Terminal 1: The Shell and Scripting

Stuff Worth Knowing, Chapter 1

Paul E. Johnson[1,2]

[1]Department of Political Science

[2]Center for Research Methods and Data Analysis
University of Kansas

2016

Outline

Practical Reasons to explore the Command Line

- Some tools only available for "command line interface"
  - rsync
- Some chores too tedious for "point and click"
  - find all files with letters "doc" and change to "odt"
  - download 33000 Ukrainian election data files and squeeze out data
  - resize 1000 images to change their resolution from 1600x1200 to 800x600
  - count the number of pdf files produced by a program that creates 1000s of directories and subdirectories
  - find the longest length of filename in a giant file hierarchy
- Only way to see "error" output in many programs.
- GUI always lags behind what's possible in the "command line"

More Practical Reasons

- Networking may not allow a GUI remote connection (May need to get by without a GUI or mouse)
- GUI may "crash" or "stall", but CLI access may still work.
- Linux is the standard Web server platform.
- Linux is the High Performance Computing platform.
- Linux is development environment preferred in neuro-science (`http://neuro.debian.net`)

There is Always a Terminal (Under There, Somewhere)

- Microsoft "DOS Box" is a terminal program
  - Menu: Under Start Menu/Accessories
  - Run prompt: "command"
- "Command" is the default "shell" program on MS Windows, but there are others.
- Mac also supplies a Terminal program
- Linux/Unix systems (of course) also offer many terminal programs.
  - Every desktop framework chooses its own favorite
  - I like "MLterm" because of multi language support and Graphics options
  - Emacs (text editor) is delivered with its own terminal programs "M-x shell" and "M-x eshell"

What is a (Virtual) Terminal?

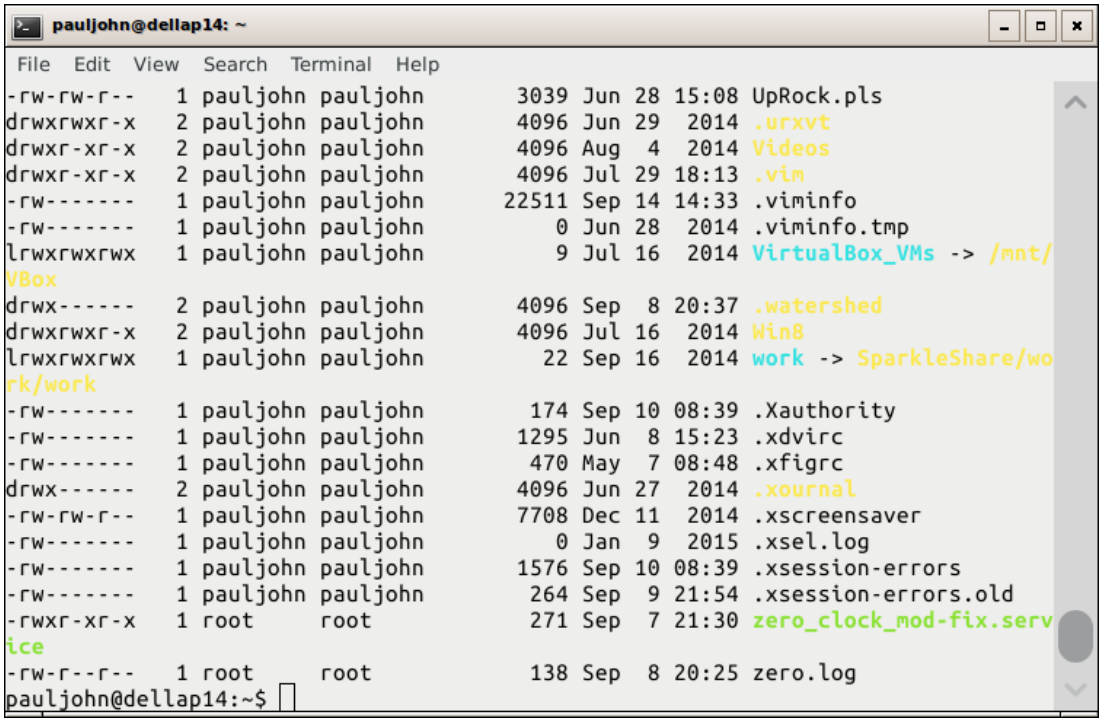
AKA: Terminal Emulator or Console

What is a (Virtual) Terminal? ...

