Lab 3

UNIX Commands

General Format of UNIX Commands – man entries

- Because of the way in which UNIX evolved, there are very few standards in the command syntax. In general, command formats follow a convention:

  `$ command –options argument1 argument2` ... etc

- However, this is not always true. Also, the same option may mean totally different things to different commands.

- As a result, most commands have a manual entry. The manual entry is a file that is stored in a particular directory (that is customizable on every system) that contains documentation for the command. To view this documentation for a particular command, you can use the **man** command. The following example looks at the manual entry for the ls command:

  ```
  $ man ls
  ```

- Manual entries usually include the proper syntax for the command, and explain all of the options and arguments that the command expects. It may also include special notes and/or about warnings about certain uses of the command.

Copying Files – cp command

- To create a copy of a file, you can use the **cp** command. The format is as follows:

```
$ cp source_file target_file
```

- The above command creates a copy of the file in the current working directory. By using pathnames, cp can create a copy of a file in a different directory as follows:

```
$ cp source_file ~/bin/target_file
```

- The above example creates a copy of the file called target_file in my home directory. If you want to create a file of the same name, but in a different directory, you can use the dot (.) instead of a target file name:

```
$ cp same_filename ~/bin/.
```

- **Note:** The cp command leaves the source file alone, but it still can potentially be destructive because it will overwrite any target file that already exists. To prevent possible data loss, you can use the –i option of cp that warns the user if the target file exists. Below is a sample:

```
$ cp –i source_file target_file
```

Moving (or Renaming) Files – mv command

- The cp command creates a new file that is an independent copy of a file without changing the source file. By contrast, the **mv** command logically makes a copy of the file and deletes the original source file. Internally, it does this by simply changing the filename (or renaming the file) in the inode. The format is as follows:

```
$ mv old_file new_file
```

- By using pathnames, mv can place a file in a different directory as follows:

```
$ mv old_file ~/bin/new_file
```

- The mv command can also move the file to a different directory using the same name by using the dot (.) instead of a target file name:

```
$ mv same_filename ~/bin/.
```

- **Note:** The mv command also will overwrite any target file that already exists. Therefore, you can also use the –i option of mv that warns the user if the target file exists. Below is a sample:

```
$ mv –i old_file new_file
```

Displaying Files - pg and more commands

- Thus far, we have seen how to view the contents of an ordinary file using the cat command and the vi editor. There are some other commands that better facilitate looking at files (particularly large files).

- The **pg** command displays a file a page at a time. The file can be scrolled using the enter key or searched using the /<search pattern> text. On AIX systems, there is a special file called /etc/filesystems that stores information on all of the filesystems. This is generally a large file, so we can use it as an example:

  `$ pg /etc/filesystems` (type <ctrl-c> to cancel)

- The **more** command is similar to the pg command. The main difference is the fact that the file can be scrolled either a line at a time with the enter key, or a page at a time with the space bar. Below is an example:

  `$ more /etc/filesystems` (type <ctrl-c> to cancel)

- Often, the more command is combined with another command using the pipe to control the output:

  ```
  $ cat /etc/filesystems | more
  ```

Displaying Files – head and tail commands

- There are a few other commands that can be useful when displaying large text files. The **head** command, by default, displays the first ten lines of a file. For an example, we can use the /etc/filesystems file again:

  ```
  $ head /etc/filesystems
  ```

- The head command is useful for glancing at the format of a large file without having to go into it.

- The **tail** command displays the last ten lines of a file by default:

  ```
  $ tail /etc/filesystems
  ```

- There are not too many cases where the tail command by itself is useful. However, some files are continuously updated. The –f option of the tail command will update the screen as output is added to the file. An example of this can be found with the system log (in AIX it is under /var/sysadm/messages). The system log is constantly updated, and the command below allows the log to be followed:

  `$ tail –f /var/adm/messages` (type <ctrl-c> to cancel)

Searching for Text Patterns – grep command

- The **grep** command is very common utility used for searching out patterns in text files. There are many options and uses of the grep command that we will expand upon throughout the class. For now, we can look at some simple examples.
- Let's say that I have a file called birthdays in which I store all of the names and dates of the birthdays that are important to me. If I want to find a person's birthday without looking at the entire file, I can use the grep command as follows:

```
$ grep India birthdays
India 12/10
$
```

- The –i option of the grep command says to ignore case:

```
$ grep –i india birthdays
India 12/10
$
```

- To look at all of the January birthdays:

```
$ grep 01 birthdays
Matt 01/28
Rob 01/27
```

More Searching for Text Patterns – egrep command

- The **egrep** command allows for multiple patterns to be searched. These patterns are separated by an or (|) symbol in the text string that is represented in single quotes (‘’). Below is an example with the –i option:

```
$ egrep –i ‘rob|india’ birthdays
India 12/10
Rob 01/27
$
```

