Summer 2013 Handout 1

▼ Homework 1.1: interesting but harmless things to do (not to be handed in)

The *operating system* is the thing that invites you to run a program, i.e., to launch an application. Most operating systems (Microsoft Windows, Apple Macintosh) do this by displaying a desktop with icons and menus. But a desktop has never been an integral part of Unix. To run a program, the Unix professional austerely types a *command line* in response to a *prompt.* The first or only word on the command line is the name of the program to be run.

The prompt in these Handouts is a number and a dollar sign (**1$**), but the prompt in the screen that occupies your Unix window will probably be something else. At the end of each command line, press the big **Enter** key on PC, **return** on Mac. Unless you say otherwise, the program's output will be directed to the screen so you can see it immediately. When the program is finished, the prompt will reappear and you can run another program.

Page numbers in the Handouts refer to *The Unix Programming Environment* by Kernighan & Pike.

```
1$ date                         and time; p. 4
Tue May 28 15:17:05 EDT 2013
```

In the Handouts, **abc1234** stands for your login name. Your actual login name is listed at **http://i5.nyu.edu/~mm64/common/students.html**. Some versions of Unix do not have **whoami**.

```
2$ whoami                       See your login name.
abc1234
```

date and **whoami** output only one line, but **who** may output many. It lists the login name of everyone who is logged in, one name per line. In Unix, all data is presented one item per line.

```
3$ who                          p. 5
```

who has no ability to display the people in alphabetical order. They didn't build this facility into **who** because there is a separate program, named **sort**, to sort the lines. We will run both programs simultaneously, feeding the output of **who** into **sort** by means of the pipe symbol **|**. See p. 31 and **ksh**(1) p. 1.

```
4$ who | sort                   alphabetical order; p. 19 for sort
```

The pipe is above the backslash on your keyboard. Sometimes the pipe appears on the screen with a small gap in the middle. But when you write the pipe, don't write the gap: just draw a vertical stroke. The *white space* (blank(s) and/or tab(s)) before and after a pipe is optional, but put it in for legibility.

Data always travels from left to right through a pipe. In other words, the name of the producing program goes to the left of the pipe; the consuming program goes to the right. No human eye sees the raw (i.e., unalphabetized) output of the above **who**: it is directed through the pipe into the **sort**. But the output of the **sort** was not directed anywhere, so by default it comes out onto the screen.

A series of two or more programs connected by pipes is called a *pipeline.* This is the characteristic way to use Unix, because to get your job done you'll almost always have to run two or more programs.

If **who** (or any Unix program) outputs more lines than will fit on the screen, they will scroll off the top faster than you can read them. So instead of directing the output of **who** directly to the screen, pipe it there through another program named **more**. **more** will accept the lines of input and dole them out onto the screen, 23 lines at a time, every time you press the space bar. To dole out the lines one at a time, press

RETURN repeatedly. Other systems may have **pg** or **less** instead of (or in addition to) **more**; the textbook has **p** on pp. 15, 180. Some versions of **more** (and **man**) go back up with **control-b** or **control-u** instead of **b**.

```
5$ who | more                 space bar to advance, b to go back up, q to quit early
6$ who | sort | more

7$ w                          See what program each person is running.
8$ w | sort
9$ w | sort | more

10$ finger                    See the login name and real name of everyone logged in.
11$ finger | sort
12$ finger | sort | more
```

Sometimes you have to type more than just the name of a program. The additional words or numbers that some programs demand are called *command line arguments* (p. 14). There must be white space before each argument.

```
13$ finger mm64               mm64 is my login name.
14$ fingermm64                It thinks you're trying to run a program named fingermm64.
15$ finger abc1234            Try your login name.

16$ cal 7 1955                Two arguments: see what day of the week you were born on, p. 133.
17$ cal 7 55                  the reign of Nero
18$ cal 9 1752                switch from Julian calendar to Gregorian: the Y2K of the 18th Century

19$ cal 2013                  One argument: the whole year won't fit on the screen.
20$ cal 2013 | more
21$ cal                       No arguments: the current month, different from p. 134

22$ whatis cal                one-line summary
23$ whatis finger
24$ whatis less
25$ whatis true               John 18:38
26$ whatis God
27$ whatis whatis
```

A command line argument that starts with a dash (sometimes a plus) is called a command line *option,* because it is optional. The **-s** in lines 30−31 is a *Solarisism,* needed only in Solaris.

```
28$ man cal                   cal documentation: space bar to advance, b to go back up, q to quit
29$ man man                   Manfred Mann, Do Wah Ditty (1964); p. 11. The -s is not needed here.
30$ man -s 3c printf          section 3c of the manual; see printf(3c).
31$ man -s 3c++ cout          documentation about the language C++ too
```

An asterisk used as an argument of **expr** means multiplication. Unfortunately, the asterisk has many other meanings in Unix (p. 27). To turn off these unwanted meanings, surround the asterisk with the 'single quotes' on p. 28. Do this for any argument that contains crazy looking characters: **#$@^!%&**

```
32$ expr 2 + 3                Must have white space before each arg. Try expr 2+3 w/o white space.
33$ expr 2 * 3                Why does the asterisk blow up?
34$ expr 2 '*' 3              This is how you have to multiply.
```

grep is the Unix search command (p. 18):

```
35$ grep mania  /usr/dict/words                 25,146 words, one per line in alphabetical order
36$ grep phobia /usr/dict/words                 /usr/share/dict/words on Mac OS X
37$ grep esque  /usr/dict/words
38$ grep anti   /usr/dict/words
39$ grep anti   /usr/dict/words | more          space bar to advance, q to quit
```

The argument of the grep, with its punctuation marks, is an example of a *regular expression.*

```
40$ grep '^mania'     /usr/dict/words           Only the caret ^ (p. 102) needs to be in quotes,
41$ grep '^'mania     /usr/dict/words           so this would also work but is harder to type.

42$ grep 'ism$'       /usr/dict/words | more    p. 102 for $
43$ grep '^in.*able$' /usr/dict/words | more    p. 103 for . and *
44$ grep '^...u.$'    /usr/dict/words | more    crossword puzzle
45$ grep  '...u.'     /usr/dict/words | more    omit the anchors
46$ grep 'mania' /usr/dict/websters             234,936 words, one per line in alphabetical order
```

