Solo Git Project Structure

Comprehensive Directory and File Organization Guide

Last Updated: October 17, 2025

Table of Contents

Overview

Solo Git follows a clean, modular structure with clear separation between:
- **Core Logic** (`sologit/`) - Python package with business logic
- **User Interfaces** (CLI/TUI in `sologit/ui/` , GUI in `heaven-gui/`)
- **Tests** (`tests/`) - Comprehensive test suite (76% coverage)
- **Documentation** (`docs/`) - User guides, API docs, wiki
- **Infrastructure** (`infrastructure/`) - Deployment configs

Quick Navigation

```
solo-git/
├── sologit/          → Core Python package
├── heaven-gui/       → Desktop GUI (Tauri + React)
├── tests/            → Test suite (555 tests)
├── docs/             → Documentation
├── infrastructure/   → Deployment configs
├── data/             → Runtime data
└── [config files]    → Project configuration
```

Root Directory

```
solo-git/
├── .git/                          # Git repository metadata
├── .gitignore                     # Git ignore patterns
├── .archive/                      # Historical artifacts (not in Git)
│   └── historical_coverage/       # Old test coverage reports
│
├── README.md                      # Main project documentation
├── ARCHITECTURE.md                # System architecture guide
├── PROJECT_STRUCTURE.md           # This file
├── CHANGELOG.md                   # Version history
├── LICENSE                        # MIT license
│
├── requirements.txt               # Python dependencies
├── setup.py                       # Package installation script
├── pyproject.toml                 # Modern Python build config
├── pytest.ini                     # Pytest configuration
├── MANIFEST.in                    # Package manifest
│
├── current_coverage.json          # Latest test coverage data
├── test_run_output.txt            # Latest test execution log
│
├── QUICKSTART.md                  # Quick start guide
├── CLI_DEMO.txt                   # CLI demo transcript
├── CLI_INTEGRATION_SUMMARY.md     # CLI integration report
│
└── [Phase Reports]                # Development phase summaries
    ├── PHASE_0_SUMMARY.txt
    ├── PHASE_0_VERIFICATION_REPORT.md
    ├── PHASE_1_100_PERCENT_COMPLETION_REPORT.md
    ├── PHASE_1_ENHANCEMENTS_SUMMARY.md
    ├── PHASE_1_VERIFICATION_REPORT.md
    ├── PHASE_2_COMPLETION_REPORT.md
    ├── PHASE_2_COVERAGE_IMPROVEMENT_REPORT.md
    ├── PHASE_2_ENHANCED_COVERAGE_REPORT.md
    ├── PHASE_2_SUMMARY.md
    ├── PHASE_3_ENHANCEMENT_REPORT.md
    ├── PHASE_3_FINAL_SUMMARY.md
    ├── PHASE_3_SUMMARY.md
    ├── PHASE_4_READINESS_REPORT.md
    ├── GIT_ENGINE_100_PERCENT_COMPLETE.md
    ├── HEAVEN_CLI_INTEGRATION_REPORT.md
    ├── HEAVEN_INTERFACE_90_PERCENT_COMPLETION_REPORT.md
    ├── HEAVEN_INTERFACE_97_PERCENT_COMPLETION_REPORT.md
    ├── HEAVEN_INTERFACE_AUDIT_SUMMARY.md
    ├── HEAVEN_INTERFACE_GAP_ANALYSIS.md
    ├── HEAVEN_INTERFACE_IMPLEMENTATION_SUMMARY.md
    ├── IMPLEMENTATION_COMPLETION_GUIDE.md
    ├── IMPLEMENTATION_SUMMARY.md
    ├── DEBUGGING_SUMMARY.md
    ├── TESTING_REPORT.md
    ├── TEST_COVERAGE_IMPROVEMENT_REPORT.md
    ├── PHASE3_COVERAGE_BASELINE_REPORT.md
    └── PHASE3_COVERAGE_IMPROVEMENT_REPORT.md
```

