File Processing Integration - Implementation Summary

✅ Completed Tasks

All requirements have been successfully implemented and tested.

1. Inline Mode File Processing ✓

File: `cmd/root.go`

- Already implemented and working
- Processes files using `fileprocessor.ProcessFiles()`
- Separates images (sent to vision models) from text content
- Provides visual feedback during processing

Usage:

```
termai -file document.pdf "Summarize this"
termai -file image.png -file code.go "Analyze these files"
```

2. Chat Mode File Support ✓

File: `cmd/chat.go`

Added Features:
- `-file` flag for attaching files when starting chat
- `/attach` command for runtime file attachment
- `/files` command to list attached files
- `/clear-files` command to clear attachments
- UI display of attached files above input area
- Automatic clearing of attachments after sending (configurable)

Usage:

```
# Start with files
termai chat -file document.pdf -file image.png

# During chat
/attach code.go readme.md
/files
/clear-files
```

3. Directory Context Feature ✓

File: `cmd/chat.go`

Implementation:
- `--dir/-d` flag to load directory as context
- Scans only top-level files (security/safety)
- Filters for supported file types

- `scanDirectory()` function for directory processing
- Context info displayed in header

Chat Commands:
- `/context` - Show context files
- `/context-add <file>` - Add files to context
- `/context-remove <file>` - Remove from context

Usage:

```
# Load directory
termai chat --dir ./project
termai chat -d ./src

# During chat
/context
/context-add newfile.txt
/context-remove oldfile.txt
```

4. UI Enhancements ✓

Header Updates:
- Shows directory context: `📁 src (12 files)`
- Context badge with file count
- Blue background color for context info

Input Area:
- Shows attached files with icons:
- 📷 Images
- 📄 PDFs
- 💻 Code files
- 📝 Text files
- Yellow/gold styling for attachment info

Example Display:

```
✨ TermAI | abacus | gpt-4 | 📁 src (12 files) | ● Ready

📎 Attached: 📷 screenshot.png, 📄 report.pdf, 💻 code.go

You ▸ Your message...
```

5. Configuration Options ✓

File: `internal/config/config.go`

Added `FileConfig` struct:

```go
type FileConfig struct {
    MaxFileSize              int64 // Maximum file size (default: 10MB)
    AutoClearAfterSend       bool  // Clear attached files after sending
    IncludeContextInEveryMsg bool  // Include context in every message
}
```

Default Configuration:

```yaml
files:
  max_file_size: 10485760          # 10MB
  auto_clear_after_send: true      # Clear attachments after send
  include_context_in_every_msg: false  # Context only in first message
```

6. Documentation ✓

Created:
- `FILE_ATTACHMENTS.md` - Comprehensive user guide
- Updated `README.md` with file attachment features
- Added examples and best practices

README Updates:
- Added file attachment features to feature list
- Added usage examples for inline and chat modes
- Added chat commands documentation
- Added file configuration section

Technical Implementation Details

File Processing Flow

1. **Validation**: Check file existence, size, and type
2. **Processing**:
   - Images → Base64 encoding with MIME type
   - PDFs → Text extraction using `github.com/ledongthuc/pdf`
   - Text/Code → Direct read
3. **Message Construction**:
   - Images → `Message.Images[]` array
   - Text content → Appended to `Message.Content`
4. **Provider Integration**: Files included in API requests

Chat Model Structure

```go
type chatModel struct {
    // ... existing fields
    attachedFiles  []*fileprocessor.FileAttachment  // User attachments
    contextFiles   []*fileprocessor.FileAttachment  // Directory context
    contextDirPath string                           // Context directory path
}
```

Command Updates

Updated `chatCommands` array for auto-completion:

```go
chatCommands = []string{
    "/help", "/exit", "/quit", "/clear", "/profile",
    "/attach", "/files", "/clear-files",
    "/context", "/context-add", "/context-remove"
}
```

Message Sending Logic

```
func (m chatModel) streamResponse() tea.Cmd {
    // Combine attached and context files
    allFiles := append(m.attachedFiles, m.contextFiles...)

    // Separate images from text
    for _, file := range allFiles {
        if file.Type == "image" {
            images = append(images, file.Content)
        } else {
            // Append text content with delimiters
            textContent +=
fmt.Sprintf("\n\n--- Content from %s ---\n%s\n--- End of %s ---",
                file.Name, file.Content, file.Name)
        }
    }

    // Update message with files
    lastMsg.Content = textContent
    lastMsg.Images = images
}
```

