Computer Practicum and Computer Science Fundamentals

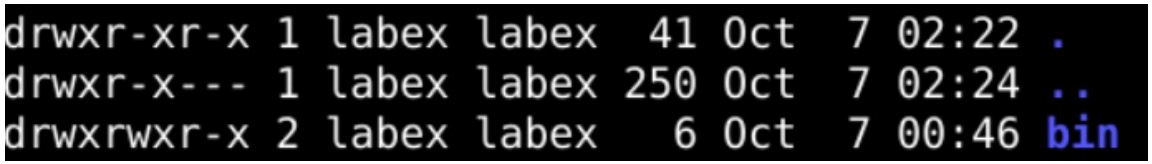
2. Linux - working with text files

Linux Basic Commands - Recap

- **man** - displays the manual
- **clear** - clears the terminal
- **ls** - lists directory contents
- **cd** - changes the current directory
- **pwd** - displays the path of the current directory
- **mkdir** - creates a directory
- **rmdir** - removes a directory
- **touch** - creates a file
- **rm** - removes a file (or directory)
- **cp** - copies a file
- **mv** - moves a file
- **history** - displays command history

File Permissions - Recap

In Linux, there are three classes of users: the owner (who created the item), the group, and others.



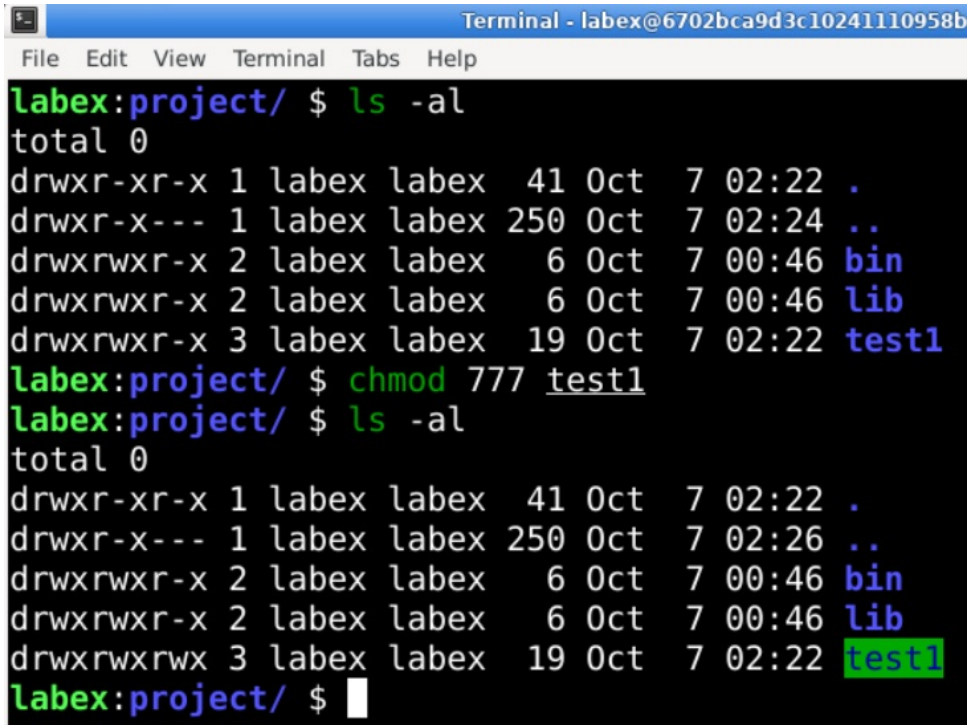
The `ls -l` command displays file permissions (the first 10 characters). The first character indicates the file type (`d` - directory, `-` - regular file); other types also have their own symbols.

The next nine characters are divided into groups of three, one for each user class, in the following order. The symbols are: `r` - Read, `w` - Write, `x` - eXecute. A `-` indicates that the corresponding permission is not granted.

Permissions can also be specified using octal notation (4 - read, 2 - write, 1 - execute).
What are the octal permission values shown in the image?

Linux Commands - chmod - Recap

chmod [permissions] [directory/file] - allows us to change permissions for directories and files. The permissions argument can specify the permissions for each user class using three octal values.



Working with Text Files

The `nano` Command

- The **nano** command launches a simple text editor that runs directly in the terminal. It can be used to create new text files or edit existing ones.
- Example: `nano file.txt`
- If the file does not exist, the editor creates it when we save it. If the file already exists, its current contents are opened for editing.
- While editing, we can enter text directly using the keyboard. The most common keyboard shortcuts are displayed at the bottom of the screen.