Root Files Purpose

File	Purpose
`README.md`	Main documentation, quick start, feature overview
`ARCHITECTURE.md`	System design, components, data flow
`PROJECT_STRUCTURE.md`	This file - directory organization
`CHANGELOG.md`	Version history and release notes
`requirements.txt`	Python dependencies (pip format)
`setup.py`	Package installation and metadata
`pyproject.toml`	Modern Python build system config
`pytest.ini`	Pytest configuration (coverage, markers)
`current_coverage.json`	Latest test coverage metrics
`test_run_output.txt`	Latest test execution logs

Core Python Package (`sologit/`)

The main application package containing all business logic.

```
sologit/
├── __init__.py                      # Package initialization
│
├── cli/                             # Command-Line Interface
│   ├── __init__.py
│   ├── main.py                     # Entry point (evogitctl command)
│   ├── commands.py                 # Core CLI commands (repo, pad, test)
│   ├── config_commands.py          # Config management commands
│   ├── integrated_commands.py      # AI pairing commands
│   └── enhanced_commands.py        # Advanced commands (TUI launcher)
│
├── ui/                              # Heaven Interface (CLI/TUI)
│   ├── __init__.py
│   ├── formatter.py                # Rich console formatting
│   ├── theme.py                    # Heaven design tokens
│   ├── heaven_tui.py               # Main TUI application
│   ├── tui_app.py                  # TUI widgets and layouts
│   ├── command_palette.py          # Fuzzy command search widget
│   ├── file_tree.py                # File browser widget
│   ├── graph.py                    # ASCII commit graph renderer
│   ├── test_runner.py              # Test results display widget
│   ├── autocomplete.py             # Shell autocomplete engine
│   └── history.py                  # Command history management
│
├── state/                           # State Management
│   ├── __init__.py
│   ├── manager.py                  # State persistence (JSON)
│   ├── schema.py                   # State data models
│   └── git_sync.py                 # Git ↔ State synchronization
│
├── config/                          # Configuration
│   ├── __init__.py
│   ├── manager.py                  # Config loading and validation
│   └── templates.py                # Default config templates
│
├── api/                             # External API Clients
│   ├── __init__.py
│   └── client.py                   # Abacus.ai RouteLLM client
│
├── core/                            # Core Domain Models
│   ├── __init__.py
│   ├── repository.py               # Repository model
│   └── workpad.py                  # Workpad (ephemeral workspace) model
│
├── engines/                         # Execution Engines
│   ├── __init__.py
│   ├── git_engine.py               # Git operations wrapper
│   ├── patch_engine.py             # Patch generation and application
│   └── test_orchestrator.py       # Test execution and sandboxing
│
├── orchestration/                   # AI Orchestration
│   ├── __init__.py
│   ├── ai_orchestrator.py          # Main AI coordinator
│   ├── model_router.py             # Multi-model selection
│   ├── planning_engine.py          # High-level planning (GPT-4/Claude)
│   ├── code_generator.py           # Code generation (DeepSeek/CodeLlama)
│   └── cost_guard.py               # Budget tracking and enforcement
│
├── analysis/                        # Test Analysis
│   ├── __init__.py
│   └── test_analyzer.py            # Test failure diagnosis with AI
│
```

```
├── workflows/                      # Automated Workflows
│   ├── __init__.py
│   ├── auto_merge.py              # Auto-merge on green tests
│   ├── promotion_gate.py          # Promotion validation rules
│   ├── ci_orchestrator.py         # Jenkins/CI integration
│   └── rollback_handler.py        # Automatic rollback logic
│
└── utils/                          # Utilities
    ├── __init__.py
    └── logger.py                  # Structured logging
```

