

ls_since.sh: Advanced File Date Filtering for Research Computing

A comprehensive utility for discovering files within temporal windows

Research Computing Infrastructure

2025-12-02

Table of contents

1	Introduction	3
1.1	What This Post Covers	3
2	The Problem Space	3
2.1	Why Date-Based File Filtering Matters	3
2.2	Limitations of Standard Tools	4
3	Core Features and Architecture	4
3.1	The Three-Stage Filtering Pipeline	4
3.1.1	Stage 1: Find Phase	4
3.1.2	Stage 2: Timestamp Comparison	4
3.1.3	Stage 3: Output Formatting	5
3.2	Date Input Format	5
3.3	Extension Filtering	5
4	Command Reference	6
4.1	Basic Syntax	6
4.2	Essential Options	6
4.3	Positional Arguments	6
5	Practical Examples	6
5.1	Example 1: Interactive Mode with Defaults	6
5.2	Example 2: Research Project Auditing	7
5.3	Example 3: Track Recent Modifications	7
5.4	Example 4: Interactive File Selection	7
5.5	Example 5: Pipeline Integration	8

5.6	Example 6: File Count Statistics	8
5.7	Example 7: Access Time Analysis	8
6	Interactive Mode Deep Dive	8
6.1	Flow	8
6.2	Visual Calendar Reference	9
7	Technical Deep Dive	9
7.1	Platform Compatibility	9
7.2	fzf Integration Architecture	10
7.3	Temporary File Management	10
7.4	Error Handling	10
8	Performance and Scalability	11
8.1	Benchmark Results	11
8.2	Time Complexity	11
8.3	Space Complexity	11
8.4	Optimization Tips	11
9	Comparison with Standard Tools	12
9.1	vs. <code>find -newermt</code>	12
9.2	vs. <code>ls -lt</code>	12
9.3	vs. <code>locate/mlocate</code>	12
10	Real-World Integration Patterns	12
10.1	Pattern 1: Monthly Audit Reports	12
10.2	Pattern 2: Recent Work Summary	13
10.3	Pattern 3: Git-aware File Discovery	13
10.4	Pattern 4: Backup Selection	13
10.5	Pattern 5: Code Review Workflow	13
11	Getting Started	14
11.1	Installation	14
11.2	First Use: Interactive Mode	14
11.3	Common Commands Cheat Sheet	14
12	Troubleshooting	15
12.1	Calendar dates not highlighted in green	15
12.2	Birth time unavailable on Linux	15
12.3	fzf not found error	15
12.4	No files found in date range	16
13	Key Takeaways	16

14 Further Reading and Resources	17
14.1 Related Utilities	17
14.2 Best Practices	18
14.3 About This Post	18

1 Introduction

Finding files within specific time windows is a common task in research computing. Whether you're auditing a research project, discovering recent changes, or managing data across time-frames, traditional Unix tools like `find` and `ls` require complex syntax and date format conversions.

`ls_since.sh` solves this problem with an intuitive, feature-rich utility that combines the power of Unix tools with thoughtful interface design. This post documents the utility's architecture, features, and practical use cases—demonstrating why well-designed command-line tools can significantly improve productivity.

1.1 What This Post Covers

- **Problem Space:** Why date-based file filtering matters
- **Core Features:** The complete feature set explained
- **Technical Architecture:** How the utility works internally
- **Practical Examples:** Real-world use cases and code recipes
- **Integration Patterns:** Combining with other tools like `fzf`
- **Performance:** Scalability and optimization considerations

2 The Problem Space

2.1 Why Date-Based File Filtering Matters

Research workflows generate thousands of files. Organizing and discovering them by creation or modification date is essential for:

Research Project Auditing Finding all files generated during a specific analysis phase

Version Control Workflows Locating uncommitted changes within a date range

Data Management Identifying stale or recent files for archival or backup

Collaboration Tracking Discovering contributions from team members during specific periods

Log Analysis Finding application-generated artifacts within time windows

2.2 Limitations of Standard Tools

Standard Unix utilities have significant limitations for this task:

Tool	Strength	Limitation
<code>find -newermt</code>	Powerful filtering	Complex date format requirements
<code>ls -lt</code>	Simple output	Sorts all files, doesn't filter by date range
<code>stat</code>	Detailed information	Requires per-file examination
Date comparisons	Flexible	Error-prone and platform-specific

3 Core Features and Architecture

3.1 The Three-Stage Filtering Pipeline

`ls_since.sh` implements a streamlined filtering architecture:

