Rento Saijo

William Tarimo

Data Structures

11 February 2025

Summary/ Notes + Learned Code/Commands

**Introduction to the UNIX Operating System**

From this tutorial, I’ve learned that UNIX is a robust, multi-user, and multi-tasking operating system that has evolved since its inception in the 1960’s. It comes in various versions such as Solaris, GNU/Linux, and MacOS X, but all versions share common features. At its core, UNIX is composed of three main parts: the kernel, which manages memory, processes, and system calls; the shell, a command line interpreter that serves as the interface between the user and the kernel (with features like filename completion using the [Tab] key and command history retrieval); and the suite of programs that perform user tasks.

An important concept highlighted is that everything in UNIX is either a file or a process, which the hierarchical directory structure starting from the root directory (/) organizing these files. For example, when you type a command like ``rm file``, the shell finds and executes the corresponding program by invoking the kernel, which then processes the removal of the file before returning you to the command prompt.

**UNIX Tutorial One**

I learned some ways to navigate and manage the file system using basic UNIX commands. One of the first commands introduced was ``ls``, which lists the contents of the current working directory; to see every file­–including the hidden ones­–you can use the ``-a`` option: ``ls -a``.

The tutorial then covered how to create directories using the ``mkdir`` command. Upon creation, we can enter this directory by using ``cd`` followed by the directory’s name. Also, there are two special entries: ``.`` (the current directory) and ``..`` (the parent directory). Additionally, simply typing ``cd`` without any arguments/options will take you back to your home directory.

Another key concept is understanding pathnames. The ``pwd`` command displays the absolute path of your current directory. To avoid any confusion, it seems like the standard convention to use ``~`` as shorthand for your home directory.

**UNIX Tutorial Two**

I learned the basics of manipulating files and viewing/searching file contents. First, I learned how to copy files using ``cp``; next, we can use ``mv`` to move (and rename) files. Additionally, we can remove files with ``rm`` and directories with ``rmdir``; one thing to note about the ``rmdir`` is that it can only remove directories that are empty.

To view files, the simplest is ``cat``, which dumps the entire file to the terminal; typically, for longer files that extend past the screen, the ``less`` command is more useful since it displays one page at a time and allows you to scroll with the space bar and exit with [q]. Some other helpful commands that I learned were ``head`` and ``tail`` to view the beginning and end of files, respectively.

Finally, to search within files, you can use ``/`` while using ``less`` to look for keywords; for a more automated approach, the ``grep`` command searches files for speicifc words or patterns. Since ``grep`` is case-sensitive by default, you can use the ``-i`` option to perform a case-insensitive search. Some other useful options mentioned were ``-v`` to display lines that do not match the pattern, ``-n`` to show line numbers along with matching lines, and ``-c`` to output only the count of matching lines. Another command I learned was the ``wc``, which can count words, lines, or characters in a file.

**UNIX Tutorial Three**

I learned about redirections and pipes. First, I learned that running the ``cat`` command without a filename causes it to read from the keyboard until I signal the end-of-file with [^D]. By redirecting output with ``>``, I can save what would normally display on the screen into a file; I also learned that using ``>>`` instead of ``>`` appends new content on an existing file instead of overwriting it.

Another key concept was concatenating files; by combing files using ``cat`` with multiple filenames, I can merge them into a new file. Once merged, I can manipulate the data further. The ``sort`` command can be used to arrange the data alphabetically or numerically. Additionally, using ``|`` allows me to send the output of one command directly into another command as its input, eliminating the need for temporary files. While doing this, I also learned another command ``who`` which returns the list of currently logged-in users.

**Command** Usage

**ls** Lists files and directories in the current working directory.

**ls -a** Lists **all** files—including hidden files (those starting with a dot).

**mkdir directory** Creates a new directory named *directory*.

**cd directory** Changes the current working directory to the specified *directory*.

**cd** or **cd ~** Returns you to your home directory.

**cd ..** Changes to the parent directory of the current directory.

**pwd** Prints the full pathname of the current working directory.

**cp file1 file2** Copies *file1* to a new file called *file2*.

**mv file1 file2** Moves or renames *file1* to *file2*.

**rm file** Removes (deletes) the specified *file*.

**rmdir directory** Removes the specified *directory* (only if it is empty).

**cat file** Displays the contents of *file* on the screen.

**cat file1 file2 > file0** Concatenates *file1* and *file2* and writes the combined output into *file0*.

**less file** Displays the contents of *file* one page at a time (useful for reading long files).

**head file** Shows the first 10 lines of *file* (use head -n number file for a specific number of lines).

**tail file** Shows the last 10 lines of *file*.

**grep 'pattern' file** Searches *file* for lines containing *pattern*. Options include:  -i (case-insensitive), -v (invert match), -n (include line numbers), and -c (count matching lines).

**wc file** Counts lines, words, and characters in *file*. Options include:  -l (lines) and -w (words).

**command > file** Redirects the standard output of *command* to *file* (overwrites *file*).

**command >> file** Appends the standard output of *command* to *file* without overwriting existing content.

**command < file** Redirects the standard input for *command* to come from *file* rather than from the keyboard.

**command1 | command2** Pipes the output of *command1* directly into *command2* as input, allowing you to chain commands without creating temporary files.

**sort** Reads input (from standard input or via redirection) and sorts lines alphabetically or numerically.

**who** Displays a list of users currently logged into the system.