Module Dependencies

```
┌─────────┐
│   CLI   │ ──────────────────────┐
└─────────┘                       │
     │                            │
     ▼                            ▼
┌─────────┐                  ┌─────────┐
│   UI    │                  │ State   │
└─────────┘                  └─────────┘
     │                            │
     ▼                            ▼
┌──────────────────────────────────┐
│         Core Components          │
│  ┌──────────┐  ┌──────────────┐  │
│  │ Engines  │  │Orchestration │  │
│  └──────────┘  └──────────────┘  │
│                                  │
│  ┌──────────┐  ┌──────────────┐  │
│  │Workflows │  │   Analysis   │  │
│  └──────────┘  └──────────────┘  │
└──────────────────────────────────┘
         │
         ▼
┌────────────────┐
│ External APIs  │
└────────────────┘
```

Key Files Detail

CLI Layer

`cli/main.py` - Entry point

```
# Defines evogitctl command
# Sets up Click command groups
# Handles global options (--verbose, --config)
```

`cli/commands.py` - Core commands

```
# Commands: repo, pad, test, promote, rollback
# Implementation: delegates to engines and workflows
```

`cli/integrated_commands.py` - AI pairing

```
# Commands: pair, plan, code
# Integration: uses AI orchestration layer
```

UI Layer

`ui/heaven_tui.py` - Main TUI

```
# Textual app with multiple screens
# Layouts: split panes, tabs, modals
# Bindings: vim-style keyboard shortcuts
```

`ui/command_palette.py` - Command palette

```
# Fuzzy search over all commands
# Keyboard shortcut: Ctrl+P
# Shows recent and favorite commands
```

`ui/graph.py` - Commit graph

```
# ASCII art commit graph
# Shows: commits, branches, merges, CI status
# Color-coded: green (success), red (failure)
```

State Layer

`state/manager.py` - State persistence

```
# Load/save state to .sologit/state.json
# Thread-safe operations
# Change notifications for live updates
```

`state/git_sync.py` - Git synchronization

```
# Sync state with Git repository
# Detect external Git changes
# Update state on Git operations
```

Engines

`engines/git_engine.py` - Git wrapper

```
# Operations: commit, merge, rebase, checkout
# Constraint: Fast-forward merges only
# Safety: Validates before destructive operations
```

`engines/patch_engine.py` - Patch handling

```
# Generate unified diffs from AI changes
# Apply patches with conflict detection
# Rollback support
```

`engines/test_orchestrator.py` - Test execution

```
# Run tests in isolated sandboxes
# Parallel execution support
# Aggregate and report results
```

Orchestration

`orchestration/ai_orchestrator.py` - AI coordinator

```
# Coordinates planning, coding, analysis
# Handles model selection
# Manages API requests
```

`orchestration/model_router.py` - Model selection

```
# Routes tasks to optimal model tier
# Escalation rules (complexity, security)
# Fallback on failures
```

`orchestration/cost_guard.py` - Budget enforcement

```
# Track spending per model
# Enforce daily budget caps
# Alert on thresholds
```

Workflows

`workflows/auto_merge.py` - Auto-merge

```
# Test → Gate → Merge pipeline
# Configurable rules
# Event notifications
```

`workflows/promotion_gate.py` - Promotion validation

```
# Required checks (tests, conflicts)
# Optional checks (coverage, linting)
# Manual override support
```

Heaven GUI (`heaven-gui/`)

