# 2.1 Practice Using the vi Text Editor

## 2.1.1

### Section 1: Moving the Cursor

\*\* To move the cursor, press the **h,j,k,l** keys as indicated. \*\*

^

**k**

< **h** **l** >

**j**

v

Hint:

* The h key is at the left and moves left.
* The l key is at the right and moves right.
* The j key looks like a down arrow

Note: The arrow keys should still work, but using hjkl is much faster once you get used to it

Note: Use **<ESC>** to go to Normal Mode for navigation and manipulation of text

### Section 2: Entering and Exiting VIM

* If you want to exit WITHOUT saving any changes:

***:q!***

* If you want to save the changes and exit:

***:wq***

### Section 3: Text Editing (Deletion)

* While in Normal Mode, press **x** to delete the character under the cursor.

### Section 4: Text Editing (Insertion)

* While in Normal Mode, press **i** to insert text
  + Use ***<ESC>*** to go back to Normal Mode when you are done inserting text

## 2.1.2

### Section 1: Deletion Commands

* In Normal Mode, move the cursor to the beginning of a word and type ***dw*** to delete it

### Section 2: More Deletion Commands

* In Normal Mode, type ***d$*** to delete to the end of the line

### Section 3: On Commands and Objects

* The format for the **d** delete command:

*[number] d object* **OR** *d [number] object*

* + **number** - how many times to execute the command (optional, default=1).
  + **d** - the command to delete.
  + **object** - what the command will operate on (a few listed below).
    - **w** - from the cursor to the end of the word, including the space.
    - **e** - from the cursor to the end of the word, **NOT** including the space.
    - **$** - from the cursor to the end of the line.
    - Pressing just the object in Normal mode without a command will move the cursor as specified in the object list.

### Section 4: An Exception to ‘Command-Object’

* Due to frequency of whole line deletion, designers of Vi decided it would be easier to type **dd** to delete a line.

### Section 5: The Undo Command

* Type **u** to undo the last command, **U** to fix a whole line (return line to its original state)
* Type **CTRL-R** (keep CTRL key pressed while hitting R) to redo commands (undo the undo)

## 2.1.3

### Section 1: The Put Command

* Type **p** to put the last deletion after the cursor
  + If a line was deleted it will go on the line below the cursor

### Section 2: The Replace Command

* Type **r** and a character to replace the character under the cursor

### Section 3: The Change Command

* To change part or all of a word, type **cw**
  + cw will delete the word starting from the cursor and place you in INSERT mode so you can insert text

### Section 4: More Changes Using C

* The change command is used with the same objects as delete: w (word), $ (end of line), etc.
  + The change command works in the same way as delete. The format is:

*[number] c object* **OR** *c [number] object*

## 2.1.4

### Section 1: Location and File Status

* Hold **CTRL** and press **g** to show your location in the file and the file status.
* Hold **SHIFT** and press **g** to move to the bottom of the file
  + Type the number of the line and then **SHIFT-g** to move the cursor to that line

### Section 2: The Search Command

* In Normal Mode, Type **/** followed by a phrase to search for the phrase, use **?** instead to search in the backwards direction
  + To search for the same phrase again, type **n**
  + To search for the same phrase in the opposite direction, hold **SHIFT** and type **n**

### Section 3: Matching Parentheses Search

* Place the cursor on a bracket: (, ), {, }, [, ]. Type **%** to find the matching bracket. The cursor will be on the matching parentheses or bracket.
  + Useful in debugging a program with unmatched parentheses

### Section 4: A Way to Change Errors

* Type ***:s/old/new/g*** to substitute ‘new’ for ‘old’. This only changes the first occurrence on the line.
  + Type ***:s/old/new/g*** to change all occurrences on the line
* To change every occurrence of a character string between two lines, type ***:#,#s/old/new/g*** where #,# are the numbers of the two lines.
* Type ***:%s/old/new/g*** to change every occurrence in the whole file.
  + Or ***:1,$ s/old/new/g*** where $ means the last line in the file.
* To ask for confirmation each time add 'c'
  + E.g. ***:%s/old/new/gc***

## 2.1.5

### Section 1: How to Execute an External Command

* Type **:!** followed by an external command to execute that command

### Section 2: More on Writing Files

* To save the changes made to the file, type ***:w FILENAME***

### Section 3: A Selective Write Command

* To save part of the file, type ***:#:# w FILENAME*** where #,# are the numbers of the two lines. The command will save everything between and including the two lines.