In a command line, a loginname with a leading tilde ~ stands for the full pathname of that person's home directory (Handout 1, p. 3). For example, my login name is **mm64**, so **~mm64** stands for the full pathname of my home directory **/home1/m/mm64** (home one). The continuation character is \, just like ^ in the Windows Command Prompt window. Type nothing after the \, not even a blank or tab; see p. 77. There must be whitespace before each argument, so there must be whitespace before the \ or before the **~mm64**.

```
47$ grep 'that is the question' \
    ~mm64/public_html/INFO1-CE9545/src/Shakespeare.complete  175,641 lines

48$ df -k | more                        "Disk free": see how full each disk is, p. 67.
49$ df -k | grep -v '^i5pool' | more    every line except the ones that start with i5pool
Filesystem           1024-blocks        Used   Available Capacity  Mounted on
rpool/ROOT/solaris      20971486     3730112    13979227    22%    /
rpool/ROOT/solaris/var     8388608      600253     7780014     8%    /var
swap                    8388608      734392     7654216     9%    /system/volatile
swap                    8388608      734392     7654216     9%    /tmp

50$ df -k | grep -v '^i5pool' | sort | more      Sort in alphabetical order by default
51$ df -k | grep -v '^i5pool' | sort -r | more   reverse alphabetical order; p. 19 for -r
52$ df -k | grep -v '^i5pool' | sort +4n | more  in order of increasing capacity; p. 106 for +4n
53$ df -k | grep -v '^i5pool' | sort +4nr | more decreasing capacity
```

The most interesting thing to **echo** is the content of a *variable,* which is a container holding a word, a number, or any string of characters. For the time being, write a **$** in front of the variable's name.

```
54$ echo hi there           echo means "print" in this language; pp. 27, 79.
55$ echo My loginname is $LOGNAME    uppercase; echo $USER on other systems
56$ echo My home directory is $HOME  full pathname of your home directory; p. 36
57$ echo My shell is $SHELL
58$ echo $PATH              the names of the directories that hold programs: p. 37; ksh(1) pp. 16, 27
59$ echo $TERM              What kind of terminal does the computer think you have?
60$ echo $PS1               PS one; echo $prompt on other systems: ksh(1) pp. 16, 19
61$ env | sort -df | more   See your environment variables: pp. 38, 91, 199; environ(5); p. 106 for -df

62$ fortune          /usr/games directory empty at NYU, but maybe not empty on your machine.
63$ fortune          try it again
64$ r                repeat the previous command: ksh(1) pp. 4, 27;  !! in csh
```

```
65$ passwd                  Change your password on other systems.
Old password: 123456        It won't echo what you type.
New password: Bacall8?      On our system, go to http://start.nyu.edu/
Re-enter new password: Bacall8?
```

Each homework ends with a triangle pointing up.

▲

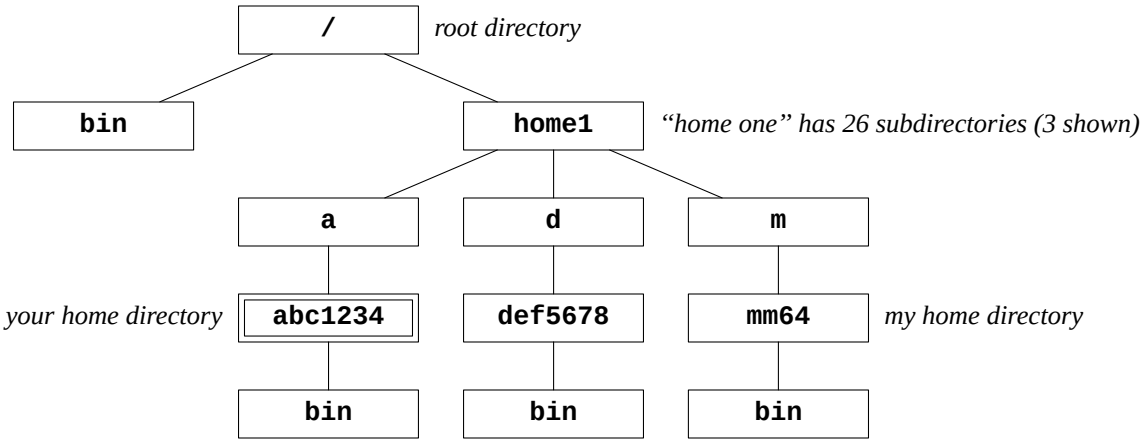
The tree of directories

A *file* is a document, program, image, video, etc. (It's called a ''document'' in Windows and Mac.) The part of the computer that holds the files is divided into sections called *directories.* (They are called ''folders'' in Windows and Mac.) Each directory has a name. For example, your *home directory* is the one whose name is your login name. It's the double box in the diagram below.

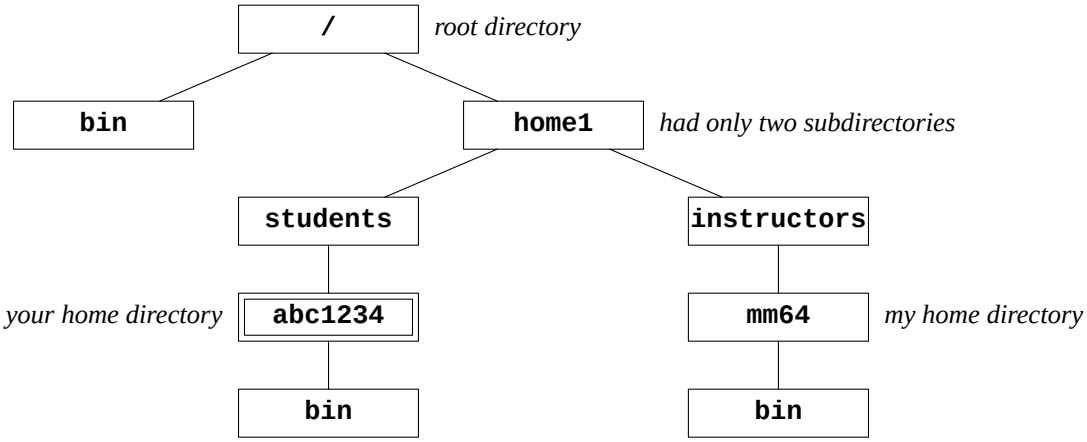
Whenever you use the computer, you are located in some directory. When you log in, for example, your initial location will always be your home directory. The directories are arranged in a tree, like a family tree. You can move to another directory with the **cd** command (p. 25; **ksh**(1) pp. 39−40). If you forget where you are, the **pwd** command (''print working directory'', pp. 21−22) outputs the name of the directory you are in, which is called your *current directory.*

```
1$ pwd                      when you log in
/home1/a/abc1234            home one

2$ pwd                      when I log in
/home1/m/mm64
```

The tree currently looks like this. Each rectangle is a directory.



