# Emacs, Shells, and Linux File Systems

## 1. Overview

In modern software development, users often deal with a triad of interrelated systems: scripting environments (such as shells), Integrated Development Environments (IDEs) like Emacs, and file systems (e.g., Linux file systems). This lecture takes a holistic, realistic approach to learning these tools simultaneously to mirror real-world development environments. Instead of isolating topics, systems are introduced in the overlapping, interdependent way in which they were historically developed and are actually used.

---

## 2. Emacs as an IDE

### History and Purpose

- Initially, the instructor used Vim (and VI), but switched to Emacs due to its programmability.
- Emacs embeds a programming language (Emacs Lisp) which allows developers to build IDE-like tooling and systems efficiently.
- Emacs is programmable, extensible, and scriptable, making it suitable for heavy development work.

### Emacs Features

Emacs is known for: - Deep programmability - Visual frame and window system - Modality through Modes - Internal shell invocation - Editing many kinds of resources (e.g., files, directories, buffers)

### Buffers vs. Files

- **Buffer**: A temporary in-memory representation of data (text, code, etc.)
- **File**: Stored on disk, persistent across system reboots.

Emacs distinguishes between these: - Opening a file loads it into a buffer - Changes are made to the buffer, not directly to the file - Save operation (Ctrl-X Ctrl-S) writes buffer content to the file

Use case:

```
Ctrl-X Ctrl-F hello.txt       # Open or create file in buffer
Editing in buffer             # Does not affect file
Ctrl-X Ctrl-S                 # Save buffer to file
```

### File Persistence in Emacs

Persistence in terms of application design means data survives crashes or reboots. Emacs approaches this with:

1. Buffers (fast, temporary)
2. Auto-save files (e.g., #hello.txt#)
3. Backup symbolic link to track editing (e.g., .#hello.txt)

Problems with persistent variables: - Naive persistent variables (e.g., writing directly to storage each change) can degrade performance due to I/O latency.

Example:

```
persistent int numberOfStudents;
numberOfStudents++;
```

Would require saving to durable storage at every write, which is slow (milliseconds rather than nanoseconds).

File Save Lifecycle in Emacs

| Stage | Description |
| --- | --- |
| Start editing | Buffer is in memory, file untouched |

| | |
|---|---|
| File lock (symbolic link) | Symbolic link to editing session metadata (e.g., .#hello.txt) |
| Manual Save (Ctrl-X Ctrl-S) | Writes buffer to hello.txt |

## Emacs Modes

- Emacs is a “modeful” editor
- A mode is a set of behaviors/rules for how keypresses and commands are interpreted
- Examples:
  - Text Mode
  - Dired Mode (directory editing)

Modes affect behavior: - Typing `d` in a buffer inserts “d” - Typing `d` in Dired marks file for deletion

List of Emacs Commands (selected):

| Command | Meaning |
|---|---|
| Ctrl-X Ctrl-F | Find (open) file |
| Ctrl-X Ctrl-S | Save buffer contents to file |
| Ctrl-X 4 D | Open directory editor in new window |
| Ctrl-X O | Switch window (focus) |
| Ctrl-S | Forward search |
| Ctrl-R | Reverse search |
| Ctrl-@ (Ctrl-Space) | Set mark for region |
| Meta-W | Copy region |
| Ctrl-Y | Paste (Yank) |
| Ctrl-G | Abort current command (interrupt) |
| Ctrl-X 1 | Maximize current window |
| Meta-X shell | Start a shell inside Emacs |

## Windows and Frames

Terminology:

- Window: A viewport into a buffer within a frame.
- Frame: A full Emacs window as the OS sees it.
- Multiple windows can show different buffers side-by-side.

Command Summary:

| Command | Action |
|---|---|
| Ctrl+X 4 D | Open Dired in new window |
| Ctrl+X O | Move between windows |
| Ctrl+X 1 | Focus single window |

## Emacs Editor for Directories (Dired Mode)

Dired is a mode for editing and navigating directories:

- Textual view of directory contents
- Limited editing (e.g., flagging for deletion)
- Drive changes back to the filesystem

Dired Commands

| Key | Action |
|---|---|

| x | Execute marked delete actions |
|---|---|
| u | Unmark |

---

### Shell Access Inside Emacs

- `M-x shell`: Opens a shell inside Emacs.
- Basic commands:
  - `cat file`: Print contents
  - `ls -l`: List files with metadata
  - `Ctrl+D`: End-of-input
  - `Ctrl+C`: Interrupt
  - `Ctrl+G`: Cancel Emacs command

---

### Emacs Save Files and Metadata

Emacs creates temporary metadata files for buffers:

1. `#filename#`: Regular auto-save file.
2. `.filename~`: Backup file (not discussed in detail).
3. `.filename.swp`: May appear in other editors (not Emacs).
4. `.filename` symlink:
   - Contains editor and session metadata:
     - Username
     - Hostname
     - Emacs process ID
     - System boot timestamp
   - Enables Emacs to warn if a file is open elsewhere.

Example:

```
ls -l
#hello.txt#    # Auto-save buffer
.hello.txt     # Symbolic link containing editing info
hello.txt      # Main file
```

---

# 3. Linux File System Concepts

### Files and File Types

In Linux:

- Files are objects storing a finite sequence of bytes
- Associated with metadata (permissions, size, timestamps)
- Common types:

| Symbol | Type |
|---|---|
| - | Regular file |
| d | Directory |
| l | Symbolic link |

### File Metadata (shown with `ls -l`)

Example:

```
-rwxr-xr-- 1 egert egert 1234 Apr 5 10:00 hello.txt
```

Breakdown: | Component | Meaning | |————|————————————-| | -rwxr-xr- | Permissions (user/group/other) | | 1 | Link count | | egert egert | Owner and Group | | 1234 | Size in bytes | | Apr 5 10:00 | Last modified time | | hello.txt | File name |

### File Permissions

- Three groups: user (owner), group, others.
- Each has read (r), write (w), execute (x).