### Section 4: Retrieving and Merging Files

* To insert the contents of a file, type ***:r FILENAME***
  + The contents of the file is inserted starting at where the cursor is located

## 2.1.6

### Section 1: The Open Command

* Type **o** to open a line below the cursor and place you in Insert mode
  + Type an uppercase **O** to open a line above the cursor instead

### Section 2: The Append Command

* Type **a** to insert text **AFTER** the cursor
  + Uppercase **A** to append to the end of the line

### Section 3: Another Version of Replace

* Type a capital **R** and type new text to override old text starting with the character under the cursor.

### Section 4: Set Option

* Set an option so a search or substitute ignores case, type ***:set OPTION***
  + Ignore case: **ic**
  + Highlighting of matches: **hls**
    - Remove: **nohls**
  + Incremental search: **is**. This command will perform the search while you type what you want to search. For example, if you want to search for the word ‘ignore’, you would type ‘/ignore’, but you would first have to type ‘/i’, then it would become ‘/ig’ and so on. Incremental search will first search for ‘i’ and then ‘ig’ and it changes as you type.
    - Remove: **nois**

## 2.1.7

### Section 1: Create a Startup Script

* To enable Vim features each that that Vim is started, you need to create a “vimrc” (vim runtime configuration) startup file. The actual filename is **~/.vimrc**
  + There is a similar “exrc” file for vi
* The vimrc file contains last-line commands without the preceding colon. Simple example of a two-line vimrc file:

syntax off “disable formatting

set ignorecase “ignore case when searching

# 2.2 Permissions

## 2.2 – Section 1: File Permissions

### File Permissions

* Unix systems use a binary value to keep track of the file permissions (security settings) for each file.
* There are **three communities of users**, and **three file permissions** which may be turned ON or OFF for each community.
  + The communities are:
    - **User** (or 'u') - the file's owner.
    - **Group** (or 'g') - the file's group.
    - **Other** (or 'o') - all other users of the system.
  + These permissions are:

Permission: **Read Write Execute**

Letter: r w x

Value: 4 2 1

* + - **Read** permission means that a community may **read a file**.
      * For directories, this means permission to **list the filenames in the directory**.
    - **Write** permission means that a community may **place information in a file or change information that is already in a file (delete, modify, overwrite)**.
      * In the case of a directory, this is permission to **add, rename, or delete files listed in that directory**.
    - **Execute** permission permits a file to be run, if it is a program or script.
      * For directories, this permission is sometimes called ***Search or Passthrough***, and it **allows files in the directory to be accessed**.
* In order to make the 'User' and 'Group' communities mean something, **each file has to be owned by a particular user, and be placed into a particular group**.
  + By default, a **file is owned by the person that created it**, but this can be changed (using the '**chown**' command).
  + Every user of a Unix system is in a group. **Files created by a user are in the same group as the user**, but that can also be changed (using the '**chgrp**' command).
  + You can see your current user ID and group ID using the '**id**' command. You can see a file's owner and group using the '**ls -l**' command or ‘**ls -ld**’ to see a directory’s owner and group

#### Symbolic Representation shown by "ls -l"

* The detailed listings produced by 'ls -l' include the permissions of each file in the first field, with an **extra character at the start** which is used to **represent the file type**.
  + '**-**' for a regular file, '**d**' for a directory, and other symbols for other special file types.
* Symbolic representation uses three sets of symbols, one per community, with three symbols in each set, one per permission. A symbolic representation of a permission takes 9 characters.
  + The communities are given in u, g, o order, and the permissions are written in r, w, x order. If a permission is turned on, then the letter for that permission is shown, otherwise, a dash is used.
* Example:
  + --------- represents all permissions being turned OFF
  + rwxrwxrwx means all permissions are turned ON
  + rw-r----- shows that User has read and write permissions, users in the Group have only read permissions, and Others have no permission.