Supported File Types

Images (Vision Models)

- `.jpg`, `.jpeg`, `.png`, `.gif`, `.webp`
- Encoded as base64 data URLs
- Sent to vision-capable models

Documents

- `.pdf` - Text extraction

Text Files

- `.txt`, `.md`, `.markdown`

Code Files

- `.go`, `.py`, `.js`, `.ts`, `.tsx`, `.jsx`
- `.java`, `.c`, `.cpp`, `.h`, `.rs`, `.rb`
- `.php`, `.sh`, `.bash`
- `.yaml`, `.yml`, `.json`, `.xml`
- `.html`, `.css`, `.sql`

Security & Safety Features

1. **Directory Scanning**: Limited to top-level files only
2. **File Size Limits**: Configurable maximum (default 10MB)
3. **File Type Validation**: Only supported types processed
4. **Path Validation**: Checks file existence and accessibility
5. **Error Handling**: Graceful failures with clear messages

Testing

Build Status

✅ Successfully compiled: `go build -o termai`

Test Files Created

- `test_files/test1.txt` - Sample text file
- `test_files/test2.md` - Sample markdown file
- `test_files/code.go` - Sample code file

Manual Testing Checklist

- ✅ Inline mode with -file flag works
- ✅ Chat mode with -file flag on start
- ✅ /attach command in chat
- ✅ /files and /clear-files commands
- ✅ –dir flag for directory context
- ✅ /context commands work
- ✅ UI shows attached files correctly
- ✅ Header shows context information
- ✅ Files are included in API requests
- ✅ Configuration options work

Git Commit

Commit ID: `428a4f6`

Commit Message:

```
feat: Add comprehensive file attachment and directory context features

- Add file attachment support in inline mode (-file flag)
- Add file attachment support in chat mode (-file flag, /attach command)
- Implement directory context feature (--dir/-d flag)
- Add chat commands: /attach, /files, /clear-files, /context, /context-add, /context-r
emove
- Support for images, PDFs, text files, and code files
- Vision model support for image attachments
- Update UI to show attached files and context information in header
- Add file configuration options
- Create comprehensive documentation (FILE_ATTACHMENTS.md)
- Update README with file attachment examples
```

Files Modified/Created

Modified Files

1. `cmd/chat.go` - Added file attachment and directory context support
2. `internal/config/config.go` - Added FileConfig struct
3. `README.md` - Updated with file attachment documentation

Created Files

1. `FILE_ATTACHMENTS.md` - Comprehensive user guide
2. `test_files/` - Test file directory
3. `IMPLEMENTATION_SUMMARY.md` - This file

Existing Files (Already Complete)

1. `cmd/root.go` - Inline mode file processing
2. `internal/fileprocessor/fileprocessor.go` - File processing logic

Usage Examples

Example 1: Analyze Code Files

```
termai chat -d ./src
# In chat: "Review the architecture"
```

Example 2: PDF Summary

```
termai -file report.pdf "Summarize key points"
```

Example 3: Image Analysis

```
termai chat
/attach screenshot.png
What issues do you see in this UI?
```

Example 4: Multi-File Comparison

```
termai -file v1.py -file v2.py "Compare these implementations"
```

Next Steps

The implementation is complete and ready for use. Additional enhancements could include:

1. **Recursive Directory Scanning** (optional flag)
2. **File Size Warnings** before processing large files
3. **Progress Bars** for large file processing
4. **File Type Detection** improvements
5. **Caching** for frequently used context files
6. **Batch File Processing** optimizations

Conclusion

✅ All requirements successfully implemented
✅ Code compiled without errors
✅ Comprehensive documentation created

✅ Changes committed to git
✅ Ready for production use

The TermAI file processing integration is complete with full support for:
- File attachments in both inline and chat modes
- Directory context for project-wide assistance
- Vision model support for images
- Comprehensive UI feedback
- Flexible configuration options
- Extensive documentation