- Olden days: A terminal is box with keyboard.

What is a (Virtual) Terminal? ...

- Now: A terminal emulator is a box on the screen, showing this or another computer (It as if she were sitting in front of the remote system.)



- Only rely on secure shell connections
  - "Secure Shell" means that the user's password is never "exposed" as non-encrypted text as it travels.

What is a (Virtual) Terminal? ...

  - **ssh login2.acf.ku.edu** # goes to our ACF headnode
  - **rsync -e ssh -rav some-dir login2.acf.ku.edu:** # copies updates in folder some-dir to my accoung on ACF
  - **scp -r some-dir login2.acf.ku.edu:** # almost same as rsync, but this copies even files that are the same.
- "telnet" and "ftp" are old fashioned protocols, considered insecure because the password is not encrypted. Avoid if possible.

Difference between "Terminal" and "Shell"

terminal A program that prompts, accepts input, renders output. It immitates a "physical terminal" connected to a computer.

shell A program that interprets user commands, supplies information to programs.

environment A collection of settings that "exist" for access by programs and can be set by users

Run: "env" or "set" to see the environment.

Examine Your Environment

In Windows

- Open a Command Shell
- Type “set”
- Run a program: Type “notepad” or the name of any other “exe” file you see in the Windows folder. Or “iexplore”.
- Control Panel -> System -> Advanced -> Environment

In Mac or Linux

- Open a Terminal (many ways to do it).
- Type “env” and “env | grep PATH” or “env | grep HOME”
- type one letter and hit TAB a few times. A list of programs appears.
- Run some programs. Try “firefox” or “safari”

Shell Features: Tab Completion and Command History

- All Terminal programs (as of 2009) had "tab completion" of program and file names.
- All Terminal programs had "command history". Usually up-arrow cycles through past commands

Things to remember about Command Line Interaction: &

- By default, most programs "occupy" the shells until they are closed
- Hence, user cannot run new commands until previous is finished.
- Workaround: Put function into shell's background by appending &
  - Example: I can't run any new commands until I close emacs

```
$ emacs myFabulousProgram.R
```

  - This free's up the command line

```
$ emacs myFabulousProgram.R &
```

  - At one time, it seemed as though all GUI programs would "control" the terminal, however, some programs now will automatically background themselves.

Things to remember about Command Line Interaction: & ...

- Example: *gvim* will free the terminal once it is launched

```
$ gvim myFabulousProgram.R
$
```

Things to remember about Command Line Arguments

- ls lists files, but
- "ls -la"
  - -a show all files & directories, including hidden files (begin with a period)
  - -l detailed listing includes ownership, file size information
- "ls -la –color=no" or "ls -la –color=yes" or ...

Things to remember about Command Line Interaction: -or --

Unfortunate: There are quite a few ways to give command line arguments

- R, itself
  - R CMD ____<one of: BATCH, INSTALL, build, check, Sweave, Stangle>____

```
$ R CMD BATCH myFabulousProgram.R
#or
$ R CMD INSTALL rockchalk_1.9.tar.gz
```

  - single dash with a script file name

```
$ R -f myFabulousProgram.R
```

  - double dash with no argument

```
$ R --vanilla -f myFabulousProgram.R
```

Things to remember about Command Line Interaction: - or -- ...

- qxlogin
  - Ever notice that the ACF qxlogin accepts arguments like this:

```
$ qxlogin 1 1
```

or this

```
$ qxlogin 1 2,program=mplus
```

- GNU style {is, was, has been} an effort to standardize this
  - Relatively widely practiced style.
  - Two dashes and an equal sign and an argument

```
$ myprogram --avar=1 --bvar=2
```

or

  - One dash with no equal sign, as a flag:

```
$ myprogram -a -b
```

Things to remember about Command Line Interaction: - or -- ...