#### Octal Notation

* Octal uses one digit per community, or three digits to represent the entire permissions setting.
  + In Octal, each digit represents 3 bits. These 3 bits are the permissions (r, w, x) for a community. Therefore, each digit is the sum of the permission values (4 for r, 2 for w, and 1 for x) for all of the permissions that are granted to each community.
* For example, in the permission:

rwxr-x--x

* + The first community, User, has been granted r (4) plus w (2) plus x (1) permission, so the first octal digit is 4+2+1=7.
  + The second community, Group, has been granted r (4) plus x (1) permission, so the second octal digit is 4+1=5.
  + The last community, Other, has been granted x (1) permission, so the last octal digit is 1.
  + Putting this together, the octal representation of the permission above would be written as 751.

## 2.2 – Section 2: Setting Permissions

### Setting Permissions

* Permissions are set using the 'chmod' (change mode) command. The syntax for this command is:

***chmod permission file-list***

* + You can list the permission as an octal number, or by using a modified form of symbolic notation.
    - **Octal**: to set file permission of 744 (rwxr--r--) on the file 'cows', you would use the command 'chmod 744 cows'
    - **Symbolic**: When you are using chmod in this way, the permissions are specified as:
      * **communities** (u, g, o, or a for all)
      * **an operation** (- to remove, + to add, = to set)
        + You can combine several operations using commas

Example: To add read permission for User and remove read for Group and Other: “chmod u+r,go-r file1”

* + - * **permissions** (r, w, or x)
      * Note that you do not affect the permissions that you don’t mention.
        + Example: “chmod g+r file1” you are only changing the read permission for Group, everything else remains unchanged

## 2.2 – Section 3: Using umask to Limit Permissions on New Files

### Using umask

* When a Unix program creates a file, it tells the kernel what permissions it wants on the new file. These permissions may be limited (restricted) by using a value called 'umask'. The umask value is set with the 'umask' command.
* The umask **should be set to the permissions that you do NOT want to grant**. umask **only affects NEW files created, not existing files**.
  + For example, to prevent the system from setting write permission for group, and prevent it from setting ANY permission for other, you would use a umask of 027.
* When using an FTP client program, the files that you copy to your system are affected by the umask. When you copy from your FTP client to a server, the files are affected by the server program's umask, so it doesn't matter what umask setting you try to use.

## 2.2 – Section 4: Review Exercise

**Question 1:** What is the symbolic representation, as displayed by the 'ls -l' command, for a permission setting for a file where the User can read, write, and execute; the Group can read; and the Other users can't do anything?

* rwxr-----

**Question 2:** What is the octal permission number that represents 'rwxr-----'?

* 740

**Question 3:** What is the command to grant all permissions for everyone for the file 'foo' (using octal representation)?

* chmod 777 foo

**Question 4**: What is the command to remove read permission for Group and Other on the file 'green'?

* chmod go-r green

**Question 5:** What is the command to add passthrough permission for everyone for your home directory (~)?

* chmod a+x ~

# 2.3 Redirection and Pipes

## 2.3 – Section 1: Standard File Descriptors

### Standard File Descriptors

* When a Unix shell starts a command, it opens three connections for that command. These three connections are called the 'standard file descriptors'. Many commands use these three connections for input and output.
  + If you are using the shell interactively, then normally all three are connected to your terminal. When your program needs input, it gets it from the keyboard, and when it displays output or error messages, they appear on the screen

|  |  |  |
| --- | --- | --- |
| Number | Name | Short Name |
| 0 | Standard Input | stdin |
| 1 | Standard Output | stdout |
| 2 | Standard Error | stderr |

* + - Example:
      * When you type the command “date”, the date appeared on the screen. This is because stdout of the date command was connected to the screen, this connection was set up automatically by the shell.

## 2.3 – Section 2: Redirecting Standard Output

### Redirecting stdout

* The standard output may be redirected with the symbol '**>**' followed by the name of the file which will contain the output.
  + Example: You can redirect the output of a command to a file by using

***command > filename***

The file will be created if it does not exist

* + - This will overwrite what’s in the file with the output of the command. The symbol '**>>**' will **append (add) to a file** instead of overwriting.