But the tree used to look like this:

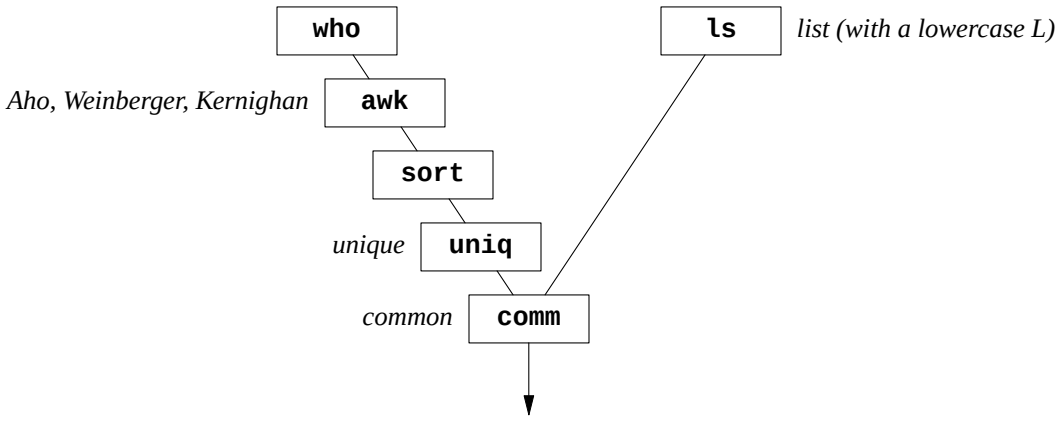



```
3$ pwd
/home1/instructors/mm64
```

For an outline of the tree of directories, see pp. 63−65 and print the **filesystem**(5) manual page (**hier**(1) on other systems) as in Handout 1, p. 16.

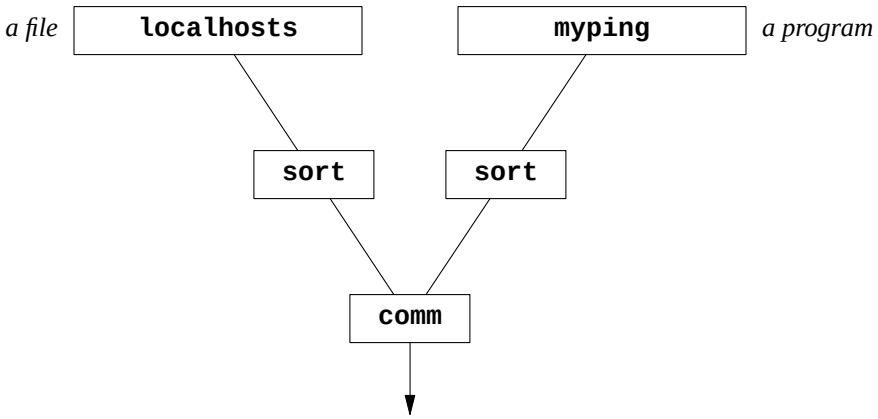
Preview of how we will combine the Unix tools

ls can output the names of all the subdirectories of a directory. Of course, you have to tell it which directory you're interested in.



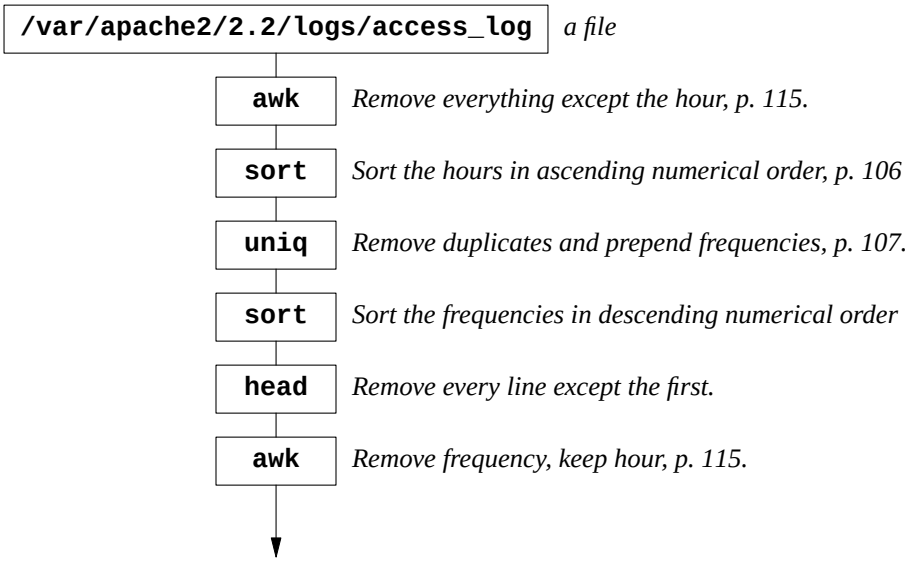

```
1$ who | more
esh322     pts/3        Apr  7 22:55     (172-26-196-152.dynapool.nyu.edu)
mm64       pts/1        May 28 14:37     (3a_imac_03.ndlab.its.nyu.edu)
elliott    pts/4        May 28 11:40     (njoerd.es.its.nyu.edu)
mm64       pts/8        May 28 14:39     (3a_imac_03.ndlab.its.nyu.edu)
mm64       pts/9        May 28 15:11     (3a_imac_03.ndlab.its.nyu.edu)
```

```
2$ ls | more              list the directory /home1/a in alphabetical order: lowercase LS
aa1762
aa2429
aa2543
aa3329
aa947                     etc.
```




```
3$ head -3 ~mm64/public_html/INFO1-CE9545/src/localhosts
128.122.109.1
128.122.109.2
128.122.109.3

4$ ~mm64/public_html/INFO1-CE9545/src/myping | head -3
128.122.109.2
128.122.109.1
128.122.109.3
```

