▲

### ▼ Homework 3.11: a shellscript that accepts standard input

The file `$$45/localhosts` contains the IP address of each host on our local network. (At least it contains the 126 IP addresses that could belong to the hosts on the network. Some of these addresses may be currently unused.) Each address consists of four “octets”, separated by three dots. The addresses are listed in ascending numerical order.

```
1$ head -3 $$45/localhosts
128.122.109.1
128.122.109.2
128.122.109.3
```

The program `$$45/mypping` broadcasts a *ping* (a request for a response) to the local network and outputs the IP address of each host that answers. They are listed in order of how promptly they answer, with the fastest on top. See `ping(1M)`.

```
2$ $$45/mypping | head -3
128.122.109.1
128.122.109.2
128.122.109.3
```

Write a shellscript named `not_responding` that will output the IP addresses of the hosts that did *not* answer the ping. This will be our first shellscript that accepts standard input. It will read a list of IP addresses from the standard input, and will then output the ones that do not appear in the output of `mypping`.

```
3$ not_responding < $$45/localhosts | head -3
128.122.109.100
128.122.109.101
128.122.109.102
```

Like `classmates`, `not_responding` will feed two lists of lines into `comm`. The first list will be a temporary file named `~/locals` containing all the IP addresses in the standard input, one per line and sorted into alphabetical order. The second list will be a temporary file named `~/responding` containing

all the IP addresses in the standard output of **mypping**, one per line and sorted into alphabetical order.

Here is a sketch of the shellsript. The input fed into the shellsript in the command line that launches the shellsript (the `< $S45/localhosts` in the above line 3) will be fed into the first program in the shellsript that accepts standard input but whose source of standard input is not specified in the shellsript (the **sort** in the following box).

```
#!/bin/ksh
#Output the IP address of every local host that did not answer a ping
#(every IP address in the input that is not in the output of mypping).

sort > ~/locals #Sort the standard input and save it in a file.
create the other file, ~/responding
feed ~/locals and ~/responding into comm
remove the two temporary files ~/locals and ~/responding
```

Do not say in the shellsript where the standard input comes from or where the standard output goes to. You get no credit if the words **localhosts**, **\$1**, **\$\***, or **\$@** appear in the shellsript.

You get no credit if you use **cat** in this shellsript. Handout 2, p. 24, ★ shows how to avoid it.

You can use this shellsript (or any other Unix program that accepts standard input) in any of the following ways:

```
4$ not_responding < infile Assumes the file is in the current directory; r bit must be on.
5$ not_responding < /another/directory/infile

6$ prog | not_responding
7$ cat infile1 infile2 infile3 | not_responding

8$ not_responding
9$ not_responding < /dev/pts/10
10$ not_responding < /dev/null
```

You can also redirect **not\_responding**'s standard output in all the ways shown in Handout 3, p. 17, lines 21–28.

Two improvements:

(1) **not\_responding** does not need to create and destroy two temporary files; it can get by with only one. Let's get rid of **~/responding**. Pipe the sorted output of **mypping** dictionary directly into **comm**, and write a dash in place of **~/responding** as the last argument of **comm**. See the note about the dash - in **comm(3)**.

<pre>#before: prog1 &gt; ~/temp1 prog2 &gt; ~/temp2 comm -23 ~/temp1 ~/temp2 rm ~/temp1 ~/temp2</pre>	<pre>#after: prog1 &gt; ~/temp1 prog2   comm -23 ~/temp1 - rm ~/temp1</pre>
-------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------

(2) The two input files fed to **comm** must be in alphabetical order. The output produced by **comm** will therefore also be in alphabetical order. But the user would rather see these IP addresses in ascending numerical order. (If all the numbers have the same number of digits, alphabetical order will just happen to be the same as numerical order. But you won't always be so lucky.)

Have **not\_responding** pipe the output of **comm** into the following pipeline. A numeric **sort** would interpret the dots as a decimal points; to avoid this confusion, the two **tr**'s temporarily hide and restore them.

```
tr . ' ' | sort +0n +1n +2n +3n | tr ' ' .
```

The output will now always be in ascending numerical order.

```
11$ not_responding < $S45/localhosts | head -3
128.122.109.4
128.122.109.5
128.122.109.6
```



## Uniform Resource Locators (URLs) in the World Wide Web

A *resource* is a file that can be displayed or copied, a program that can be executed, etc. For simplicity, we'll assume that the resource is a file. A *URL* is a one-line string of characters that tells your web browser how and where to get a resource for you.