## 2.3 – Section 3: Redirecting Standard Input

### Redirecting stdin

* The standard input may be redirected using ‘**<**’
  + Example: You can redirect the contents of a file into the input of a command by using

***command < filename***

* + - The translate command '**tr**' takes its input from stdin, which is usually the terminal. The command to translate to uppercase is '**tr "[a-z]" "[A-Z]"**'. We can add ‘< filename’ to make the command take its input from the file and translate all the contents of the file to uppercase.
    - Some commands take their input from stdin if there is no file argument on the command-line.
      * Example: you can give a filename argument to the 'cat' command; but if you do not include a filename argument, then the input is taken from stdin. This means ‘cat < filename’ and ‘cat filename” do the same thing

## 2.3 – Section 4: Redirecting Standard Error

### Redirecting stderr

* It is often useful to be able to separate error messages from a program's normal output. For example, you may want to redirect the output from a command into a file, but still have any error messages go to the screen so that you can tell when something is wrong.
  + For example: if you type “cal 1963 8 > filename”, you will receive an error saying that you have given am illegal month value (1963). We see this errors message even though the command’s output was supposed to be redirected to a file. This shows that stderr is separate from stdout
* To redirect standard error we use the same symbols as we use for redirecting
* standard output -- '>' and '>>' -- except that we add the file descriptor number for stderr (2) directly in front of the symbol.

2> file (overwrites)

2>> file (appends)

## 2.3 – Section 5: Connecting Commands with Pipes

### Connecting Commands with Pipes

* We can connect the stdout of one command to the stdin of another using the pipe symbol ‘**|**’. The commands are connected from left-to-right
  + Example: “cal 2018” will display the calendars for the year 2018 and “head” will display the first 10 lines of some input. We can use “cal 2018 | head” to connect the output of “cal 2018” to the input of “head” so it will display the first 10 lines of the output from “cal 2018”

## 2.3 – Section 6: Putting It All Together

### Putting It All Together

* Redirection and pipes are often used together, but it can be tricky getting the syntax right. There are **three important rules to remember**:
  + Redirection only connects commands to files
  + Pipes only work between commands (|).
  + Data goes into the left side of a pipe and flows out the right side.

## 2.3 – Section 7: Review Exercise

**Question 1:** Redirect the output of 'cal -y' into a new file called 'review1'

* cal -y > review1

**Question 2:** Redirect the output of 'date' so that it is added to the end of the file 'review1'.

* date >> review1

**Question 3:** Display a listing of the directory '/etc' one screen at a time.

* ls /etc | more

**Question 4:** The file 'threelines' contains 3 lines of text. Display the middle line only on the screen. You are not allowed to use the more command.

* head -2 < threelines | tail -1

**Question 5:** Display the current month using 'cal', translating to uppercase using 'tr'.

* cal | tr ‘[a-z]’ ‘[A-Z]’

**Question 6:** Run the command 'mkdir old' and redirect any error messages into the file 'errors'.

* mkdir old 2> errors

**Question 7:** Search the file 'data' for all of the lines that contain the pattern 'linux' and put those lines in the file 'matches'.

* grep ‘linux’ < data > matches

**Question 8**: The file 'numbers' contains a list of numbers. Write a command to place the largest one of those numbers into the file 'largest' (there should be nothing else in that file). Do not use the 'head' command in your answer.

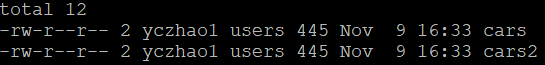
* sort -n numbers | tail -1 > largest

# 2.4 Linking Files & Directories

## 2.4 – Section 1: Hard Links

### Linking Files

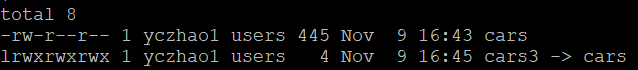
* The 'ln' command creates links to files. A link may simply be considered an additional name for a file. The link may be in the same directory or a different directory than the original filename.
* In order to create a link, we use the 'ln' command followed by the current name of the file, followed by the additional name we wish to give it.
  + Example: “ln cars cars2” will create a link named car2.
    - When you use “ls -l”, you will see that the “number of links” shown after the permissions is 2 for both filenames. This means that the file has two links to it, one with the name ‘cars’ and one with the name ‘cars2’. The statistics for these two are identical, ‘cars’ and ‘cars2’ are two names for the same physical file.