Transform the data as it flows through a pipeline



Here are the first three and last three lines of the constantly growing file **`/var/apache2/2.2/logs/access_log`**. The seven fields on each line are described under ‘‘Common Log Format’’ in **`http://httpd.apache.org/docs/1.3/mod/mod_log_config.html`**.

```
::1 - - [16/Jul/2012:16:08:02 -0400] "GET / HTTP/1.1" 200 44
128.122.170.164 - - [16/Jul/2012:16:08:21 -0400] "GET / HTTP/1.1" 200 44
128.122.170.164 - - [16/Jul/2012:16:08:21 -0400] "GET /favicon.ico HTTP/1.1" 200 1406
```

```
128.122.108.95 - - [28/May/2013:15:12:19 -0400] "GET / HTTP/1.1" 200 6311
100.2.35.247 - - [28/May/2013:15:13:02 -0400] "GET /~ajg494/Final/paperbackground.jpg HTTP/1.1" 20
100.2.35.247 - - [28/May/2013:15:13:53 -0400] "GET /~ajg494/Final/paperbackground.jpg HTTP/1.1" 20
```

What is the IP address of the host that sent the most requests? What was the web page on **`i5.nyu.edu`** that was most frequently requested? What was the biggest page that was requested and what was its size? What was the busiest month? What is the busiest hour?

```
16                      The web server was launched shortly after 3 a.m.
16
16                      Many web pages were served during the first hour.
```

```
00                      The midnight lines come first because they’re numerically smallest.
00
00                      Many lines have midnight.
```

```
36093 00                Now only one line has midnight.
28889 01
23582 02
14714 03
10860 04
11761 05
12652 06
19658 07
23072 08
25964 09
35933 10
41715 11
42322 12
46765 13
50496 14
52698 15
56840 16
50245 17
52838 18
50693 19
43958 20
42358 21
40979 22
40039 23
```



```
56840 16
52838 18
52698 15
50693 19
50496 14
50245 17
46765 13
43958 20
42358 21
42322 12
41715 11
40979 22
40039 23
36093 00
35933 10
28889 01
25964 09
23582 02
23072 08
19658 07
14714 03
12652 06
11761 05
10860 04
```

```
56840 16
```

```
16
```

Preview of regular expressions: pp. 102−105

Do not type the following commands yet—they will hang because no source of input was specified.

1$ grep '212' *every line that contains* **212***; fooled by* **718-234-5212**
2$ grep '^212' *every line whose first three characters are* **212***; fooled by* **(212) 234-5678**

3$ grep '[^0-9]*212' *fooled by* **718-234-2125**
4$ grep '^[^0-9]*212' *no longer fooled: every line whose first three digits are* **212**
5$ grep '^[^A-Za-z0-9]*212' *every line whose first three alphanumerics are* **212**

6$ egrep '^[^0-9]*1?[^0-9]*212' *allow one optional* **1** *among the characters before the* **212**
7$ perl -ne 'print if /^[^0-9]*1?[^0-9]*212/;' *same thing in Perl*
8$ perl -ne 'print if /^\D*1?\D*212/;' *same thing in Perl*

Let's build up the regular expression **[^0-9]*** in the above line 3:

[0123456789] *Look for any line that contains a character that is a digit.*
[0-9] *An easier way to look for any line that contains a character that is a digit.*
[^0-9] *Look for any line that contains a character that is not a digit.*
[^0-9]* *Look for any line that contains zero or more consecutive characters that are not digits.*

Travel around the tree: pp. 25−26

```
1$ pwd                      print working directory: pp. 21−22
/home1/a/abc1234

2$ cd /home1/m/mm64         "change directory"; or cd ~mm64; home one; dead silence
3$ pwd                      Your cancelled check is your receipt.
/home1/m/mm64

4$ cd                       Go back to your home directory.
5$ pwd
/home1/a/abc1234

6$ cd ..                    Go up one level: p. 25; intro(2) p. 11. Space before, but not between, the dots.
7$ pwd
/home1/a

8$ cd ..                    Go up another level.
9$ pwd
/home1

10$ cd /                    Go directly to the root directory at the top of the tree.
11$ pwd
/

12$ cd ..                   What will happen now? What happened to Icarus?
13$ pwd
```

Harmless but useless: p. 25; intro(2) p. 11.

```
1$ pwd
2$ cd .                     Dot stands for the name of your current directory.
3$ pwd
```

Full vs. relative pathnames

Let's say you have traveled to the directory **/home1/a**. In other words, you have changed your *current directory* to **/home1/a**.

```
1$ cd /home1/a
2$ pwd                      Verify that you arrived there.
```

Suppose you now want to travel two levels down to the **bin** directory which is a subdirectory of your home directory.

```
3$ cd /home1/a/abc1234/bin
4$ pwd                      Verify that you arrived there.
```

Instead of typing the full name **/home1/a/abc1234/bin**, you can omit the underlined part:

```
5$ cd abc1234/bin
6$ pwd                      Verify that you arrived there.
```

A full name such as **/home1/a/abc1234/bin** is called an *full* (or *absolute*) *pathname;* a shortened one such as **abc1234/bin** is called a *relative pathname.* An absolute pathname always begins with a slash; a relative pathname never does. See p. 25.

Whenever you mention the name of a directory or file whose full pathname would begin with the name of the current directory, you can always omit the name of the current directory (and the following slash) from the start of the absolute pathname, yielding a relative pathname.