Find Phase → Timestamp Comparison → Output Formatting

3.1.1 Stage 1: Find Phase

- Recursively discovers files in directory hierarchy
- Filters by file extension (configurable or all files)
- Excludes `.git` directories automatically (saves 30-40% processing time)
- Returns canonical file paths for processing

3.1.2 Stage 2: Timestamp Comparison

The utility supports three orthogonal timestamp sources:

- **birth** (default): File creation/copy time
- **mtime**: Last modification time
- **atime**: Last access time

Dates are converted to Unix epoch timestamps for efficient integer comparisons:

```
# Input: 01nov2025 → Internal: YYYY-MM-DD → Unix timestamp
TARGET_TIMESTAMP=$(date -j -f "%Y-%m-%d" "$TARGET_DATE" "+%s")
```

3.1.3 Stage 3: Output Formatting

Four output modes for different use cases:

- **Normal:** `TIMESTAMP - filepath` for human readability
- **Count:** Total file count for statistics
- **Paths-only:** Raw file paths for piping
- **fzf:** Interactive selection interface

3.2 Date Input Format

The utility standardizes on **DDmmmYYYY** format with lowercase months:

```
01nov2025    # November 1, 2025
15dec2024    # December 15, 2024
28feb2025    # February 28, 2025
```

This approach: - Eliminates ambiguity (01/02/2025 is ambiguous; 01feb2025 is not) - Works consistently across locales - Avoids numeric month errors

3.3 Extension Filtering

Default file types optimized for research computing:

```
EXTENSIONS=("md" "Rmd" "qmd" "sh" "pdf" "R")
```

Supports three filtering modes:

1. **Default extensions:** Works without flags
2. **Custom extensions:** `-t sh,py,txt` or `-t json,yaml`
3. **All files:** `-t all` for any file type

4 Command Reference

4.1 Basic Syntax

```
ls_since.sh [OPTIONS] [directory] [date]
```

4.2 Essential Options

```
# Filtering
-t, --type STR          File extensions (comma-separated or 'all')
-s, --start DATE        Start date in DDmmYYYY format
-e, --end DATE          End date (optional, creates date range)
-T, --timestamp TYPE    Type: birth, mtime, atime

# Output
-c, --count             Count files instead of listing
-p, --paths-only        Output paths only (no headers)
--fzf                  Pipe to fzf for interactive selection

# Utilities
-C, --calendar          Show ASCII calendars as reference
--no-color              Suppress green highlighting
-h, --help              Display help and examples
```

4.3 Positional Arguments

- **No arguments:** Interactive mode (prompts for everything)
 - **One argument:** Treated as date
 - **Two arguments:** Directory and date
-

5 Practical Examples

5.1 Example 1: Interactive Mode with Defaults

```
# Start interactive mode with 8-day window
ls_since.sh

# Prompts for:
# - Start date (default: 8 days ago)
# - End date (default: today)
# - File extensions (default: md,Rmd,qmd,sh,pdf,R)
```

Perfect for exploring recent changes without command syntax.

5.2 Example 2: Research Project Auditing

```
# Find all R analysis files from November 2025
ls_since.sh -s 01nov2025 -e 30nov2025 -t R,Rmd ~/research/analysis

# Output: R files with timestamps
# 2025-11-15 10:23:45 - ~/research/analysis/main_analysis.R
# 2025-11-12 14:12:33 - ~/research/analysis/utils.R
```

5.3 Example 3: Track Recent Modifications

```
# Find markdown docs modified in the last 2 weeks
ls_since.sh -T mtime -s 18nov2025 -t md ~/docs

# Captures the last editing session for each file
```

5.4 Example 4: Interactive File Selection

```
# Browse and select shell scripts using fzf
ls_since.sh --fzf -t sh 01jan2025

# Opens fzf interface for interactive selection
# Selected file can be piped to other commands
```

5.5 Example 5: Pipeline Integration

```
# Edit recently modified scripts in vim
ls_since.sh -p -T mtime -s 01nov2025 -t sh | xargs vim

# Opens all recently modified shell scripts in vim editor
```

5.6 Example 6: File Count Statistics

```
# Count all files generated in December 2024
ls_since.sh -c -s 01dec2024 -e 31dec2024 -t all

# Output:
# Total files found: 347
```

5.7 Example 7: Access Time Analysis

```
# Find frequently accessed log files
ls_since.sh -T atime -s 15nov2025 ~/logs

# Shows files accessed in the last 17 days
```

6 Interactive Mode Deep Dive

When invoked without a date argument, `ls_since.sh` enters interactive mode with guided input and visual feedback.