* + - Using “ls -i” will give the ‘inode number of filenames. ‘inode’ stands for information node, and contains all the information about the file (permissions, owner, group, create date-time, last modified date-time, etc.). Each file has a unique inode, since ‘cars’ and ‘cars2’ are just names for the same file, their inode is the same.
    - If you remove ‘cars’, the file ‘cars2’ will be still there. **A file is not considered deleted until ALL hard links are removed.**
      * Files are not physically deleted from a hard drive, and maybe recovered, until their specific inodes are overwritten with new data

## 2.4 – Section 2: Symbolic Links

* Symbolic links give another name to a file, but a **file can be deleted even if there are remaining symbolic links**. Symbolic links are also called soft links.
* In order to give a file a symbolic link, the **'ln' command is used with the '-s'** option.
  + Example: “ln -s cars cars3” will create a symbolic link named cars3
    - Using “ls -l”



* + - * Notice that the first character is l which means symbolic link.
      * **The symbolic link 'cars3' DOES NOT point to the inode that 'cars'** is pointing to. Instead, 'cars3' points at the name 'cars'.
      * **'cars3' will continue to point to the name 'cars', even if 'cars' is deleted.** If we now create a file called 'cars', 'cars3' will automatically point to the new file.

## 2.4 – Section 3: Linking to Directories

* Directories can be linked only symbolically, except by the system administrator.
* It can be very cumbersome to type a long pathname. If we were often accessing the files in a directory, we could create a link.
  + E.g. we can create a symbolic link for the directory “\1\2\3\4\5\6\7\8\9” by using

ln -s \1\2\3\4\5\6\7\8\9 nine

Whenever we want to reference the folder, let’s say for copying a file to home directory, we can just do “cp nine/filename ~”

## 2.4 – Section 4: Review Exercise

**Question 1**: Create a subdirectory called autos within the current directory.

* mkdir autos

**Question 2**: Create an additional name (hard link) to the file 'cars'. The link should be called 'cars.link' and should be in the directory 'autos'.

* ln cars autos/cars.link

**Question 3**: Create a link to the directory '~uli101/2018c/sample\_dir3/linked\_directories/sample\_files'. The link, called 'samples', should be in the current directory.

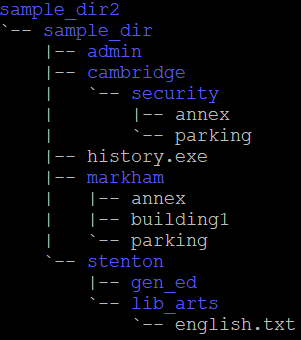
* ln -s ~uli101/2018c/sample\_dir3/linked\_directories/sample\_files samples

**Question 4**: There is a command called 'showtree' in the directory you just linked to. Execute this command using the newly created 'samples' link

* sample/showtree

# 2.5 Command Summary

## 2.5 – Part 1



**Question 1**: sample\_dir2 is a subdirectory of your home directory. Move to stenton (make stenton your current directory), using a relative-to-home pathname.

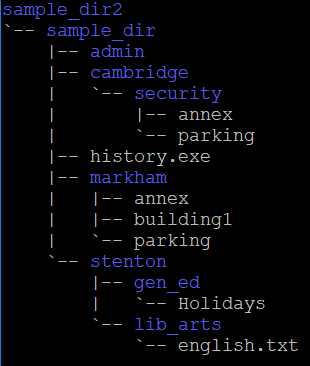
* cd ~/sample\_dir2/sample\_dir/stenton

**Question 2**: Your current directory is stenton. Display detailed information about the markham directory itself, not the files within it.

* ls -ld ../markham

**Question 3**: Your current directory is stenton. Place a calendar listing for the month of December 2018 into a file called Holidays within the gen\_ed directory.