The output of ls -l: pp. 13−14

With no arguments, **ls** lists the files and directories in your current directory:

```
1$ ls | more                    "list": lowercase LS
curly
homework
larry
moe
```

The **-l** option marks files with a leading **-** and directories with a leading **d**:

```
2$ ls -l | more                 "long" (i.e., verbose): minus lowercase L

-rw-r--r--   1 abc1234     users   3395 May 28 13:46 curly
drwxr-x---   2 abc1234     users   8192 May 28 18:04 homework
-rwxr-xr-x   1 abc1234     users    616 May 28 14:47 larry
-r--r--r--   1 abc1234     users     41 May 28 17:12 moe

3$ ls -l | grep '^-' | more     List only the files in the current directory.
4$ ls -l | grep '^d' | more     List only the subdirectories of the current directory.
```

ls -l may output many lines, but **ls -ld** will output only one. The **-d** option shows you the directory itself, not the contents of the directory. The dot stands for the current directory; see Handout 1, p. 9.

```
5$ ls -ld                       Combine -l and -d to the single option -ld.
drwxr-xr-x   1 abc1234     users   3395 May 28 19:00 .
```

List individual files instead of all the files in a directory:

```
6$ ls -l moe                    ls will accept a filename.
7$ ls -l moe larry
```

You can list another directory without going there:

```
8$ ls -l /other/directory | more    ls will accept a directory name.
9$ ls -ld /other/directory          Why doesn't this need | more?
10$ ls -l /other/directory1 /other/directory2 | more
11$ ls -ld /other/directory1 /other/directory2
```

List individual files in other directories:

```
12$ ls -l /etc/passwd
```

Chronological order instead of alphabetical:

```
13$ ls -lt | more       "Time order": start with the newest. Combine -l and -t to -lt.
14$ ls -ltr | more      "Reverse time order": start with the oldest. Combine -l, -t, and -r.
```

File permission bits: p. 53

If you have permission to read, write, or execute a file, a lowercase **r**, **w**, or **x** will appear in the output of **ls -l**. If you have no permission, a dash will appear in place of the letter.

r permission lets you input the file into a program. Without **r** permission, you can't input the file into any program, including the programs for displaying or printing files. In this case we say that the file is *read protected* because there is no way to see the contents of the file.

w permission lets you direct a program's output into the file. Without **w** permission, you can't direct the output of any program into the file, including programs such as editors. In this case we say that the file is *write protected* because there is no way to change the contents of the file.

x permission lets you execute the file if it is a program. (Not all files are programs—some contain résumés, images, etc.) Without **x** permission you can't execute the file. In this case we say that the file is

not executable.

A simple way to create a file: p. 29

We'd like to turn the nine permission bits of a file on and off. But we can do this only if we own the file. (If we could do it to any file, the bits would serve no purpose—they wouldn't prevent anybody from doing anything.) By default, a file is owned by the person who creates it. In a moment, we'll see why the file we create has to go in our home directory.

```
1$ cd                       Go to your home directory (because you have w permission there).
2$ pwd                      Verify that you arrived there.

3$ date
Tue May 28 15:17:59 EDT 2013

4$ date > junk              Create file named junk in current directory. This time, you see no output from date.
5$ ls -l                    Verify you created junk. rw-r--r-- from umask 022 in /etc/profile, l. 63.
-rw-r--r--   1 abc1234  users         29 May 28 15:17 junk

6$ chmod 777 junk           "change mode": give it three octal digits
7$ ls -l
-rwxrwxrwx   1 abc1234  users         29 May 28 15:17 junk
```

Octal digits for the first argument of chmod: p. 56

An octal digit represents three bits, i.e., three yesses and noes.

```
0     ---
1     --x
2     -w-
3     -wx
4     r--
5     r-x
6     rw-
7     rwx
```

Therefore three octal digits represent nine bits. For example,

600 means **rw-------**
444 means **r--r--r--**
644 means **rw-r--r--**
555 means **r-xr-xr-x**
755 means **rwxr-xr-x**
777 means **rwxrwxrwx**

In Handout 7, we'll write our own version of **chmod** that will let you say

```
1$ chmod rw-r--r-- filename
```

instead of

```
2$ chmod 644 filename
```

Directory permission bits: pp. 55–6

x permission for a directory lets you **cd** to that directory. You must also have **x** permission for all of the directory's ancestors. To see why you can't **cd** to the directory **~mm64/45/grades**,


```
1$ cd ~mm64/45
2$ pwd

3$ ls -ld grades
drwx------   2 mm64     users          3 May 28 15:17 grades

4$ cd grades
ksh[1]: cd: grades: [Permission denied]
```

Having **r** permission allows you to **ls -l** the directory after you have **cd**'d there.

Having **w** permission allows you to put files into the directory, take files out of the directory, or rename the files that are in the directory. That's why we created our first file in our home directory: it had to be a directory where we had **w** permission.

There are two ways to put a file into a directory: you can create it there, or move it there from somewhere else. Similarly, there are two ways to take a file out of a directory: you can remove (i.e., destroy) it entirely, or you can move it somewhere else. But without **w** permission, you can't do any of these things: the existing files are locked into the directory. Even so, you may still have **w** permission for an individual file in the directory, allowing you to edit the file as long as you don't try to take it out of the directory or rename it.

▼ Homework 1.2: chmod a directory (not to be handed in)

Let other people **cd** to your home directory and **ls -l** it:

```
1$ cd
2$ pwd
/home1/a/abc1234

3$ ls -l | more                 This may output many lines.

4$ ls -ld                       This will output only one line.  Dot means your current directory.
drwx--x--x   2 abc1234  users        512 May 28 18:00 .

5$ chmod 755 .                  Don't turn on the last two w bits!
6$ ls -ld
drwxr-xr-x   2 abc1234  users        512 May 28 18:00 abc1234
```

▲

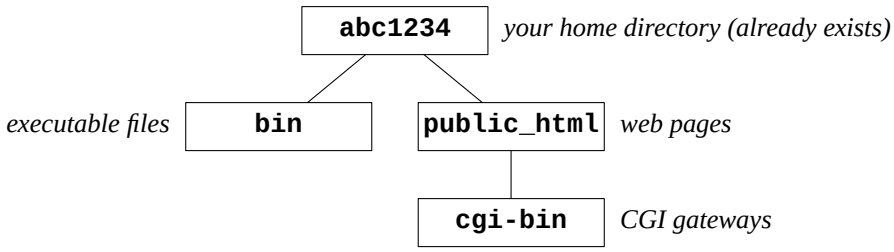
Create and remove a directory: pp. 25−26

```
1$ cd                   Go to your home directory.
2$ pwd                  Verify that you arrived there.
3$ ls -l | more         Verify that you don't already have a file or directory named mydir.

4$ mkdir mydir          "make directory": dead silence.
5$ ls -l | more
drwxr-xr-x   2 abc1234     users        117 May 28 15:18 mydir

6$ cd mydir             Go down into your new directory.
7$ pwd                  Verify that you arrived there.

8$ ls -l                It's empty, isn't it?
total 0
```



```
9$ cd ..                Go back up to your home directory.
10$ pwd                 Verify that you arrived there.

11$ rmdir mydir         "remove directory": dead silence
12$ ls -l | more        Verify that it's gone.
```

▼ Homework 1.3: create three directories (not to be handed in)



If you do not have a **bin** subdirectory of your home directory, create it. Be sure that the name is all lowercase. Later in the course, you will put your executable files (i.e., programs, e.g., shellscripts) there. **bin** stands for ''binary'', because all programs were binaries (i.e., compiled) in the early days of Unix.