- One dash with a value smashed up against the argument (no equal sign or space between)

```
$ myprogram -a1 -b2
```

- Many GNU programs have both the
  - verbose –argument=1 style
  - less verbose -a1 style
- Some programs do whatever they want
  - ps “ps -aux” or “ps aux”
  - tar
  - java, -option=value

Important Concepts in All Shells in Any OS

- PATH: list of directories where OS searches for programs
- Linux path is colon separated list, eg: mine is:

```
PATH=/home/pauljohn/
    bin:/usr/local/
    sbin:/usr/local/
    bin:/usr/sbin:/
    usr/bin:/sbin:/
    bin:/usr/games
```

- Referred to as $PATH in other commands
- In Windows, the semicolon is a separator, slashes backwards

```
PATH=C:\Windows;C:\
    Windows\system32;C:\
    Program Files\Mozilla
     Firefox;
```

- Referred to in other commands as %PATH%

Home, Working Directory

- HOME. User's personal "folder", default place where files go.
- Working Directory. Most programs will read & write from current working directory
- **pwd** # lists the working directory
- **cd** # sets the working directory
- Windows Icon GUI calls this "Start In" option

Is the current working directory in the path? Maybe

- Suppose you install a program that is not in the path. What happens?
- Can run by typing full address of program
  - "C:\Program Files\GNU Emacs 23.2\bin\runemacs.exe"
  - /usr/bin/emacs

  Suppose you cd to that folder
  - Windows allows to run by "name" b/c current dir is in path
  - Linux does not have current dir in path, hence, run as: ./name
- The inherent problem with spaces and special characters in directory or file names. We can workaround, but don't create work for yourself

Things to Keep on the Tip of Your Tongue

1. ls list directory contents
2. mkdir create a directory
3. cd change the current working directory
4. mv move (for renaming files or relocating directories)
5. cp copy
6. rm remove

1. ls :List Files

- ls
- ls -la
- ls -la | more
- ls -s1
- ls --color=auto
- ls --color=no

View Permissions with "ls -la"

```
$ ls -la
total 89724
drwxr-xr-x 182 pauljohn pauljohn    20480 2011-01-24 01:28 .
drwxr-xr-x   7 root     root         4096 2010-11-16 20:41 ..
-rw-r--r--   1 root     root         3460 2010-11-07 22:55 50emacs-ess-ku.el
-rw-r--r--   1 pauljohn pauljohn    19661 2010-07-26 11:37 ABM.bib
drwx------   5 pauljohn pauljohn     4096 2010-10-03 21:48 .adobe
drwxr-xr-x   3 pauljohn pauljohn     4096 2009-08-03 21:37 Adobe
-rw-r--r--   1 pauljohn pauljohn       15 2010-12-13 01:57 #adsf.R#
-rw-r--r--   1 pauljohn pauljohn      120 2009-07-19 13:27 .album.conf
drwxr-xr-x  25 pauljohn pauljohn     4096 2009-02-28 23:38 .amaya
-rw-r--r--   1 pauljohn pauljohn      528 2011-01-23 15:36 .anyconnect
-rw-r--r--   1 pauljohn pauljohn      406 2010-12-31 23:33 #armani.txt#
```

perms owner group filesize date filename

- "." at top is current working directory name
- ".." in 2nd line is directory above corrent working directory

Focus on "drwxr-xr-w"

There are 3 types of Users declared for each file or directory

- owner
- group
- others: every account excluding owner & group

d is it a directory (if -, a file)

rwxr-xr-x permissions of 3 user types

- r: read, w: write, x: execute
- owner has rwx
- group members have only rx
- others (the "world") have only rx
- permissions can be revised by the program "chmod"
- ownership can be revised by "chown" or "chgroup"

Regular and Hidden and Backup Files

. dot files, by custom are hidden (not displayed by "ls" unless you specifically ask for them). Used for configurations

\# backup files, created by editors.