* cal 12 2018 > gen\_ed/Holidays



**Question 4**: Your current directory is stenton. Append (add) the current date and time to the file called Holidays

* date >> gen\_ed/Holidays

**Question 5**: Display your userid

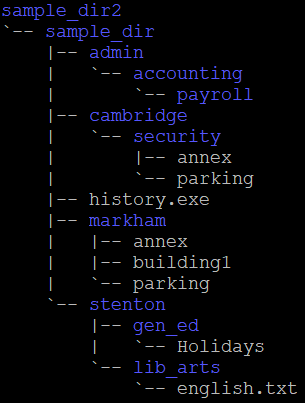
* whoami

**Question 6**: Display all the users logged into the system.

* who

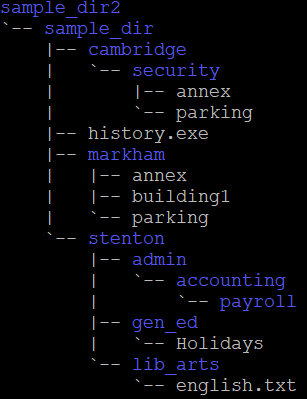
**Question 7**: Your current directory is stenton. Create a new directory called payroll within a new directory called accounting, which should be in admin.

* mkdir -p ../admin/accounting/payroll



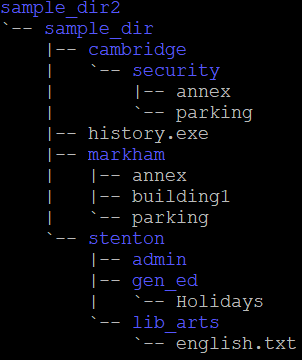
**Question 8**: Your current directory is stenton. Move the admin directory under stenton.

* mv ../admin .



**Question 9:** Your current directory is stenton. Remove the accounting directory.

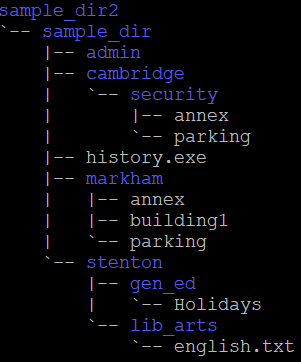
* rm -r admin/accounting



**Question 10**: Your current directory is stenton. Move admin back to sample\_dir.

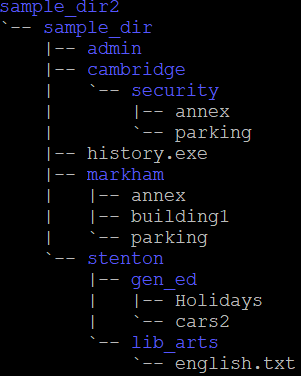
* mv admin ..

## 2.5 – Part 2



**Question 1**: Your current directory is stenton. Copy the file ~uli101/2018c/cars to gen\_ed, but calling the new file cars2.

* cp ~uli101/2018c/cars gen\_ed/cars2

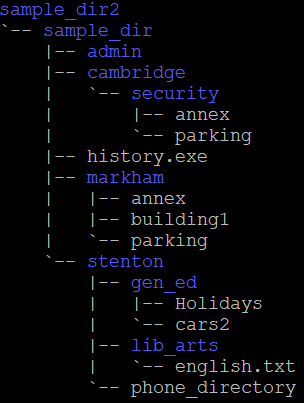


**Question 2**: Your current directory is stenton. Display the entire cars2 file.

* cat gen\_ed/cars2

**Question 3**: Your current directory is stenton. Copy the first 300 lines of file ~uli101/2018c/phonebook to a new file called phone\_directory in stenton.

* head -300 ~uli101/2018c/phonebook > phone\_directory



**Question 4**: Your current directory is stenton. The phone\_directory file contains phone number and other information, sorted by last name, which is the first field in each record. Display phone\_directory one screen at a time, until you find the record for 'Bruce Byce'. Make a note of his 4-digit telephone extension (you'll need it for the next question), then quit the command.

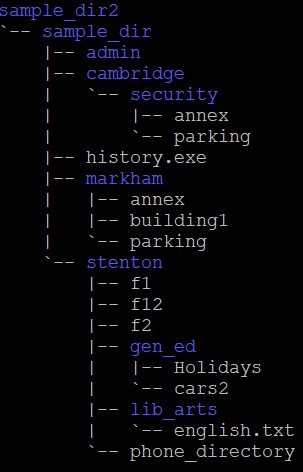
* more phone\_directory

**Question 5**: Enter the 4-digit telephone extension for 'Bruce Byce'.

* 2215

**Question 6:** Your current directory is stenton. Create empty files called f1, f2, and f12 (in that order), within stenton.