If you do not have a **public_html** subdirectory of your home directory, create it. Be sure that the name has an underscore, not a dash, and ends with a lowercase L. Later in the course, you will put your World Wide Web pages (e.g., your home page) there. **html** stands for ''HyperText Markup Language''.

If you do not have a **cgi-bin** subdirectory of your **public_html** directory, create it. Be sure that the name has a dash, not an underscore. Your **cgi-bin** directory will be a grandchild of your home directory. Later in the course, you will put your World Wide Web CGI gateways there. CGI stands for ''Common Gateway Interface''.

chmod the three new directories (as well as your home directory, Homework 1.2) to **rwxr-xr-x** if they don't already have these modes.

▲

▼ Homework 1.4: draw part of the tree

cd to a few directories and draw me a map of what you find. Don't explore the whole tree. Just draw 10 or 15 directories and files to demonstrate that you can move around comfortably. Visit the root directory (its name is just a **/** by itself), the **cgi-bin** grandchild of your own home directory, and all the directories along the path between them. Visit the **bin** subdirectory of your home directory. Try to visit a classmate's home directory: you won't be able to unless they have given you permission in Homework 1.2. Try to get to

```
 1      ~mm64                          my home directory
 2      ~mm64/bin
 3      ~mm64/public_html
 4      ~mm64/public_html/INFO1-CE9545
 5      ~mm64/public_html/INFO1-CE9545/src
 6      ~mm64/public_html/cgi-bin
 7      ~mm64/45/grades                Should be inaccessible.
 8      /etc
 9      /bin
10      /usr/games                     Maybe you have this; ours is empty.
11      /var/apache                    our web server
```

Draw a box around the name of each directory to distinguish them from the files. Draw a double box around your home directory. (Its name is not **home**.) Use

(1) **pwd** to see where you are now, i.e., the name of your current directory (pp. 21−22);

(2) **ls -l** or **ls -l | more** to see the names of the files and subdirectories in the current directory (pp. 53−54); and

(3) **cd** to travel around the tree (pp. 25−26).

You get no credit if you forget the root directory at the top of the tree, ★ or if you write lowercase file and directory names in uppercase. Draw a tree: do not hand in a printout of directory listings. Draw one connected diagram, not several separate diagrams.

▲

▼ Homework 1.5: create, copy, and print a file

In your home directory, create one small file whose name is your last name (not your login name) in all lowercase. Copy it into the directory **~mm64/public_html/INFO1-CE9545/homework**. Turn on all three of the copy's **r** bits. Then print it.

This is the only homework that you should copy into this directory. Do not put more than one file into this directory. Do not put a subdirectory into this directory.

Create the file anyway you wish, and put anything you want into it. For example, take the standard output of any command that has one (i.e., any command we've seen except **cd**). If your name is Neuman, any one of the following **>** commands will do:

```
1$ cd                          First go home.
2$ pwd                         Verify that you arrived there.

3$ date > neuman               Create file neuman in current directory.
4$ pwd > neuman
5$ ls -l > neuman
6$ cal 5 2013 > neuman
7$ echo hi there > neuman
8$ fortune > neuman

9$ who > neuman
10$ who | sort > neuman        Not limited to one program; could have pipeline of programs.
```

Now that you've created the file and written text into it, append some more text to the end of the file with the **>>** operator (p. 29):

```
11$ ls -l neuman               See how big it is.
12$ whoami >> neuman           no space between the two >'s
13$ ls -l neuman               Observe that it got bigger.
```

What happens if you try to append to the end of a file after turning off its three **w** bits? Remember to turn them back on after the experiment.

cat or **more** the file to see what you've put in it. On pp. 1−2 we fed **more** from a pipe, but now we're feeding it from a file. Unix programs don't know and don't care where their input comes from, or where their output goes to.

```
14$ cat neuman                 if you know that the file is small enough to fit on the screen
15$ more neuman                if the file is big: space bar to advance, b to go back up, q to quit
```

What happens if you try to **cat** or **more** one of your files after turning off its leftmost **r** bit? Remember to turn it back on after the experiment.

Copy your file (p. 17) into the directory **/home1/m/mm64/public_html/INFO1-CE9545/homework**, go there, and see if it actually arrived:

```
16$ cp neuman ~mm64/public_html/INFO1-CE9545/homework/neuman
```



```
17$ cd ~mm64/public_html/INFO1-CE9545/homework
18$ pwd                                  Verify that you arrived there.
19$ ls -l | more                         Verify that the cp worked.

20$ chmod your file to rw-r--r-- or r--r--r-- so everyone can read it.
21$ ls -l neuman                         Verify that the chmod worked.

22$ cd                                   Go back home.
23$ pwd                                  Verify that you arrived there.
```

But before copying the file, see if that directory already contains a file with your name that belongs to someone else. In that case, call your copy **neuman2**:

```
24$ cp neuman ~mm64/public_html/INFO1-CE9545/homework/neuman2
```

You can also see the contents of this directory by pointing your browser at

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

You can use any printer for this course. To print the file on campus on a laser printer, write **-P** in front of the name of the printer. Other systems use **lp -d** instead of **lpr -P**.

```
25$ lpstat -a                            see the names of "all" the printers
26$ lpr -Pedlab neuman                   printer at 14 Washington Place
27$ lpr -Pndlab neuman                   printer at the North Dorm
```

If the file is taking suspiciously long to print, see what's happening by typing

```
28$ lpq -Pedlab                          See the queue for the printer you used above.
```

You may have to say **/usr/ucb/lpq** instead of plain **lpq**; it will probably give you an error message. Do not give the **lpr** command again until your first file has finished printing.
▲