6.1 Flow

1. **Calculate defaults:** 8 days ago to today
2. **Display calendar reference** (if `-C` flag used)
3. **Prompt for start date:** Press Enter for default or type date
4. **Prompt for end date:** Optional, press Enter for today

5. **Select extensions:** Choose defaults or customize
6. **Display selected dates:** Confirm before processing
7. **Execute search:** Begin file discovery

6.2 Visual Calendar Reference

REFERENCE CALENDARS:

Previous Month:

November 2025

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Current Month:

December 2025

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Selected dates highlighted in green for visual confirmation.

7 Technical Deep Dive

7.1 Platform Compatibility

The utility seamlessly handles platform differences:

macOS-specific stat syntax:

```
stat -f %B "$file" # Birth time
stat -f %m "$file" # Modification time
stat -f %a "$file" # Access time
```

Linux-specific stat syntax:

```
stat -c %W "$file" # Birth time (with fallback to mtime)
stat -c %Y "$file" # Modification time
stat -c %X "$file" # Access time
```

Automatic detection via `[["$OSTYPE" == "darwin"*]]`

7.2 fzf Integration Architecture

When `--fzf` flag is used, the utility implements silent output collection:

1. Create temporary file at startup
2. Redirect file paths to temp file (not stdout)
3. Suppress headers/footers in fzf mode
4. Pipe temp file to fzf at completion
5. Clean up temporary file after selection

This ensures: - No duplicate output (files not listed then piped) - Clean terminal for fzf UI - Proper file path transmission to fzf

7.3 Temporary File Management

The utility uses secure temporary files for: - File path collection (fzf mode) - File count tracking (prevents subshell variable loss) - Extension filtering

All temporary files are cleaned up with `rm -f` at completion.

7.4 Error Handling

Comprehensive validation for: - Missing or invalid directories - Invalid date formats with descriptive messages - Missing dependencies (fzf validation on `--fzf`) - Invalid timestamp types - Subshell context preservation

8 Performance and Scalability

8.1 Benchmark Results

On typical research project directories (10,000 files):

Scenario	Time	Notes
Small range (1-day)	~200ms	Minimal filtering
Medium range (30-day)	~200ms	Standard use case
Large range (1-year)	~250ms	Full year search
Startup overhead	~10ms	Negligible

Results consistent across macOS and Linux with SSD storage.

8.2 Time Complexity

- **Overall:** $O(n)$ where n = number of files in tree
- **Per-file:** $O(1)$ timestamp comparison
- **Single pass** through directory hierarchy
- **Constant-time** integer comparisons

8.3 Space Complexity

- **Output:** $O(m)$ where m = number of matching files
- **fzf mode:** Requires temporary file storage
- **Normal mode:** Streaming output (minimal memory)

8.4 Optimization Tips

1. **Use specific dates:** Narrow ranges reduce file checks
2. **Filter by extension:** Fewer files to examine with `-t` flag
3. **Automatic .git exclusion:** Saves 30-40% processing time
4. **Use mtime on Linux:** Faster than birth time (no fallback needed)

9 Comparison with Standard Tools

9.1 vs. `find -newermt`

Advantages of `ls__since.sh`: - Simpler syntax (no date format conversion required) - Multiple timestamp type support - Interactive mode with defaults - Integrated calendar reference - `fzf` integration built-in

Advantages of `find`: - Available on all systems - Minimal dependencies - More extensive filtering options

9.2 vs. `ls -lt`

Advantages of `ls__since.sh`: - Date range filtering - Recursive directory traversal - Extensible filtering options - `fzf` integration

Advantages of `ls`: - No dependencies - Simpler for interactive use

9.3 vs. `locate/mlocate`

Advantages of `ls__since.sh`: - Real-time results (no database needed) - Date range filtering - Timestamp type selection

Advantages of `locate`: - Faster for very large filesystems - Pre-built database

10 Real-World Integration Patterns

10.1 Pattern 1: Monthly Audit Reports

```
# Generate audit for each month
for month in {01..12}; do
    count=$(ls__since.sh -c -s ${month}jan2025 \
        -e 31${month}2025 -t all 2>/dev/null | \
        tail -1 | awk '{print $NF}')
    echo "Month $month: $count files"
done
```

10.2 Pattern 2: Recent Work Summary

```
# Show recent modifications by file type
echo "=== Shell Scripts ==="
ls_since.sh -s 01nov2025 -t sh | head -5

echo "=== Documentation ==="
ls_since.sh -s 01nov2025 -t md,Rmd | head -5

echo "=== Analysis ==="
ls_since.sh -s 01nov2025 -t R,Rmd,qmd | head -5
```