2. mkdir : Make directory

directory = "folder" = collection of files and directories

- **mkdir *some-dir-name*** # creates directory *some-dir-name*
- **mkdir -p *some-dir-name/ sub-dir-name/ sub-sub-dir***

3. cd : change directories

- **cd *some-dir-name*** #changes current working directory to *some-dir-name*
- **cd** # changes to user's HOME
- **cd /** # changes to "top level"
- **cd *some-dir-name/subdir-name/subsub-name*** # ok to nest
- **cd ../** # changes to higher directory
- **cd ../../*some-dir-name*** # 2 dir up, down into *some-dir-name*

4. mv: move file or directory

- **mv *some-file-name some-other-name*** # "renames" file
- **mv *some-dir some-other-dir***
  - If *some-other-dir* exists, this relocates *some-dir* inside *some-other-dir*
  - If *some-other-dir* does not exist, this effectively "renames" *some-dir* as *some-other-dir*.

Moving Example

```
$ mkdir test
$ cd test
$ mkdir a

$ mkdir b
$ ls -la
total 16
drwxr-xr-x 4 pauljohn pauljohn 4096 2011-01-24 01:11 .
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:10 ..
drwxr-xr-x 2 pauljohn pauljohn 4096 2011-01-24 01:10 a
drwxr-xr-x 2 pauljohn pauljohn 4096 2011-01-24 01:11 b

$ mv a b
$ ls -la
total 12
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:11 .
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:10 ..
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:11 b

$ ls b
a
$ ls -la b
total 12
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:11 .
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:11 ..
drwxr-xr-x 2 pauljohn pauljohn 4096 2011-01-24 01:10 a

$ mv b c
```

Moving Example ...

```
$ ls -la
total 12
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:11 .
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:10 ..
drwxr-xr-x 3 pauljohn pauljohn 4096 2011-01-24 01:11 c
```

5. cp: copy file or directory

- **cp *some-file-name some-other-name***
- **cp *some-dir some-other-dir*** # does not work
- **cp -R *some-dir some-other-dir*** # -R means "recursive"
- **cp -a *some-dir some-other-dir*** # -a recursive and also preserves file attributes (modification time, etc)
- like mv in semantics:
  - If *some-other-dir* exists, this creates a copy of *some-dir* inside *some-other-dir*
  - If *some-other-dir* does not exist, this creates a copy of *some-dir* called *some-other-dir*.

6. rm: remove

- **rm *some-file-name some-other-name***
- **rm** may be very dangerous, can remove things immediately, without confirmation
- Run like this to ask for interactive yes/no approval: **rm -i**
- I forget that, so on my systems, I insert a fail-safe that asks for confirmation of deletions. I suggest all Linux systems should do this, an surprised many do not.
  Find out if your system is "safe": Run "type rm"
  On my System, I get
  "rm is aliased to 'rm -i"
- **rm -f *some-file-name some-other-name*** # -f=force
- **rm -rf *some-dir*** # removes directory, -r means "recursive"
- If you forget the -f

```
$ mkdir a
pauljohn@pols124:tmp$ rm a
rm: cannot remove 'a': Is a directory
```

Important, Useful, Handy (but not quite Vital)

cat	dump file output to the screen
grep	scan text for terms
|	the pipe
>	redirect to new file
»	redirect and augment file
find	find files by various characteristics
tar	contraction of “tape archive”
df	report disk usage (“disk free”)
free	report on free memor (“RAM”)
top	display running programs and memory usage
kill	kill a program by PID
ps	ps displays running processes (ps aux)

cat

- Suppose a file "whatever.txt" exists
- List file contents on the screen

```
$ cat whatever.txt
```

- more and less are 2 competing "text pagers". more was commercial, so less was offered as a free competitor

```
$ cat whatever.txt | less
$ cat whatever.txt | more
```

- Either will break up cat output into screen-sized pages

grep: Text scanning

- scans all non-hidden files for text string “flopper”

```
$ grep flopper *
```

- Sends cat output to grep for line-by-line scanning to check for “flopper”

```
$ cat some-file | grep flopper
```

- | is pronounced “pipe”

The Difference between > and >>

- Append from output from cat|grep into a text file

```
$ cat some-file | grep flopper >> newfile.txt
```

- Erases original "newfile.txt", writes output into newfile.txt

```
$ cat some-file | grep flopper > newfile.txt
```

The Mighty Pipe: |

prog | other program The "pipe", diverts "stdout" from **prog** to **program** after pipe

- stdout: "standard output"
- stderr: "standard error"

Example: Handle a tar.gz "tarball".

