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.

> **Realism in Learning:** The lecture intentionally blends topics, reflecting how technologies were co-developed and are used together in practice. This mirrors the real-world experience of being "thrown into" multiple new systems at once, as in industry.

---

2. Emacs as an IDE

History and Purpose

- The instructor originally used Vim/VI, but switched to Emacs for its programmability and extensibility.
- Emacs embeds a programming language (Emacs Lisp), enabling rapid development of IDE-like tools and customizations.
- Emacs is programmable, extensible, and scriptable, making it suitable for heavy development work and building custom environments.

Emacs Features

Emacs is known for:

- Deep programmability
- Visual frame and window system
- Modality through Modes ("modeful" editor)
- Internal shell invocation
- Editing many kinds of resources (files, directories, buffers)

Buffers vs. Files

- **Buffer:** A temporary, in-memory representation of data (text, code, etc.). Fast, but not persistent.
- **File:** Stored on disk, persistent across system reboots.

Key distinction:

- 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
```

Buffer Management

- Emacs can have many buffers open at once (files, directories, shell sessions, etc.).
- Switch buffers: `Ctrl-X b` (prompt for buffer name), `Ctrl-X Ctrl-B` (list all buffers).
- Kill buffer: `Ctrl-X k` (prompt for buffer to close).

ASCII Diagram: Buffers and Files

```
[File on Disk] <---save/load---> [Buffer in RAM] <---view---> [Window in Frame]
```

The Problem of Persistence

- Applications should survive power outages (persistence), be fast, and be understandable.
- Naive persistent variables (e.g., writing to storage on every change) are too slow due to I/O latency (milliseconds vs. nanoseconds).

Example:

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

This would require saving to durable storage at every write, which is slow.

Emacs' Approach to Persistence

Emacs balances speed and persistence by:

1. Keeping data in buffers (fast, in-memory)
2. Periodically auto-saving to temp files (e.g., `#hello.txt#`)
3. Creating symbolic links for lock/metadata (e.g., `.hello.txt`)
4. Only writing to the actual file on explicit save (`Ctrl-X Ctrl-S`)

Auto-save, Backup, and Lock Files:

- `#filename#`: Auto-save file, periodically updated (for crash recovery).
- `filename~`: Backup file, created on first save (previous version, for undoing mistakes).
- `.filename`: Symbolic link containing metadata (user, host, Emacs PID, system boot time) for lock detection.

Crash Recovery Example:
If Emacs crashes, on restart it will detect `#filename#` and offer to recover unsaved changes. If a lock file (`.filename`) exists but no Emacs process is editing, it warns of a stale lock.

Table: Emacs Save/Backup/Lock Files

File Type	Example	Purpose
Auto-save	`#foo.txt#`	Crash recovery (periodic save)
Backup	`foo.txt~`	Previous version (undo)
Lock (symlink)	`.foo.txt`	Prevent concurrent edits

Emacs Modes

- **Major Modes:** Define the main editing mode for a buffer (e.g., Text, Python, Dired, Shell). Only one major mode per buffer.
- **Minor Modes:** Add optional features (e.g., line numbers, auto-fill, flyspell). Multiple minor modes can be active.
- Mode line (bottom of window) shows active modes.

Table: Major vs. Minor Modes

Type	Example	Description
Major Mode	`python-mode`	Main editing mode (syntax, keys)
Minor Mode	`flyspell-mode`	Optional feature (spellcheck)

Windows and Frames

- **Window:** A viewport into a buffer within a frame (Emacs terminology).
- **Frame:** What most OSes call a "window"; can contain multiple Emacs windows.
- Multiple windows can show different buffers side-by-side.

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

Dired Mode (Directory Editor)

- Dired is a mode for editing and navigating directories.
- Shows a textual view of directory contents.
- Limited editing (e.g., flagging for deletion, renaming).
- Changes are driven back to the filesystem.

Key	Action
d	Mark file for deletion

Key	Action
x	Execute marked delete actions
u	Unmark

- Advanced features:
  - Batch rename: `R` (rename multiple files)
  - Mark files by regex: `% m` (mark by pattern)
  - Run shell commands on files: `!` (run command on marked files)
  - Copy (`C`), move (`R`), create symlinks (`S`), compress (`Z`), etc.

Example: Batch Rename

```
Mark files with `m` or `% m`, then press `R` to rename all at once.
```

Shell Access Inside Emacs

- `M-x shell`: Opens a shell inside Emacs.
- Multiple shells: `M-x shell` creates a new shell buffer each time.
- Shell buffers can be renamed for organization (`M-x rename-buffer`).

Emacs Lisp Example

Emacs is programmable via Emacs Lisp. Example: define a command to insert the current date.

```
(defun insert-date ()
  "Insert the current date at point."
  (interactive)
  (insert (format-time-string "%Y-%m-%d")))
;; Bind to key: Ctrl-c d
(global-set-key (kbd "C-c d") 'insert-date)
```

Emacs Save Files and Metadata

- Emacs creates temporary metadata files for buffers:
  1. `#filename#`: Regular auto-save file (periodically updated)
  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 PID, system boot timestamp)
- Symbolic link is used as a lock and to warn about concurrent edits (dangling symlink if no process is editing).
- If Emacs crashes, these files help recover work and warn about stale locks.

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

- 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

Symbol	Type
b	Block device
c	Char device
p	FIFO/pipe
s	Socket

File Metadata (shown with `ls -l`)

Example:

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

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

- 9-bit mask: 3 groups (user/owner, group, others), each with 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)
```

- The sticky bit (t) on directories means only the file's owner can delete/rename files within.
- setuid (s) means the program runs as the file's owner (e.g., `su` runs as root).