The `nano` Command

- The most important keyboard shortcuts:
  - `Ctrl + O` - saves changes to the file (Write Out).
  - `Enter` - confirms the file name when saving.
  - `Ctrl + X` - exits the editor.
  - `Ctrl + W` - searches for text in the file.
  - `Ctrl + K` - cuts the current line.
  - `Ctrl + U` - pastes the most recently cut text.
- When exiting the editor, the program may ask whether we want to save our changes. If we are editing the file for the first time, we must also confirm its name when saving.
- Shortcuts in `nano` are usually displayed using the `^` symbol. For example, `^X` means `Ctrl + X`.

The `cat` Command

- The **cat** command prints the contents of one or more files to standard output, which is usually the terminal. It is most commonly used to quickly view short text files.
- Example: `cat file.txt`
- The `cat` command does not open an interactive text editor or modify the contents of a file. If a file does not exist or cannot be read, it displays an error message.

The `cat` Command

- The `cat` command can be given multiple files. Their contents are printed consecutively, in the order in which the files are specified.
- Example: `cat first.txt second.txt`
- The contents of multiple files can also be combined into a new file by redirecting the output.
- Example: `cat first.txt second.txt > combined.txt`

The `cat` Command

- The `cat` command can also display special characters and number lines.
- Example: `cat -n notes.txt`
- The `-n` option numbers all output lines.
- Example: `cat -b notes.txt`
- The `-b` option numbers only non-empty lines.
- Example: `cat -A notes.txt`
- The `-A` option also displays certain characters that are not normally visible, such as tabs and end-of-line markers. This is useful for checking the formatting of text files.

The `more` Command

- The **more** command allows us to view long text files one portion at a time. Instead of printing the entire contents at once, it displays as much text as fits on the terminal screen.
- Example: `more file.txt`
- When the current screen is full, the output pauses. We can then continue viewing the contents.
- Limitation: `more` is primarily designed for moving forward through the text. For more flexible navigation, especially when moving backward or searching the contents, `less` is more suitable.

The `more` Command

- While viewing the contents, we can use the following keys:
  - `Space` - moves forward by one screen.
  - `Enter` - moves forward by one line.
  - `b` - moves backward by one screen, if supported by the version being used.
  - `q` - quits the viewer and returns to the terminal.
- Note: The set of supported keys may vary slightly between program versions and terminal environments.

The `more` Command

- The `more` command can also be used to view the output of another command. We use the pipeline operator `|`, which passes the output of the first command as input to the second command.
- Example: `ls -l | more`
- The `ls -l` command displays a detailed listing of the contents of the current directory, while `more` allows us to view the output one portion at a time.
- This approach is particularly useful when a command produces more lines than can be displayed on the terminal screen at once.

The `less` Command

- The **less** command allows us to interactively view long text files. Like `more`, it displays the contents in portions, but provides more flexible navigation, including moving both forward and backward.
- Example: `less file.txt`
- The command opens a viewer for the file's contents. It does not modify the file, and when we finish viewing it, we return to the terminal.
- Advantage of `less`: it allows us to efficiently inspect long outputs without printing the entire contents at once or opening the file in a text editor.

The `less` Command

- While viewing the contents, we can use the following keys:
  - `Space` - moves forward by one screen.
  - `Enter` or `j` - moves forward by one line.
  - `b` - moves backward by one screen.
  - `k` - moves backward by one line.
  - `g` - moves to the beginning of the file.
  - `G` - moves to the end of the file.
  - `q` - exits the viewer.
- Note: The set of supported keys may vary slightly between program versions and terminal environments.
- Like `more`, `less` can also be used to view the output of another command.

The `head` Command

- The **head** command displays the beginning of a file. By default, it displays the first ten lines, which is useful when we want to quickly check the contents of a longer file without displaying the entire contents.
- Example: `head file.txt`
- The number of lines displayed can be specified using the `-n` option.
- Example: `head -n 5 file.txt`
- We can also view the beginning of multiple files.
- Example: `head -n 3 first.txt second.txt`
- The command displays the first three lines of each file and indicates which file each output belongs to.

The `tail` Command

- The **tail** command displays the end of a file. By default, it displays the last ten lines, which is useful when we want to quickly check log files, where newer entries are located at the end.
- Example: `tail file.txt`
- The number of lines displayed can be specified using the `-n` option.
- Example: `tail -n 5 file.txt`
- Similarly, we can view the end of multiple files.
- Example: `tail -n 3 first.txt second.txt`
- The command displays the last three lines of each file and indicates which file each output belongs to.

The `tail` Command

- One of the most useful options of `tail` is `-f`, which allows us to monitor changes to a file in real time.
- Example: `tail -f file.txt`
- The command first displays the last ten lines of the file and then waits for new data. When new lines are added to the end of the file, they are also displayed in the terminal.
- This is commonly used to monitor log files while a program is running or while troubleshooting problems.
- Monitoring is usually stopped by pressing `Ctrl + C`.