- What is a tarball?
  - tar groups files together into an archive
  - gzip is a compression program.
  - A tar.gz file is the result of "tarring" and "gzipping"
- Could do this in 2 steps

```
$ gzip -d file.tar.gz  ## decompresses to
    create "file.tar"
$ tar xvf file.tar  ## de-archives file.tar
```

The Mighty Pipe: | ...

- Do this in 1 step with pipe

```
$ gzip -dc file.tar.gz | tar xvf -
```

  - c the gzip option -c means send results to standard output
  - - the minus sign on tar means "standard input"

- tar authors noticed complications and created command line options to handle decompression without pipe (see below).
- I use grep that way all the time to scan stdout

```
$ whatever | grep magicWord
```

Check what's running: **ps**

ps lists processes that are running under user's name in the current shell

```
$ ps
  PID TTY          TIME CMD
18023 pts/17   00:00:00 bash
18034 pts/17   00:00:00 ps
```

ps Run something, so you can see the effect

```
$ emacs &
$ ps
  PID TTY          TIME CMD
16912 pts/13   00:00:00 bash
18095 pts/13   00:00:00 emacs
18104 pts/13   00:00:00 ps
```

ps aux list processes by all users

Check what's running: **ps** ...

```
$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0 182852  5640 ?        Ss   Sep10   0:11 /sbin/init
    splash
root       1339  0.0  0.0 167144  6540 ?        Sl   Sep10   0:00 lightdm --
    session-child 12 19
pauljohn   1373  0.0  0.6 1660908 53344 ?       S<l  Sep13   0:14 /usr/NX/bin/
    nxnode.bin
root       1401  0.0  0.1 335628  9588 ?        Ssl  Sep10   0:14 /usr/lib/upower
    /upowerd
pauljohn   1414  0.0  0.4 1542012 35744 ?       Sl   Sep13   0:04 /usr/NX/bin/
    nxclient.bin --monitor --pid 24
rtkit      1432  0.0  0.0 168956  2592 ?        SNsl Sep10   0:02 /usr/lib/rtkit/
    rtkit-daemon
root       1444  0.0  0.1 2103936 8484 ?        Ssl  Sep10   0:02 /usr/sbin/
    console-kit-daemon --no-daemon
```

pipe and filter Scan for programs running that have letters "fire"

```
$ ps aux | grep fire
```

kill To eliminate an undesired program, run the kill function.

```
$ kill -9 1373
```

Check what's running: **ps** ...

kill sounds violent, but it is a standard shutdown signal to programs.
-9 is violent/agressive, however. It means "get out and don't try to save your work"