Post a file on the World Wide Web and print it from there

Make sure that your home directory and its **public_html** subdirectory are both **rwxr-xr-x** (Homeworks 1.2 and 1.3). Copy your file into the **public_html** subdirectory of your home directory if it's not already there. (The tilde ~ stands for the full pathname of your home directory.) Then give everyone permission to read the file, to make it visible in your browser.

```
1$ cp neuman ~/public_html/neuman.txt    Create a copy named newman.txt.

2$ cd ~/public_html
3$ pwd
4$ ls -l neuman.txt

5$ chmod 644 neuman.txt                  Change it to rw-r--r--.
6$ ls -l neuman.txt                      Verify that the chmod worked.
```

Then you can read, print, or download the file by pointing your web browser at

http://i5.nyu.edu/~abc1234/neuman.txt

In a Unix command line, a loginname with a leading tilde **~abc1234** stands for the full pathname of that person's home directory. But in a web URL, a loginname with a leading tilde **~abc1234** stands for the full pathname of the **public_html** subdirectory of that person's home directory.

```
7$ chmod 600 neuman.txt                  Change to rw------- so no one else can read it.
```


See a Handout in your web browser

Point your web browser at

```
http://i5.nyu.edu/~mm64/handout/
```

See a manual page in your web browser

To see our Unix manual in PDF format, point your web browser at

```
http://i5.nyu.edu/~mm64/man/
```

The man pages for the Apple iPhone iOS are at

```
http://developer.apple.com/library/ios/#documentation/System/Conceptual
/ManPages_iPhoneOS/index.html
```

In an emergency, you can certainly print a manual page without using the web:

```
1$ man ksh | lpr -Pedlab
```

but it will not have italics, boldface, bullets, etc.

Syllabus for INFO1-CE9545

Unix programs (utilities, tools)

alpine	false	mv	tee
at	find	netstat	test
awk	finger	nice	tr
bc	fold	nm	troff
cal	fortune	nngrep	true
cat	grep	nroff	tty
cc	head	nslookup	uncompress
chmod	kill	passwd	uniq
clear	ksh	pr	vi
cmp	last	ps	view
comm	locale	pwd	vim
compress	ln	rm	w
cp	lpq	rmdir	wall
cut	lpr	script	wc
cvs	lprm	sed	whatis
date	lpstat	sleep	whereis
deroff	ls	sort	which
df	lynx	spell	who
echo	mailx	ssh	whoami
egrep	man	tail	write
env	mkdir	talk	zcat
expr	more	tar	

Files

/etc/motd	~/.profile	/dev/null	/usr/pub/ascii
/etc/passwd	~/.mailrc	/dev/tty	/usr/dict/words
/etc/group	~/.plan	/usr/openwin/lib/rgb.txt	
/etc/shells	~/.project	~/public_html/index.html	
/var/apache2/2.2/logs/access_log		/etc/apache2/2.2/httpd.conf	

Korn shell language punctuation marks

> 2> 1>&2	' '	$*varname*	${*varname*%%}
<	" "	${*varname*}	| & ;
>>	` `	${*varname*#}	&& ||
<<	( )	${*varname*##}	* ? [!-]
#	#!	${*varname*%}	

Korn shell language keywords

if then fi	continue	export	control-c
else elif	break	set	control-d
for in do done	exit trap	fg bg %1 jobs	control-z
while do done	alias r	cd	.

Korn shell language variables

$HOME	$SHELL	$1 $2 $3	$* $@
$PATH	$TERM	$0	$$
$LOGNAME	$PS1	$#	$?

vi editing commands

a i	p P	^ $	control-D
cw C	r	/ ? n %	control-U
dd dw D	s	:g//+1p	control-E
G	u	:q	control-Y
hjkl w b	x	:quit!	ESC
J	yy	:s	control-G
o O	!!	:w	control-L

Regular expressions in grep, egrep, more, vi, sed, and awk

^$	.	\(\)	(|)
*+?	[abc]	\1 \2 \3	\
\{,\}	[a-z]	\< \>	
	[^a-z]		

awk

NR	$1 $2 $3	substr()	{print}
NF	$NF $(NF-1)	BEGIN	
length	$0	END	

HTML tags for World Wide Web pages

<HTML>	<A>	<OL>	<TABLE>
<HEAD>	<EM>	<UL>	<TH>
<BODY>	<STRONG>	<LI>	<TR>
<TITLE>	<IMG>	<FORM>	<TD>
<H1>	<MAP>	<INPUT>	
<P>	<AREA>	<PRE>	<HR>
<!-- -->			

Reference book

The UNIX Programming Environment by Brian W. Kernighan and Rob Pike.

`http://netlib.bell-labs.com/cm/cs/upe/index.html`
`http://www.cs.bell-labs.com/~bwk`
`http://herpolhode.com/rob/`

Contact information

Home page for this course:	**`http://i5.nyu.edu/~mm64/INFO1-CE9545/`**
Mark Meretzky's email address:	**`mark.meretzky@nyu.edu`**
Mark Meretzky's home page:	**`http://i5.nyu.edu/~mm64/`**

The system administrator's address is **`comment@i5.nyu.edu`**. For the NYU computer help desk, send email to **`its.clientservices@nyu.edu`**, or call (212) 998-3333, or visit the ITS Client Services Center at **`http://www.nyu.edu/its/askits/`**. For problems with computer accounts and passwords, send email to the accounts office at **`its.clientservices@nyu.edu`** or call (212) 998-3333, or visit **`http://www.nyu.edu/its/accounts/`**. For information about grades, incompletes, and NYU courses, including courses which I will teach next semester, call the School of Continuing and Professional Studies at (212) 998-7190. To contact me after the course is over, please send me email—don't phone.