Desktop application built with Tauri (Rust) + React (TypeScript).

```
heaven-gui/
├── src/                          # React frontend
│   ├── App.tsx                   # Root component
│   ├── main.tsx                  # React entry point
│   │
│   ├── components/               # UI components
│   │   ├── CodeEditor.tsx        # Monaco editor wrapper
│   │   ├── CommandPalette.tsx    # Command search (Ctrl+P)
│   │   ├── CommitGraph.tsx       # D3.js visualization
│   │   ├── FileTree.tsx          # File browser with icons
│   │   ├── TestDashboard.tsx     # Test results with charts
│   │   ├── AIAssistant.tsx       # Chat interface
│   │   ├── StatusBar.tsx         # Bottom status bar
│   │   ├── SettingsPanel.tsx     # Configuration UI
│   │   ├── DiffViewer.tsx        # Side-by-side diff
│   │   └── NotificationToast.tsx # Toast notifications
│   │
│   ├── hooks/                    # React hooks
│   │   ├── useStateSync.ts       # Sync with backend state
│   │   ├── useCommands.ts        # Execute backend commands
│   │   ├── useWebSocket.ts       # Live updates via WebSocket
│   │   ├── useTheme.ts           # Heaven theme management
│   │   └── useKeyboardShortcuts.ts # Global shortcuts
│   │
│   ├── services/                 # Business logic
│   │   ├── api.ts                # Backend API client
│   │   ├── state.ts              # State management
│   │   ├── ipc.ts                # Tauri IPC bridge
│   │   ├── git.ts                # Git operations
│   │   └── websocket.ts          # WebSocket client
│   │
│   └── styles/                   # Styling
│       ├── theme.css             # Heaven design tokens
│       ├── components.css        # Component styles
│       └── layout.css            # Grid and layout
│
├── src-tauri/                    # Rust backend
│   ├── src/
│   │   ├── main.rs               # Tauri app setup
│   │   ├── commands.rs           # IPC command handlers
│   │   ├── state.rs              # State bridge to Python CLI
│   │   ├── events.rs             # Event emitter
│   │   └── menu.rs               # Application menu
│   │
│   ├── Cargo.toml                # Rust dependencies
│   ├── tauri.conf.json           # Tauri configuration
│   └── build.rs                  # Build script
│
├── dist/                         # Production build output
│   ├── index.html
│   └── assets/
│
├── node_modules/                 # npm dependencies (large)
│
├── index.html                    # HTML entry point
├── package.json                  # npm dependencies and scripts
├── package-lock.json             # Dependency lock file
├── tsconfig.json                 # TypeScript config
├── tsconfig.node.json            # TypeScript config (Vite)
├── vite.config.ts                # Vite bundler config
├── .gitignore                    # Git ignore
│
```

```
└── [Documentation]
    ├── README.md                    # GUI-specific README
    ├── BUILDING.md                  # Build instructions
    ├── DEVELOPMENT.md               # Development guide
    └── UX_AUDIT_REPORT.md           # UX audit findings
```

Key Frontend Components

CodeEditor.tsx
- Monaco editor (VS Code editor)
- Syntax highlighting
- Code completion
- Diff mode support

CommitGraph.tsx
- D3.js force-directed graph
- Interactive (pan, zoom, click)
- Jenkins build status badges
- Color-coded by status

CommandPalette.tsx
- Fuzzy search (fuse.js)
- Keyboard shortcuts (Ctrl+P, Ctrl+K)
- Recent commands
- Quick actions

TestDashboard.tsx
- Test results table
- Coverage charts (Recharts)
- Failure analysis
- Historical trends

Key Backend (Rust) Files

main.rs

```
// Tauri setup
// Window configuration
// System tray integration
```

commands.rs

```
// IPC commands exposed to frontend
// Examples:
// - execute_git_command()
// - get_repository_state()
// - run_tests()
// - call_ai_model()
```

state.rs

```
// Bridge to Python CLI
// Spawns evogitctl as subprocess
// Parses JSON state output
```

Tests (`tests/`)