top: a more interactive sort of ps

mlterm

```
top - 17:27:27 up 4 days,  8:48,  2 users,  load average: 0.17, 0.21, 0.19
Tasks: 303 total,   1 running, 302 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.8 us,  1.2 sy,  0.0 ni, 96.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   8057560 total,  7963184 used,    94376 free,   878492 buffers
KiB Swap:  8265724 total,        0 used,  8265724 free.  4075064 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
15492 pauljohn  20   0 1333328 582312  91012 S   7.3  7.2   5:30.45 firefox
 1270 root      20   0 1070188 336976 306712 S   5.3  4.2  40:57.84 Xorg
13132 pauljohn  20   0 1121324 455036  89788 S   3.6  5.6  10:28.17 thunderbird
 2557 pauljohn  20   0  517656  34564   5472 S   1.3  0.4  13:25.22 ibus-daemon
 3563 pauljohn  20   0  437612  48552  35376 S   1.0  0.6  34:44.00 git-annex
15589 pauljohn  20   0  500860  81720  54924 S   1.0  1.0   1:49.95 lyx
 2603 pauljohn  20   0  455252  45016  24500 S   0.7  0.6   2:31.61 ibus-ui-gtk3
 2661 pauljohn  20   0  481760  32672  24740 S   0.7  0.4   0:50.85 xfce4-panel
18880 pauljohn  20   0  589364  27404  23016 S   0.7  0.3   0:00.17 screenshot
    7 root      20   0       0      0      0 S   0.3  0.0   3:57.13 rcu_sched
 2550 pauljohn  20   0  486584  28176  19724 S   0.3  0.3   0:30.10 bamfdaemon
 2636 pauljohn  20   0  190500   6144   5552 S   0.3  0.1   3:19.61 ibus-engine-sim
 2653 pauljohn  20   0  325152  34092  18968 S   0.3  0.4   0:35.54 openbox
10459 root      20   0       0      0      0 S   0.3  0.0   0:09.84 kworker/2:0
15441 pauljohn  20   0 1096584 185636  67316 S   0.3  2.3   0:46.71 evince
18875 pauljohn  20   0   25136   3324   2652 R   0.3  0.0   0:00.32 top
    1 root      20   0  182852   5640   3728 S   0.0  0.1   0:11.15 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.06 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:02.99 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0  0.0   1:10.76 rcuos/0
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcuob/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.32 migration/0
   12 root      rt   0       0      0      0 S   0.0  0.0   0:01.11 watchdog/0
```

Keyboard interaction with top

- kill. Letter k causes a prompt to ask which process should be killed.
- Then it asks how severely do you mean that. -9 is an aggressive choice for stalled programs.
- q to quit

find

- **find . -iname “*some*”**
  Beginning in “.”, the current directory Scan file names, ignoring capitalization, for all files that have the letters “some” anywhere in them.
- **find . -name “*some*”**
  Capitalization counts.
- **find /usr/local -name “*some*”**
  Search in /usr/local instead
- **find . -name “*some*” -exec emacs {} \;**
  Opens the selected files in Emacs.
- find has many search options, to look for files by size, modification time, or other details.
- Many systems have a less formal/powerful alternative “locate”

find has many super powers that can save you

Examples based on the DLM project Summer 2015

- Program creates 100s of directories, 1000s of subdirectories inside them, then writes pdf files (and other files in there).
- Question: How many pdfs are there altogether in a directory structure

```
$ find myoutdir -name "*.pdf" | wc
```

- Question: Drop a list of those pdfs into a text file

```
$ find myoutdir -name "*.pdf" > reports.txt
```

- Question: List pdfs have total names (directory path beginning at current location) longer than 200 characters:

```
$ find -regextype posix-extended -regex '.{200,}'
```

find has many super powers that can save you ...

- Question: List files with the name "Jerry" in them:

```
$ find . -name "*Jerry*.pdf"
```

- Question: List files created within the last 60 minutes

```
$ find . -cmin -60
```

cmin: creation time
mmin: modification time

- Question: find all the files named .log and delete them

```
$ find . -name "*.log" --delete
```

tar: "tape archive" program

- **tar czvf progs-2011.tar.gz some-dir-name**
  Creates a GNU zipped archive of a folder "some-dir-name"
- **tar tzvf progs-2011.tar.gz**
  Scans and lists the contents of "progs-2011.tar.gz"
- **tar xzvf progs-2011.tar.gz**
  Decompresses and un-tars the files. Creates "some-dir-name"
- tar can deal with other types of compression
  - bzip: **tar xjvf progs-2011.tar.bz2**
  - see man tar for other types

Zip files less common, but still encountered

- "zip": Archives created by proprietary algorithm PK-zip
- **unzip -t whatever-2011.zip** # tests the archive
- **unzip whatever-2011.zip** # extracts the archive

Emacs has shells built in

Start Emacs, run
M-x shell
or
M-x eshell
and it will be obvious how you can keep records on your sessions.