- Use `id` command to check current user and group.
- Each file has an owner and a group field.

| Section | Symbol | Octal | Role |
|---|---|---|---|
| User | rwx | 7 | Owner permissions |
| Group | r-x | 5 | Group permissions |
| Other | r– | 4 | Others' permissions |

Common Permission Formats:

| Symbolic | Octal | Meaning |
|---|---|---|
| rwxr–r– | 744 | Owner can read/write/execute |
| rw-r–r– | 644 | Owner can read/write, others read |
| rwxrwxr-x | 775 | Group shares full rights |

## Special Permission Bits

Linux includes 12 permission bits: - 9 common bits (3x rwx for user/group/others) - 3 special bits:

| Name | Symbol | Effect |
|---|---|---|
| setuid | s | Run file as file's owner |
| setgid | s | Run as file's group |
| sticky | t | Applies to directories — restricts deletion |

Example:

```
-rwsr-xr-- 1 root root 1234 su       # setuid (run as root)
drwxrwxrwt 10 root root 4096 /tmp   # sticky bit (shared dir)
```

## Symbolic and Hard Links

- Symbolic Links:
  - Point to a path
  - Created with: `ln -s target linkname`
  - Can be "dangling" (point to non-existent file)
  - Used by Emacs with metadata to store process ID, editor, and system state in symlink name.
- Hard Links:
  - Share inode (data) and increase link count
  - Created with: `ln file1 file2`
  - Both names refer to the same data

Hard Link Example:

```
touch foo
ln foo bar        # bar is another name for foo
```

Inode Confirmation with `ls -i`:

```
123456 foo
123456 bar       # same inode = same file
```

Use `rm` to remove a name; file persists until all links are removed.

Observation: - Link count > 1 means hard link(s) exist.

## Directories

- Directories are files that map names to inodes

- Directories start with link count = 2:
  - One for itself (`.`)
  - One from parent (`dirname`)
- Each subdirectory adds 1 to parent's link count (from its `..`)
  - `..` from `/` (root) points to itself.
  - Root has link count matching number of subdirectories + 1.

```
mkdir dir
ls -ld dir        # shows link count 2
mkdir dir/subdir
ls -ld dir        # link count now 3
```

### Tree-Structured File System

- Root directory: `/`
- Files organized in hierarchy beneath root
- Paths:
  - Absolute: `/home/user/file.txt`
  - Relative: `./file.txt` or `../file.txt`

Diagram:

```
/
├── home/
│   └── egert/
│       ├── foo
│       └── bar
├── usr/
│   └── bin/
└── tmp/
```

### Inode Numbers

- Every file has a unique inode number
- Use `ls -i` to view
- Link count: Number of names pointing to an inode

---

## 4. Shell Commands and Utilities

These utilities interact with the file system and processes.

### Basic Shell Overview

- Default shells provide process management, file commands, and scripting features.
- Setting: Emacs can embed shell as buffer (`M-x shell`)

### Control Characters

| Key | Meaning |
|---|---|
| Ctrl+D | End-of-file/input |
| Ctrl+C | Interrupt (stop process) |
| Ctrl+G | Emacs interrupt (abort command) |
| Ctrl+@ / Ctrl+Space | Set mark for copy |
| Meta+W | Copy region (kill-ring) |
| Ctrl+Y | Paste ("yank") |

---

### cat

- `cat file`: Outputs content of a file.
- `cat`: With no args, reads from keyboard and echoes input.

```
cat file1             # Print content
cat file1 file2       # Concatenate and print
```

Special behavior: - `cat` with no arguments reads from terminal (stdin) - Ctrl+D at start of line signals EOF

### ps

- Shows process status
- `ps -ef`: Lists all processes with full info

Sample Output:

```
UID   PID   PPID  CMD
egert 22588 1     emacs
```

Process ID (PID) used in Emacs symbolic links:

```
.#hello.txt -> egert@machine.22588:timestamp
```

### chmod

- Changes file permissions
- Syntax:

  ```
  chmod 644 file.txt
  chmod +x script.sh
  ```

### ln

Creates links: - `ln file1 file2`: Hard link - `ln -s path link`: Symbolic link

Special cases: - Cannot hard-link directories (except by filesystem convention for `.` and `..`) - Emacs uses symbolic links to indicate lock/status metadata for edited files

### touch, rm, and truncate

- `touch`: Creates empty file or updates timestamp
- `rm`: Removes file name (not file unless final link)
- `truncate`: Sets file size without writing content

Example:

```
truncate -s 1T bigfile   # Create 1 terabyte sparse file
ls -lh bigfile           # Shows size
cat bigfile              # Outputs null bytes (if not crashed)
```

---

## 5. Summary

This lecture introduced three core interconnected systems in a Unix/Linux environment: scripting via shell, file systems, and editing environments like Emacs. Emphasis was placed on realism through simultaneous learning and usage reflective of real-world software engineering scenarios. Emacs was covered in depth with its concept of buffers, windows, modes, saving mechanics, and integration with directories and the shell. File system fundamentals such as inode numbers, link counts, file permissions, symbolic vs hard links, and hierarchical structure were explored. Command-line utilities like `cat`, `ls`, `chmod`, `ps`, and `ln` helped demonstrate these principles. The engineer's need to balance persistence, performance, and understandability in system architecture was repeatedly highlighted.