Comprehensive test suite with 555 tests and 76% coverage.

```
tests/
├── __init__.py
│
├── conftest.py                      # Pytest fixtures and configuration
│
├── [Core Tests]
│   ├── test_core.py                # Repository and Workpad tests
│   ├── test_core_100_percent.py    # Extended core tests
│   └── test_workpad_enhancements.py
│
├── [Engine Tests]
│   ├── test_git_engine.py          # Git operations tests
│   ├── test_git_engine_100_percent.py
│   ├── test_git_engine_extended.py
│   ├── test_patch_engine.py        # Patch application tests
│   ├── test_patch_engine_100_percent.py
│   ├── test_patch_engine_enhanced.py
│   ├── test_test_orchestrator_comprehensive.py
│   └── test_test_analyzer.py
│
├── [AI Orchestration Tests]
│   ├── test_ai_orchestrator.py
│   ├── test_ai_orchestrator_coverage.py
│   ├── test_ai_orchestrator_enhanced.py
│   ├── test_model_router.py
│   ├── test_model_router_coverage.py
│   ├── test_model_router_enhanced.py
│   ├── test_planning_engine.py
│   ├── test_planning_engine_coverage.py
│   ├── test_planning_engine_enhanced.py
│   ├── test_code_generator.py
│   ├── test_code_generator_coverage.py
│   ├── test_code_generator_enhanced.py
│   ├── test_cost_guard.py
│   ├── test_cost_guard_coverage.py
│   └── test_cost_guard_enhanced.py
│
├── [Workflow Tests]
│   ├── test_auto_merge_enhanced.py
│   ├── test_promotion_gate.py
│   ├── test_promotion_gate_coverage_boost.py
│   ├── test_promotion_gate_enhanced.py
│   ├── test_ci_orchestrator_enhanced.py
│   ├── test_ci_orchestrator_coverage_boost.py
│   ├── test_rollback_handler_comprehensive.py
│   ├── test_test_analyzer_coverage_boost.py
│   ├── test_phase3_workflows.py
│   └── test_phase3_enhanced_mocks.py
│
└── [E2E Tests]
    └── test_workflow_e2e.py        # End-to-end workflow tests
```

Test Categories

Category	Purpose	Count
Core	Repository, Workpad models	50+
Engine	Git, Patch, Test engines	100+
AI	Orchestration, routing, cost	150+
Workflow	Auto-merge, CI, rollback	100+
E2E	Full workflow scenarios	50+
Analysis	Test analyzer	50+

Test Execution

```
# Run all tests
pytest tests/ -v

# Run specific category
pytest tests/test_core*.py -v
pytest tests/test_ai*.py -v
pytest tests/test_phase3*.py -v

# Run with coverage
pytest tests/ --cov=sologit --cov-report=html

# Run specific test
pytest tests/test_git_engine.py::test_fast_forward_merge -v
```

Documentation (`docs/`)

Comprehensive documentation organized by topic.

```
docs/
├── wiki/                           # Documentation wiki
│   ├── Home.md                     # Wiki home page
│   ├── README.md                   # Wiki navigation
│   │
│   ├── architecture/               # Architecture docs
│   │   ├── core-components.md
│   │   └── git-engine.md
│   │
│   ├── guides/                     # User guides
│   │   ├── quick-start.md
│   │   ├── setup-guide.md
│   │   ├── cli-reference.md
│   │   ├── config-reference.md
│   │   └── phase3-usage-examples.md
│   │
│   ├── phases/                     # Phase completion reports
│   │   ├── phase-0-overview.md
│   │   ├── phase-0-completion.md
│   │   ├── phase-0-verification.md
│   │   ├── phase-1-overview.md
│   │   ├── phase-1-completion.md
│   │   ├── phase-1-enhancements.md
│   │   ├── phase-2-completion.md
│   │   ├── phase-2-enhanced-coverage.md
│   │   ├── phase-3-completion.md
│   │   └── phase-4-completion.md
│   │
│   └── timeline/                   # Project timeline
│       ├── 2025-10-16-vision.md
│       ├── 2025-10-16-concept.md
│       └── 2025-10-16-game-plan.md
│
├── examples/                       # Code examples
│
├── api/                            # API documentation
│
├── [Main Docs]
│   ├── API.md                      # Complete API reference
│   ├── SETUP.md                    # Installation guide
│   ├── SETUP_GUIDE.md              # Detailed setup
│   ├── HEAVEN_INTERFACE.md         # Design system spec
│   ├── HEAVEN_INTERFACE_GUIDE.md   # Implementation guide
│   ├── KEYBOARD_SHORTCUTS.md       # Keyboard reference
│   ├── TESTING_GUIDE.md            # Testing guide
│   ├── BETA_LAUNCH_CHECKLIST.md    # Beta launch tasks
│   ├── UX_AUDIT_REPORT.md          # UX audit findings
│   └── PHASE_4_COMPLETION_REPORT.md
```