Difference between > and 2>&1

- **prog > file.txt** # only diverts stdout into file.txt
- **prog > file.txt 2>&1** # diverts stdout and stderr into file
- Example usage: run "make" on a huge program, tons of output appears on screen
- run "make > build.out 2>&1 " and all output goes into file.

Do I Love Perl More Than Bash? Does Bash Mind?

- Bash shell is the Linux default shell
- see "man bash"
- For simple chores, Bash scripts are sufficient
- Interesting exercise: Convert a DOS script into a Linux shell script.
- For elaborate scripting, I have much more experience with Perl
- Perl-CGI was (in 2000) the predominant approach for writing interactive Web pages
- Many other scripting languages have their advocates, I don't intend to disparage (Python)

Example Bash scripting exercise

- Vacation photos too huge to email to family
- Need to shrink them

```
#!/bin/bash

for i in *.jpg; do base=`basename $i .jpg`;
  convert $i -resize 800x600 -quality 85 $base-800x600.jpg;
  done
```

- Method 1: Executable script
  - Save that in a file "resize.sh"
  - Use chmod to make it executable
  - Run with ./

```
$ chmod +x resize.sh
$ ./resize.sh
```

- Method 2: Run a shell, which executes this program.

```
$ sh resize.sh
```

Example Bash scripting exercise ...

- Result: All jpg in current directory will have smaller versions written.
- 2 details worth mentioning
  - The script "resize.sh" is not executable, and it is not in the path.
  - Method 1 uses chmod to make it executable, and then runs it with the "./" prefix. That means "In the current directory, find this program."
  - The OS reads the "shebang" line, #!/bin/bash, and is uses the bash program to run the script.
  - Method 2. Note that "bash" is a shell program, and "sh" is also a shell program. I'm in the habit of using "sh" to run things, but "bash" would be more correct. Both work in this case because the script does not use any special features that are unique to bash (so sh can do the job).
- If we were doing this over and over, we should
  - Move the file into the path, say in $HOME/bin

Example Bash scripting exercise ...

- Make sure the file is executable (chmod)

rename-perl.pl: A Perl Gem

- This was written by the authors of Perl, and was distributed as “rename” on most Linux systems in the olden days.

```
#!/usr/bin/perl

# Example usage: rename script examples from lwall:
#rename-perl 's/\.orig$//' *.orig

$op = shift;
for (@ARGV) {
  $was = $_;
  eval $op;
  die $@ if $@;
  rename($was,$_) unless $was eq $_;
}
```

- Line 1 is the "shebang" line
- When this script is executed, like this

```
$ rename-perl.pl some-options-here
```

The OS reads line 1 and executes this for us:

rename-perl.pl: A Perl Gem ...

```
perl rename-perl.pl some-options-here
```

- That only works because the script is executable. Otherwise, we'd have to explicitly call perl, like so:

```
$ perl rename-perl.pl some-options-here
```

- The latter approach does NOT require that rename-perl.pl is an executable file.

rename-perl.pl Travels With Me

- I keep "rename-perl.pl" in the $HOME/bin folder on any system I go to.
- I make sure it is an executable file
- rename-perl is a SUPER powerful, easy to customize approach for renaming lots of files.
- Suppose you accidentally put the wrong number in a lot of file names

```
$ ./rename-perl s/1988/1993/ baseball*
```

- The "s" notation means "here is a sed script". Sed is a very powerful text manipulation framework. Here, we scan for the "1988" and replace with "1993".

I use rename-perl ALL THE TIME

```
$ ls -la
total 572
drwxr-xr-x  3 pauljohn pauljohn   4096 Jul 26 14:33 .
drwxr-xr-x 11 pauljohn pauljohn   4096 Aug 12 13:07 ..
-rw-r--r--  1 pauljohn pauljohn  52778 Mar  2 13:41 hpcexample-1.lyx
-rw-r--r--  1 pauljohn pauljohn  48753 Feb 26 16:41 hpcexample-1.lyx~
-rw-r--r--  1 pauljohn pauljohn 468913 Mar  2 13:41 hpcexample-1.pdf
drwxr-xr-x  6 pauljohn pauljohn   4096 Aug 12 13:06 .svn
% $
```

- I want to change the basename of all of these files from "hpcexample" to "HPC-Overview"
- A silly Windows/Mac user would click each one individually and re-type
- If there were 1000 files, the Windows/Mac user would be discouraged.