Getting a resource from another host (i.e., computer on the Internet) requires the cooperation of two programs. The program running on your host is called the *client*, and the one on the host containing the resource is called the *server*. The client and server must agree on the *protocol* they will use, which is a set of rules and formats for data communication. Each protocol has a name: **http** (Hypertext Transport Protocol), **ftp** (File Transfer Protocol), etc.

The first part of a URL is the name of the protocol that the client and server will use when sending the resource from the server to the client.

```
http://i5.nyu.edu           i5.nyu.edu home page
ssh://i5.nyu.edu           log in to i5.nyu.edu
```

After the protocol comes a colon, two slashes, the name of the host that holds the file, the name of the directory that holds the file, and the name of the file.

(1) In a URL, a loginname with a tilde in front of it stands for the full pathname of the **public\_html** subdirectory of the home directory of that person. (The superuser sets this up with the **UserDir** in Handout 3, p. 8; it may be different on other hosts.) For example, the URL

```
http://i5.nyu.edu/~abc1234/index.html
```

refers to the file `/home1/a/abc1234/public_html/index.html` on the host **i5.nyu.edu**.

(2) If the name of the directory in the URL is not a loginname with a tilde in front of it, the prefix `/var/apache/htdocs/` is added to the name of the directory. (The superuser sets this up with the **DocumentRoot** in Handout 3, p. 8; it may be different on other hosts.) For example, the URL

```
http://i5.nyu.edu/manual/index.html
```

refers to the file `/var/apache/htdocs/manual/index.html` on the host **i5.nyu.edu**.

If the URL contains no directory at all, the directory is assumed to be `/var/apache/htdocs`. For example, the URL

```
http://i5.nyu.edu/index.html
```

refers to the file `/var/apache/htdocs/index.html` on the host **i5.nyu.edu**. (There are web pages in many languages in that directory.)

(3) If the URL contains no filename, the filename is assumed to be **index.html**. (The superuser sets this up with the **DirectoryIndex** in Handout 3, p. 7; it may be different on other hosts.) For example, the URL

```
http://i5.nyu.edu/~abc1234/           Can omit the trailing slash.
```

refers to the file `/home1/a/abc1234/public_html/index.html` on the host **i5.nyu.edu**, and the URL

```
http://i5.nyu.edu/           Can omit the trailing slash.
```

refers to the file `/var/apache/htdocs/index.html` on the host **i5.nyu.edu**.

(4) If the URL contains no filename and the directory contains no file named `index.html`, the browser will display an `ls -l` of the directory. For example, the URL

`http://i5.nyu.edu/~mm64/INFO1-CE9545/src/` *Can omit trailing slash.*

will list the directory `/home1/m/mm64/public_html/INFO1-CE9545/src` on the host `i5.nyu.edu`.

**A relative URL is like a Unix relative pathname.**

(5) In a document whose own URL starts with `http://i5.nyu.edu`, you can omit the leading `http://i5.nyu.edu` from any URL. For example, you can write either of the following in your home page `http://i5.nyu.edu/~abc1234/`:

```
http://i5.nyu.edu/~mm64/INFO1-CE9545/
      /~mm64/INFO1-CE9545/
```

In fact, in a document whose URL starts with `http://i5.nyu.edu/~abc1234/`, you can omit the leading `http://i5.nyu.edu/~abc1234/` from any URL. For example, you can write either of the following in your home page `http://i5.nyu.edu/~abc1234/`:

```
http://i5.nyu.edu/~abc1234/hobbies.html
      hobbies.html
```

### ▼ Homework 3.12: put links into your home page

Put links such as the following into your `~/public_html/index.html` file:

```
I like
<A HREF = "http://www.nyt.com/">The New York Times</A>
and
<A HREF = "http://www.wikipedia.org/">Wikipedia</A>.
```

I like [The New York Times](http://www.nyt.com/) and [Wikipedia](http://www.wikipedia.org/).

▲

### ▼ Homework 3.13: Look at the foreign language files

Look at the files in the directory `/var/apache/htdocs`. For example, to see the file `/var/apache/htdocs/index.html.es`, point your browser at `http://i5.nyu.edu/index.html.es`.

▲

□