* touch f1 f2 f12



**Question 7**: Your current directory is stenton. Move to sample\_dir.

* cd ..

**Question 8**: Your current directory is sample\_dir. Display all the files within sample\_dir (and it's subdirectories) that are larger than 10k bytes.

* find . -size +10k

**Question 9**: Your current directory is sample\_dir. Edit the file cars2 using vi, and delete a line or make any other changes you wish, and save the file.

* vi stenton/gen\_ed/cars2

**Question 10**: Your current directory is sample\_dir. Display the differences between the files ~uli101/2018c/cars and cars2:

* diff ~uli101/2018c/cars stenton/gen\_ed/cars2

## 2.5 – Part 3

**Question 1**: Display the contents of the file '.bashrc' within the home directory of your instructor (userid eric.brauer).

* cat ~eric.brauer/.bashrc

**Question 2**: Your current directory is sample\_dir. Display the 2nd line of the file cars2, translated to uppercase (hint: start with a 'head' command).

* head -2 stenton/gen\_ed/cars2 | tail -1 | tr ‘a-z’ ‘A-Z’

**Question 3**: Your current directory is sample\_dir. Display cars2 sorted numerically by the 4th field.

* sort -n -k4 stenton/gen\_ed/cars2

**Question 4**: Your current directory is sample\_dir. Display only the 1st field (last name) of each record in phone\_directory using 'cut'. Note that the field delimiter is a space.

* cut -d" " -f1 stenton/phone\_directory

**Question 5**: Your current directory is sample\_dir. Find the records in cars2 that contain the string "ford", and display the number of bytes in the result.

* grep "ford" stenton/gen\_ed/cars2 | wc -c

**Question 6**: Your current directory is sample\_dir. Change the permissions (using octal) for cars2 so that an 'ls -l' listing would show rwxr-x--x.

* chmod 751 stenton/gen\_ed/cars2

**Question 7**: Your current directory is sample\_dir. Add the permission (using symbolic) for gen\_ed so that all users can access the file cars2.

* chmod a+x stenton/gen\_ed

**Question 8**: Your current directory is sample\_dir. Create a hard link to cars2, called cars3 within sample\_dir.

* ln stenton/gen\_ed/cars2 cars3

# 2.6 Linux Processes

## 2.6 – Section 1: Process Concepts

### Process Concepts

* An operating system controls the hardware and software resources of a computer system.
  + These software resources (programs) work with the various hardware resources such as the CPU, memory, input/output devices, and networking connections.
* Since the computer may have only one or a few processors to manage computer resources as well as allow many end-users to run their own programs, a mechanism is required to schedule and allocate CPU time for these "simultaneously" running programs.
  + In Unix and Linux operating systems, a **running program (any type of running program)** is referred to as a"**process**".
  + In Linux and Unix, whenever a command is issued or a program executed, a process is initiated.
    - For identification purposes a **PID** or "Process IDentification" number is assigned to the process. This allows the user to keep track of the process and abort it if necessary.
    - To view the processes that are running, type **ps**
      * The first column (PID) shows the process number. The right column shows the name of the program that is running.
    - Processes are related to each other. To view the relationship between the processes: type **ps f -f**
      * You can use this command output to trace the relationship between a process (**PID**) and its parent process (**PPID**).
      * In another column, you will see either "S" or "R". These characters represent the **"state" of the process**.
        + The **state "R"** indicates that the process is "Runnable" which is either running, or queued by the scheduling program to be run.
        + The **state "S"** indicates that the process is sleeping, waiting for the child process(es) to end, in order to "wake-up" or "run".
        + **T** - Stopped
        + **X** - Terminated or "dead" process
        + **Z** - Defunct process, but still using resources (dead, but still "running" - like a "Zombie")
    - You can use the "**-u**" option with the ps command to view all processes relating to a user (like "yczhao1")
      * Your Linux session (shell) was run from the sshd process when you connected to the server via your SSH program. This process, in turn, can be traced back to **PID 1**. This is a special program called "**init**" which is used to start programs after the **kernel (PID 0)** was started following the server "boot-up".
        + The "init" program launches the "start-up" programs and controls the processes. There are many other processes besides sshd. Many of the programs that the "init" program runs are independent of the shell. In other words, the program or process "disconnects itself" from the terminal and runs in the background. These types of programs are referred to as "**daemons**".
    - A user can issue the "**ps -e**" option to view every process on your system.
      * There are many programs or processes that can be identified as "daemons" simply by viewing a "d" at the end of the program name.
    - The "**pstree**" command will "draw" a tree-diagram showing the relationships of all processes running on a computer system.
      * You can add the username argument to show the processes for the user only
      * You may or may not need the **-A** option, depending on whether your session is set up to support the ASCII character set.