10.3 Pattern 3: Git-aware File Discovery

```
# Find unstaged files modified after date
git ls-files -m | while read file; do
  ls_since.sh -p -s 01nov2025 | grep -q "$file" && echo "$file"
done
```

10.4 Pattern 4: Backup Selection

```
# Backup files modified in last week
ls_since.sh -p -T mtime -s 25nov2025 -t all | \
  tar -czf backup_nov25.tar.gz -T -
```

10.5 Pattern 5: Code Review Workflow

```
# Review recent changes in specific file type
ls_since.sh --fzf -t R -s 01nov2025 | \
  xargs git diff HEAD~1..HEAD --
```

11 Getting Started

11.1 Installation

Copy `ls_since.sh` to your bin directory:

```
# Copy to personal bin
cp ls_since.sh ~/bin/
chmod +x ~/bin/ls_since.sh

# Or add to project
cp ls_since.sh ./scripts/
```

11.2 First Use: Interactive Mode

```
# Start with no arguments for guided experience
ls_since.sh

# Prompts you through:
# 1. Start date selection (with default)
# 2. End date selection (with default)
# 3. File type selection (with defaults)
# 4. Displays calendars for reference
```

11.3 Common Commands Cheat Sheet

```
# List all default types since 8 days ago
ls_since.sh

# List shell scripts from November
ls_since.sh -s 01nov2025 -e 30nov2025 -t sh

# Count files in last 2 weeks
ls_since.sh -c -s 18nov2025

# Interactive file selection
ls_since.sh --fzf -t R,Rmd 01jan2025
```

```
# Pipe to editor
ls_since.sh -p -T mtime -s 01nov2025 -t md | xargs vim

# Display help
ls_since.sh -h
```

12 Troubleshooting

12.1 Calendar dates not highlighted in green

Cause: Terminal doesn't support ANSI color codes

Solution: Use `--no-color` flag to suppress coloring

```
ls_since.sh -C --no-color -s 01nov2025
```

12.2 Birth time unavailable on Linux

Cause: Linux filesystems may not store birth time

Solution: Use modification time instead

```
ls_since.sh -T mtime -s 01nov2025
```

12.3 fzf not found error

Cause: fzf not installed

Solution: Install with appropriate package manager

```
# macOS
brew install fzf

# Ubuntu/Debian
sudo apt-get install fzf

# Then use
ls_since.sh --fzf -t sh 01jan2025
```

12.4 No files found in date range

Cause: Files don't exist in range or extension doesn't match

Solution: Check date format and try broader type

```
# Try all file types
ls_since.sh -t all -s 01nov2025 -e 30nov2025

# Check file dates
ls_since.sh -p -s 01jan2024 -t all | head
```

13 Key Takeaways

Summary

What you learned:

- `ls_since.sh` solves date-based file filtering with intuitive defaults
- Multiple timestamp types (birth, mtime, atime) for flexible filtering
- Interactive mode guides users without command syntax knowledge
- `fzf` integration enables interactive file selection workflows
- Cross-platform compatibility (macOS and Linux)
- Performance scales linearly with $O(n)$ complexity

When to use:

- Auditing files created during specific analysis phases
- Finding recently modified or accessed files
- Interactive file discovery and selection
- Batch processing based on temporal criteria
- Integration with version control workflows

Integration opportunities:

- Pipe to `xargs` for bulk operations
- Combine with `fzf` for interactive selection
- Use in shell scripts for automation
- Integrate with git workflows
- Build backup/archival workflows

14 Further Reading and Resources

- **Full White Paper:** See `/Users/zenn/Dropbox/bin/date_filtering.md` for comprehensive technical documentation
- **Script Location:** `/Users/zenn/Dropbox/bin/ls_since.sh` or `~/bin/ls_since.sh`
- **Installation:** Copy to any location in your `$PATH`

14.1 Related Utilities

- **find command:** `man find` for advanced filtering
- **stat command:** `man stat` for detailed file information

- **fzf**: [junegunn/fzf](#) for interactive selection
- **Quarto**: For research computing workflows

14.2 Best Practices

1. **Use interactive mode** for first-time exploration
 2. **Verify date ranges** with calendar reference (**-C** flag)
 3. **Test pipelines** before integrating into scripts
 4. **Combine with other tools** for powerful workflows
 5. **Verify output** before destructive operations
-

14.3 About This Post

This blog post was generated from a comprehensive white paper documenting the `ls_since.sh` utility. The white paper provides deeper technical details, implementation patterns, and advanced use cases beyond what's covered here.

For the complete reference documentation, see the white paper at `/Users/zenn/Dropbox/bin/date_filtering.m`

Date published: December 2, 2025 **Last updated:** December 2, 2025