- One of the more useful ways in which to use grep and egrep is to redirect the input of another command using the pipe (|) symbol. Therefore, commands which generate a great deal of output can be filtered for the relevant data. Below is an example with the df command. The df command produces output for every filesystem (with the –k option specifying the size in kilobytes). If you are only interested in the home filesystems, then the command below will filter the output (in this case only 1 home filesystem):

```
$ df –k | grep home
/dev/Less          1228800    1137740     8%       809       1% /home/ess
$
```

Sorting Text Files – sort command

- The sort command takes the lines of input to a specified file and sorts them according to the options. With the sort command, the format is important:

  $ sort [-t delimiter] [+field[.column]] [-options] file

- Some of the main options are as follows:
  - d – sorts in dictionary order (alphabetical)
  - n – sorts in numeric order
  - r – sorts in reverse order

- Sort has some common sense defaults, such as the space being the delimiter and the first field and column. Therefore, to sort the names in the birthday file in alphabetical order, you can use the following:

```
$ sort –d birthdays
```

- However, if you wanted to sort the file by the birth month in reverse order, you could use the following:

```
$ sort +1 –rn birthdays
```

Command I/O Redirection

- By default, the sort command generates output to what is referred to as **standard output**. In most cases, standard output means the screen. As we have seen, by using the redirection (>), the output of a command can be redirected to a file. Therefore, to create a file of the birthdays in alphabetical order, you can use the following:

```
$ sort –d birthdays > alphabetical_birthdays
```

- Standard output is known as a **file descriptor** that is represented by (>). There are two other standard file descriptors that are associated with every command. They are **standard input**, which is represented by (<), and **standard error** that is represented by (2>).
- An example of redirecting standard input is using a file for the input of the mail command. Below, we can see how this technique sends a file to someone via mail:

```
$ mail user1 < letter
```

- All commands may potentially generate error messages. These messages are sent to standard error (once again, usually the screen). These can be redirected to a file as well. For example, if we try to cat a file that does not exist, we will get an error. In the example below, this error is redirected to an error log:

```
$ cat non_existent_file 2> error_log
```

Combined Command I/O Redirection

- As mentioned before, using (>) to redirect standard output is a potentially dangerous technique. If a file by the same name exists, the (>) will overwrite the file. To solve this, redirection can be instructed to append to rather than overwrite a file using (<<) for standard input, (>>) for standard output and (2>>) for standard error. Below is an example:

```
$ cat file >> append_file
```

- **Note:** If the file specified in the >> or 2>> does not exist, one will be created automatically.
- By combining these I/O redirection commands, commands can utilize several files. In the example below, a file will be used for input, a new file will be created for output and an error log will be appended:

```
$ custom_command < infile > outfile 2>> error_log
```

- The pipe (|) operator that we have discussed previously has a built-in I/O redirection combination mechanism. It takes the standard output of one command and uses it as (or “pipes” it to) the standard input of another command. Another example of a combination takes the output of the df command, pipes it to a search for home filesystems, sends the standard output to a new file and appends the standard error to an error log:

```
$ df -k | grep home > home_filesystems 2>>error_log
```

Metacharacters and Wildcards

- Metacharacters are characters that have a special meaning to the shell. We mentioned earlier that the metacharacters (*?></:$![]{}| \`”) should not be used in filenames. This will become abundantly clear as we begin to use them with shell commands and programs.
- The metacharacters make the foundation of what is known in UNIX as regular expressions. The regular expressions form a language of pattern matching in text that is generally understood by most UNIX programs. These are used to greatly enhance shell commands and programs.
- Wildcards are a type of metacharacter that are used to substitute filenames that are of certain criteria. They are used with many UNIX commands. The wildcards are as follows:

  * – substitutes all characters

  ? – substitutes a single character

  ! – substitutes all characters except this character

  [ ] – substitutes a range of characters

Examples of Wildcards – find and grep

- Many of the commands that we have learned up to this point can and often do make use of wildcards. Below are examples:

  `$ ls test?` (lists files with test + 1 char such as test1 & test2)

  `$ cp * ~/.` (copy everything in the current directory to my home directory)

  `$ rm –r *` (recursively remove everything in current directory ***VERY DANGEROUS)

  `$ ls *[1-5]` (lists any files ending in 1-5 such as matt1 and test5)

  `$ mv [!n]* ~/.` (move all files except beginning with an n to my home directory)

- The find command is a powerful tool with many options. It searches a directory hierarchy for files and can perform a number of different functions on the files it finds. In the simple example below, find locates all of the files in the current directory structure that begin with the letter m and prints them to the screen:

```
$ find . –name m* -print
```

- The grep command is also interesting when combined with wildcards. For an example, let’s say that I have a sub-directory under my home directory called personal where I keep information about people. If I wanted to know everything in there about my wife India, grep would print a line for each occurrence of the name and also print which files in which they reside. Below is the example:

```
$ grep India ~/personal/*
/u/ds59478/personal/birthdays:India 12/10
/u/ds59478/personal/email:India IndiaRS@aol.com
/u/ds59478/personal/phone:India (972) 506-0843
$
```