The `wc` Command

- The **wc** command (word count) counts lines, words, and bytes in a text file. It is useful for quickly obtaining basic information about the size of a file.
- Example: `wc file.txt`
- By default, the output contains three numerical values in the following order: the number of lines, the number of words, and the number of bytes.
- The file name is displayed at the end of the output. The command counts lines based on newline characters and words based on sequences of characters separated by whitespace.

The `wc` Command

- Options allow us to specify which value we want to display.
- The `-l` option displays the number of lines.
- The `-w` option displays the number of words.
- The `-c` option displays the number of bytes.
- The `-m` option displays the number of characters.
- The number of characters may differ from the number of bytes, especially in text containing characters encoded using multiple bytes, such as some accented letters.

The `sort` Command

- The **sort** command sorts the lines of a text file and prints the result to standard output. By default, it sorts lines lexicographically, although the order may also depend on the locale settings of the environment.
- Example: `sort file.txt`
- By default, the `sort` command does not modify the original file; it only prints the sorted result.
- Numeric sorting and sorting in reverse order can be performed using the `-n` (numeric) and `-r` (reverse) options.

The `sort` Command

- If lines contain multiple fields, we can specify the column (or columns) by which we want to sort. The `-k` option defines the sort key. Multiple `-k` options can be specified to define sorting priorities.
- Sorting by the second column, which contains numbers, can be performed as follows.
- Example: `sort -k2,2n file.txt`
- The command sorts the lines by the numerical values in the second column, from smallest to largest. The notation `-k2,2n` specifies that only the second column is used as the sort key and that it is interpreted numerically.

The `uniq` Command

- The **uniq** command identifies consecutive identical lines in the input text and, by default, prints each group only once. It is useful when processing lists, log entries, and other text containing repeated data.
- Example: `uniq file.txt`
- To remove all duplicate lines, we must first bring identical lines together. This can be achieved using the `sort` command.
- The `-c` option displays the number of occurrences of each group of consecutive identical lines.
- Important: `uniq` does not find all duplicates in a file by itself; it only processes consecutive identical lines.

The `cut` Command

- The **cut** command displays selected parts of each line in a text file. It can be used to select individual characters, character positions, or fields when the data is arranged in a structured format.
- The `-c` option specifies the character positions to display.
- Example: `cut -c 1-5 file.txt`
- The command displays the first five characters of each line in the file `names.txt`.
- To display only the third character, use:
- Example: `cut -c 3 file.txt`

The `cut` Command

- More commonly, `cut` is used to select columns or fields separated by a specific character. The `-d` option specifies the delimiter, while the `-f` option specifies the number of the field to display.
- Displaying the first field when the data is separated by commas:
- Example: `cut -d ',' -f 1 file.txt`
- Displaying the second and third fields:
- Example: `cut -d ',' -f 2,3 file.txt` or `cut -d ',' -f 2-3 file.txt`

The `paste` Command

- The **paste** command combines the contents of multiple files by printing their corresponding lines together on the same line. By default, individual fields are separated by a tab character.
- Example: `paste names.txt scores.txt`
- By default, fields are separated by tabs. The `-d` option allows us to specify a different delimiter.
- Example: `paste -d ',' names.txt scores.txt`

The `grep` Command

- The **grep** command searches for **lines** matching specific text or a pattern. It prints only the lines in which a match is found, making it useful for examining log files, configuration files, and other text.
- Example: `grep "error" system.log`
- By default, searches are case-sensitive.
- We can also search for multiple strings at once in one of the following ways:
- Example: `grep -E "string1|string2" file.txt`
- Example: `grep -e "string1" -e "string2" file.txt`
- Example: `egrep "string1|string2" file.txt`

Linux Operators - > and >>

- The > operator redirects a command's output to a file instead of displaying it on the screen.
- Example: `ls -l > file.txt`
- It can be used with any command that writes output to the terminal.
- Redirection is useful when we want to save command results for later inspection, further processing, or use in another program.
- If the destination file does not exist, it is created. If it already exists, its previous contents are usually overwritten.
- To preserve the existing contents and append new output to the end of the file, we use the >> operator.

Linux Operator - |

- The | operator (**pipe** or **pipeline**) allows the output of one command to be passed directly to another command. This enables us to process the results of one command further without first saving them to a file.
- Example: `cat file.txt | sort`
- The command reads the contents of the file using `cat` and passes them to `sort`, which sorts the lines alphabetically.
- Basic rule: the command on the left side of the | operator produces output, while the command on the right receives that output as input.
- We can also use multiple pipeline operators in sequence.
- Example: `cat file.txt | sort | uniq`
- The command reads the file, sorts its lines, and removes consecutive duplicate lines.