## 2.6 – Section 2: Managing Processes

### Managing Processes

* When you issue a command in Unix / Linux, a new child process is created and executed. The parent process, your shell, is suspended or "sleeps" until the child process terminates.
  + Your terminal may be "tied-up" while running long and time-consuming programs or commands.
  + You can have child processes run in the background to "free-up" the parent process or terminal. **Processes that are run in the background, but are still "tied" to a terminal, are referred to as "jobs".**
    - To run a command in the background, type an ampersand "**&**" symbol after the command or program name.
      * Example: To demonstrate background processes, we will use a command that never ends, "tail -f filename". This command will display the last 10 lines of the specified file and will stay active to display any lines that are added to the file. This is a very useful command for system administrators to view changes to a file, such as a log file, in real time.
        + You should have seen a number in square brackets, which is the "job number", followed by the process ID of the new process.
        + You received terminal control immediately, even though the "tail" is still running. This is a big advantage of using background processes.
        + If we use it again, we would have two background jobs running.
  + Type **jobs** to view information regarding processes or jobs running in the background
    - **[#]** Indicates job number
    - **+** Indicates most recent job sent to background
    - **-** Indicates second most recent job sent to background
  + You can bring the most recent job to the foreground by simply issuing the command "**fg**" by itself.
    - You may also want to specify the job # that you want to bring to the foreground. (We could also use PID instead of job #).
      * Example: **fg %2** will bring the second job to the foreground
        + You can terminate a foreground process by using **<ctrl>-c**, or stop it from running using **<ctrl>-z**
  + You can continue a job "running" in the background. This allows the user to run long commands or programs in the background without occupying the terminal.
    - Example: **bg %2** will continue the second job in the background
  + You can terminate a job that is running in the background by issuing the "kill" command specifying either the job number or PID.
    - Unpriviledged (non-administrator) users can only kill their own processes or jobs. Root (administrator) can kill any process.
    - Example: **kill %3** will terminate the third job in the background

|  |  |  |
| --- | --- | --- |
| **Sig#** | **SigName** | **Description** |
| 1 | SIGHUP | Terminal Disconnect |
| 2 | SIGINT | Terminal Interrupt (<Ctrl>-c) |
|  | SIGTSTP | Stop process |
| 15 | SIGTERM | Terminate process Gracefully (kill pid or %job) |
| 9 | SIGKILL | Terminate process Forcefully (kill -9 pid or %job) |

* + - * The kill command actually sent a "**signal**" to the job or process that was running in the background. A signal is a special code that is sent from one process to another. Some signals are:
        + Example: **kill -SIGTSTP %1** will stop job 1, this is the equivalent of <ctrl>-z for a foreground process
        + Note that there are **two signals that can be used to kill a job**.

**Signal #15 (SIGTERM)** is the default kill signal, and should always be tried first.

**Signal #9 (SIGKILL)** should only be used as a last resort, if signal #15 doesn't work. This would mean that either the program was written to ignore signal #15, or there is some kind of a problem. Signal #9 cannot be ignored, but will not clean up resources, such as memory and page space.

## 2.6 – Section 3: Review Exercise

**Question 1**: Enter the command that is used to display currently running processes, including an option to display PPID's (parent process id's).

* ps -f

**Question 2**: Enter the command that is used to display the background jobs that are currently running.

* Jobs

**Question 3**: Enter the command to bring job #2 to the foreground.

* fg %2

**Question 4**: Enter the command to terminate job #3.

* kill %3

**Question 5**: Enter the command to terminate job #3, assuming "kill %3" didn't work.

* kill -9 %3