I use rename-perl ALL THE TIME ...

```
$ rename-perl s/hpcexample/HPC-Overview/ *

$ ls -la
total 572
drwxr-xr-x  3 pauljohn pauljohn   4096 Aug 12 13:08 .
drwxr-xr-x 11 pauljohn pauljohn   4096 Aug 12 13:07 ..
-rw-r--r--  1 pauljohn pauljohn  52778 Mar  2 13:41 HPC-Overview-1.lyx
-rw-r--r--  1 pauljohn pauljohn  48753 Feb 26 16:41 HPC-Overview-1.lyx~
-rw-r--r--  1 pauljohn pauljohn 468913 Mar  2 13:41 HPC-Overview-1.pdf
drwxr-xr-x  6 pauljohn pauljohn   4096 Aug 12 13:06 .svn
```

More about Perl

- I've used Perl to manage computer simulations (earmark: "replicator.pl")
- Perl is fairly widely used, plenty of documentation
- Somewhat "dangerous" because of changing styles and bad habits of authors who offer advice on Iternet
- Self defense in "use strict" and warnings pragma.

In the old MP3 days

- One could use Napster (or similar) to download songs in MP3 format
- Challenge: convert those to the right format and put on an audio CD
- Before writing MP3 -> CDROM, it is good to know if all of the songs "fit" on the disk.

```
#!/usr/bin/perl
#This mp3estimate perl program just checks whether a directory
#of mp3's will fit on a disk.

#Determine whether MPEG::MP3Info is present and load it
$no_mp3info = 1;
eval "use␣MPEG::MP3Info;";
unless ($@) {
  undef $no_mp3info;
  use MPEG::MP3Info;
}
use Getopt::Std;
use File::Basename;
```

In the old MP3 days ...

```
my $time_allocated = "74:00";

getopts('ac:dt:o:', \%opts);

($min,$sec)=split(/\:/,$time_allocated);

my @mp3list = <*.mp3>;

for($i = 0; $i <= $#mp3list; $i++) {
  die "$mp3list[$i] does not exist" unless (-f $mp3list[$i]); #Check to see if
       file exists
  $fifo[$i] = $tmpdir . basename $mp3list[$i];  #set the names of the fifos
  $fifo[$i] =~ s/mp3$/cdr/i;    #foo.mp3 -> foo.cdr
  if ($sec) {
    if (-l $mp3list[$i]) {    #mp3info doesn't work on symlinks
      $file = readlink $mp3list[$i];
    } else {
      $file = $mp3list[$i];
    }
    $info = get_mp3info $file; #Let's get the mp3's time
    $totsecs += ($info->{MM}*60) + $info->{SS} + 4; #Calculate total time adding
         a fudge factor of 4 secs
  }
}

$totmin=int $totsecs/60;
$totsec=$totsecs % 60;
if (($totsecs > (($min*60)+$sec)) && $sec) {
  printf "The max time allocated was [%d:%.2d].\n",$min,$sec;
  printf "The total time came to [%d:%.2d].\n",$totmin,$totsec;
```

In the old MP3 days ...

```
}
if ($sec){
  printf "Total time is [%d:%.2d]\n",$totmin,$totsec;
}
```

- Perl names
  - variables: dollar signs ($)
  - arrays: at signs (@)
- Hash symbol (#) begins comments
- "use" accesses modules that are found elsewhere
- Many customs common across computer languages
  - Conditional "if" "then"
  - Note "+=": Add following to previous
  - printf similar to C (note % format for variables)
- Some weird unique-to-Perl

  $@ most recently evaluated result

  my declaration for variables (otherwise global!)

  die, unless stops program gracefully

Template for two-column slide: Blank

- item
- item

Right column