- setgid (s) means the program runs as the file's group.

Changing Permissions

- `chmod` changes file permissions.
- Syntax:

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

- Only the file's owner (or root) can change permissions.

Symbolic and Hard Links

- **Symbolic Links (symlinks):**
  - Point to a path (text string)
  - Created with: `ln -s target linkname`
  - Can be "dangling" (point to non-existent file)
  - Used by Emacs for lock/metadata files
  - Symlinks are like pointers (textual), followed by splicing the path
- **Hard Links:**
  - Share inode (data) and increase link count
  - Created with: `ln file1 file2`
  - Both names refer to the same data (same inode)
  - Cannot hard-link directories (except for `.` and `..` by convention)

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

Table: Hard vs. Symbolic Links

Feature	Hard Link	Symbolic Link
Points to	Inode (data)	Path (name)
Link count	Increases	No effect
Cross-filesystem	No	Yes
Can link dirs	No (except . ..)	Yes (with caution)
Survives target	Yes	No (dangling link)
`ls -l` shows	Same inode	Arrow (`->`)

File Deletion with Hard Links

- Deleting a file name with `rm` only removes that name.
- File data is deleted **only** when all hard links are removed (link count = 0).

Example:

```
touch foo
ln foo bar
rm foo      # bar still exists, data not deleted
rm bar      # now data is deleted
```

Directories

- Directories are files that map names to inodes.
- Always contain `.` (self) and `..` (parent).
- 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
```

ASCII Diagram: Directory Link Count Changes

```
mkdir dir
ls -ld dir   # link count = 2 (dir itself + parent)

mkdir dir/subdir
ls -ld dir   # link count = 3 (dir itself + parent + subdir)

# General rule:
# link count = 2 + number of subdirectories
```

Tree-Structured File System

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

ASCII 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.
- Inode numbers are like addresses for files, but you cannot access a file by inode alone—names are required.

---

4. Shell Commands and Utilities

These utilities interact with the file system and processes.

Basic Shell Overview

- Shells provide process management, file commands, and scripting features.
- Emacs can embed a shell as a buffer (`M-x shell`).
- Shells can run scripts: a file with commands, executed with `sh script.sh` or by making it executable (`chmod +x script.sh`).
- Scripts can use variables, loops, and conditionals.

Example: Simple Shell Script

```
#!/bin/bash
for file in *.txt; do
  echo "Processing $file"
done
```

Control Characters

Key	Meaning
Ctrl+D	End-of-file/input (EOF)
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")

Redirection and Pipes

- `>`: Redirect output to file (`ls > out.txt`)
- `>>`: Append output to file (`echo hi >> out.txt`)
- `<`: Read input from file (`cat < in.txt`)
- `|`: Pipe output to another command (`ls | grep foo`)

Example: Combine Commands

```
cat file.txt | grep hello | wc -l > count.txt
```

Process Management

- Every process has a parent (except `init`/`systemd`).
- `ps -ef` shows parent PID (PPID).
- `kill <PID>` sends a signal to a process (default: SIGTERM).
- `kill -9 <PID>` forcefully kills (SIGKILL).
- `jobs`, `bg`, `fg` manage background/foreground jobs in shell.

ASCII Diagram: Process Tree

```
1 (init)
 |
 +-- 22588 (emacs)
      |
      +-- 22601 (bash)
            |
            +-- 22610 (cat)
```

cat

- `cat file`: Outputs content of a file.
- `cat`: With no args, reads from keyboard and echoes input.
- Ctrl+D at start of line signals EOF.
- `cat file1 file2`: Concatenates and prints both files.

ps

- Shows process status
- `ps -ef`: Lists all processes with full info
- Process ID (PID) is used in Emacs symbolic links for lock/metadata

Sample Output:

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

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
- Cannot hard-link directories (except for `.` and `..` by convention)
- 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)
```

- Modern filesystems may use sparse files, compression, or encryption, so file size may not reflect actual disk usage.

---

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.

File Persistence in Emacs

ASCII Flowchart: Emacs File Save Lifecycle

```
[Buffer in RAM]
     |
     v
[Auto-save file (#filename#)]
     |
     v
[Lock file (.filename symlink)]
     |
     v
[Manual Save (Ctrl-X Ctrl-S)]
     |
     v
[File on Disk]
```

Windows and Frames

ASCII Diagram: Emacs Frame, Windows, and Buffers

```
+------------------------------+   Frame (OS window)
| +-----------+ +-----------+ |
```

```
| |  Window   | |  Window   | |
| |  (view 1) | |  (view 2) | |
| +-----------+ +-----------+ |
+-----------------------------+
      |              |
      v              v
   Buffer A       Buffer B
```

Directories

ASCII Diagram: Directory Link Count Changes

```
mkdir dir
ls -ld dir   # link count = 2 (dir itself + parent)

mkdir dir/subdir
ls -ld dir   # link count = 3 (dir itself + parent + subdir)

# General rule:
# link count = 2 + number of subdirectories
```

---

6. Additional Resources

- Emacs Documentation
- Linux File System Basics
- Shell Scripting Tutorial

---

7. Conclusion

This lecture provided a comprehensive overview of Emacs, Linux file systems, and shell scripting. It highlighted the importance of understanding file systems and shell commands in a Unix/Linux environment. The engineer's need to balance persistence, performance, and understandability in system architecture was emphasized throughout the lecture.