Computer labs at NYU:
`http://www.nyu.edu/its/labs/`

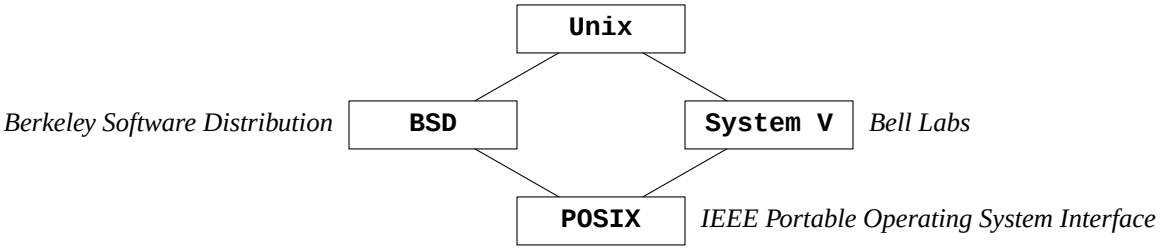
(212) 998-3409	Room LC-8 Tisch Hall, two flights down
PC's	40 West 4th Street at Greene Street
printer:	none accessible via the i5.nyu.edu **`lpr`** command
(212) 998-3457	14 Washington Place, one flight down
PC's	between Greene and Mercer Streets
printer:	none accessible via the i5.nyu.edu **`lpr`** command
(212) 998-3421	Education Building, second floor
Macs	35 West 4th Street at Greene Street
printer:	**`edlab`**
(212) 998-3504	North Dorm, two flights down
PC's and Macs	75 Third Avenue (southeast corner of Third Avenue & 12th Street)
printer:	**`ndlab`**

To get your plastic, magnetic NYU ID card, see

`http://www.nyu.edu/nyucard/`

The names have been changed because "Unix" is copyrighted

company	*name*
Apple	**A/UX**
AT&T	**Unix**
DEC	**Ultrix**
Hewlett-Packard	**HP-UX**
IBM	**AIX**
IEEE	**POSIX**
Linus Torvalds	**Linux**
Microsoft	**Xenix**
Silicon Graphics	**Irix**
Oracle	**Solaris**



Your i5.nyu.edu account

Our computer is Solaris 250 server running Solaris 10 (SunOS 5.10). It is actually a Solaris *zone*, a virtual machine sitting on top of a ''global'' zone. Its Internet hostname is **i5.nyu.edu** and its IP version 4 address is **128.122.109.53**.

Your i5.nyu.edu login name is listed at

http://i5.nyu.edu/~mm64/common/students.html

It is the same as your NYU NetID used by your NYU DIAL or NYU Home account. It will be two or three *lowercase* letters (your initials) followed by one or more digits. In these Handouts, we'll assume your login name is **abc1234**.

Your i5.nyu.edu secret password is the same as your central NYU single sign-on password used by your NYU DIAL or NYU Home account. In these Handouts, we'll assume your password is **Bacall8?**.

Before using your login name and password for the first time, register at **http://start.nyu.edu/**. First time i5.nyu.edu users must leave the password field blank as they have not yet set their password. They will then be prompted to enter their social security number and birth date.

To change your password to a more colorful one, e.g., **bogart!** or **Bacall8?**, go back to **http://start.nyu.edu/**.

Read all of this Handout before doing anything, especially the ''What can go wrong'' section.

The "secure shell" ssh

To log into **i5.nyu.edu**, you have to run a program that speaks the ''secure shell protocol''. One example is a program named **PuTTY** or **putty.exe**. If you don't already have it, get it from

http://www.chiark.greenend.org.uk/~sgtatham/putty/
https://www.nyu.edu/its/software/

(1) On Windows, run **putty.exe**. A window named **PuTTY Configuration** will appear. Type **i5.nyu.edu** as the host name, and select the radio button for the protocol **SSH**. The port number should be 22. Then press **Open**. Dismiss any **PuTTy Security Alert** window that may appear.

```
login as: abc1234
Password: Bacall8?
```

To copy from the **PuTTy** window, select (highlight) the text you want to copy. You do not have to say ''copy''. To paste into the window, click on the place where you want to paste and press the right mouse button.

(2) From any other Unix host, you can get to **i5.nyu.edu** by running the program **ssh**. If you don't already have it, get it from

```
http://www.openssh.com/
```

On Mac OSX, for example, lauch the **Terminal** application to get a Unix shell window. (If you can't find **Terminal**, go to the Finder, pull down **File**, select **Find...**, and search for **Terminal.app**.) Pull down the **Font** menu and select a pleasant font. Then give the command

```
ssh abc1234@i5.nyu.edu
```

(3) On an Apple iPhone, get the **TouchTerm** app. On an iPhone or iPad, get the **pTerm** app.

(4) On Android, get the **ConnectBot** app.

After logging in

When you are finished logging in, you will see the prompt. To verify that you are really logged in, run simple programs such as

```
1$ date                          Press RETURN after each command line.
2$ cal 5 2013                    Need space before each command line argument.
```

Log out

See note (1) below if you can't log out because of ''stopped jobs''.

```
1$ exit                          or logout on other systems
```

If the terminal window is still open, pull down the **File** menu and select **Exit** or **Quit** to close it.

The special keys: symptoms and antidotes

(1) If you accidentally type **control-z**, it will say **Stopped**. To start things up again, type

```
1$ fg                            Bring the most recently stopped program back into the foreground.
```

If it says **You have stopped jobs** when you try to log out, type **fg** to give your stopped job a chance to finish. Repeat if necessary.

(2) Press **Backspace** on PC, **delete** on Mac to erase the last character typed. As a last resort, see if **control-h** will backspace.

(3) To kill a long program, type **control-c**. You may have to type it more than once.

(4) **Control-s** may freeze the screen; unfreeze it with **control-q**. Similarly, **Scroll Lock** on PC may freeze the screen; unfreeze it with another **Scroll Lock**.

(5) Never press **Caps Lock** in Unix: almost everything we type will be lowercase. Don't confuse
- (5a) the lowercase letter **l**, the uppercase letter **I**, and the digit **1**
- (5b) the lowercase letter **o**, the uppercase letter **O**, and the digit **0**
- (5c) the diagonal slash **/** and the backslash ****
- (5d) the single quote **'**, the double quote **"**, and the back quote **`**
- (5e) the dash **-**, the underscore _, and the tilde **~**

(5f) the left parentheses **(**, the left curly brace **{**, and the left square bracket **[**
(5g) the right parentheses **)**, the right curly brace **}**, and the right square bracket **]**
(5h) the vertical bar (pipe symbol) **|**, the colon **:**, and the exclamation point **!**
(5i) any uppercase letter and the corresponding lowercase letter.

□