Documentation Categories

User Guides:
- Quick Start: Get running in 5 minutes
- Setup Guide: Detailed installation
- CLI Reference: All commands and options
- Config Reference: Configuration options

Technical Docs:
- API: Complete API documentation
- Architecture: System design (see ARCHITECTURE.md)

- Heaven Interface: UI design system
- Testing Guide: How to write and run tests

Phase Reports:
- Phase 0-4 completion reports
- Coverage improvement reports
- Enhancement summaries
- Verification reports

Infrastructure (`infrastructure/`)

Deployment and CI/CD configuration.

```
infrastructure/
├── docker/                          # Docker images
│   ├── Dockerfile.cli              # CLI container
│   ├── Dockerfile.sandbox          # Test sandbox
│   └── docker-compose.yml          # Multi-container setup
│
├── jenkins/                         # Jenkins CI/CD
│   ├── Jenkinsfile                 # Pipeline definition
│   ├── jenkins-config.xml          # Jenkins configuration
│   └── plugins.txt                 # Required plugins
│
└── sandbox/                         # Test sandbox configs
    ├── sandbox.yml                 # Sandbox configuration
    └── entrypoint.sh               # Sandbox entry script
```

Docker Setup

CLI Container (`Dockerfile.cli`):

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY sologit/ ./sologit/
CMD ["evogitctl"]
```

Sandbox Container (`Dockerfile.sandbox`):
- Isolated test environment
- No network access
- Timeout enforcement
- Resource limits

Configuration Files

Python Configuration

`setup.py`

```
# Package metadata
# Dependencies
# Entry points (evogitctl command)
```

`pyproject.toml`

```
# Build system requirements
# Project metadata
# Tool configurations (black, isort, mypy)
```

`pytest.ini`

```
# Test discovery patterns
# Coverage settings
# Markers and options
```

`requirements.txt`

```
# Production dependencies
click>=8.0.0
textual>=0.40.0
rich>=13.0.0
pydantic>=2.0.0
requests>=2.31.0
# ... more
```

GUI Configuration

`package.json`

```
{
  "dependencies": {
    "react": "^18.2.0",
    "typescript": "^5.2.0",
    "monaco-editor": "^0.44.0",
    "d3": "^7.8.0",
    "recharts": "^2.10.0"
  }
}
```

`tauri.conf.json`

```
{
  "build": {
    "distDir": "../dist"
  },
  "tauri": {
    "allowlist": {
      "all": false,
      "shell": {
        "execute": true
      }
    }
  }
}
```

Data and Runtime

Data Directory (`data/`)

```
data/
├── repos/                          # Repository storage
│   └── [repo-id]/                  # Each repo in subdirectory
│       ├── .git/                   # Git metadata
│       ├── .sologit/               # Solo Git state
│       │   ├── state.json          # Repository state
│       │   ├── config.yaml         # Repo-specific config
│       │   └── cache/              # Temporary cache
│       └── [project files]
│
└── logs/                           # Application logs
    ├── evogitctl.log               # CLI logs
    ├── ai_requests.log             # AI API request logs
    ├── test_runs.log               # Test execution logs
    └── [date]-audit.log            # Daily audit logs
```

User Configuration

```
~/.sologit/                         # Global config directory
├── config.yaml                     # Global configuration
├── credentials.enc                 # Encrypted API keys
├── history.json                    # Command history
└── cache/                          # Global cache
    ├── models/                     # Model response cache
    └── tmp/                        # Temporary files
```

State File Structure

`.sologit/state.json` example:

```
{
  "version": "0.4.0",
  "repository": {
    "id": "repo-123",
    "trunk": "main",
    "workpads": {
      "pad-abc": {
        "id": "pad-abc",
        "title": "add-auth",
        "status": "ACTIVE",
        "base_commit": "abc123",
        "patches": [],
        "test_results": []
      }
    }
  },
  "ai_metrics": {
    "total_requests": 42,
    "total_cost_usd": 2.45,
    "daily_budget_remaining": 7.55
  }
}
```

File Naming Conventions

Python Files

- **Snake case**: `file_name.py`
- **Test files**: `test_*.py` (pytest discovery)
- **Private modules**: `_internal.py` (single underscore)
- **Package markers**: `__init__.py`

TypeScript/React Files

- **PascalCase** for components: `CodeEditor.tsx`
- **camelCase** for utilities: `apiClient.ts`
- **kebab-case** for CSS: `theme-tokens.css`

Documentation

- **ALL CAPS** for major docs: `README.md`, `LICENSE`
- **Snake case** for guides: `setup_guide.md`
- **Hyphens** for multi-word: `cli-reference.md`

Configuration

- **Lowercase** with extensions: `pytest.ini`, `setup.py`
- **Dotfiles**: `.gitignore`, `.env`

File Metadata

File Sizes (Approximate)

Category	Files	Total Size
Python Source	100+	~500 KB
Tests	40+	~400 KB
Documentation	80+	~2 MB
Heaven GUI (src)	50+	~200 KB
node_modules	5000+	~200 MB
Git History	-	~50 MB

Critical Files

Files that should **never** be deleted:

1. `README.md` - Main documentation
2. `setup.py` - Package installation
3. `requirements.txt` - Dependencies
4. `sologit/cli/main.py` - CLI entry point
5. `sologit/core/repository.py` - Core model
6. `pytest.ini` - Test configuration
7. `.gitignore` - Git ignore rules
8. `LICENSE` - Legal license

Summary

Total Structure:
- **Python Modules**: 30+ modules
- **Test Files**: 40+ test files
- **UI Components**: 15+ React components
- **Documentation**: 80+ markdown files
- **Configuration Files**: 10+ config files

Key Directories:
1. `sologit/` - Core application (500 KB)
2. `heaven-gui/` - Desktop GUI (200 MB with deps)
3. `tests/` - Test suite (400 KB)
4. `docs/` - Documentation (2 MB)

Cleaned Up:
- ✅ Removed 63 duplicate PDF files
- ✅ Archived historical coverage reports

- ✅ Removed Python bytecode files
- ✅ Organized phase reports
- ✅ No tar.gz archives

Maintenance Notes

Regular Cleanup

```
# Remove Python bytecode
find . -type d -name "__pycache__" -exec rm -rf {} +
find . -type f -name "*.pyc" -delete

# Remove temporary files
rm -rf .pytest_cache/
rm -rf htmlcov/
rm -f .coverage

# Update documentation dates
# Update test coverage numbers
# Archive old phase reports
```

Adding New Components

New Python Module:
1. Create in appropriate `sologit/` subdirectory
2. Add `__init__.py` if new package
3. Write tests in `tests/test_[module].py`
4. Update `ARCHITECTURE.md` if significant

New React Component:
1. Create in `heaven-gui/src/components/`
2. Export from `components/index.ts`
3. Add to storybook if available
4. Document props with TypeScript

New Documentation:
1. Create in `docs/` or `docs/wiki/`
2. Link from `README.md` or `docs/wiki/Home.md`
3. Follow markdown style guide
4. Add to table of contents

Version History

- **v0.1.0** (Phase 0): Initial structure
- **v0.2.0** (Phase 1): Git engine and workpads
- **v0.3.0** (Phase 2): AI orchestration
- **v0.4.0** (Phase 3): Workflows and auto-merge
- **v0.4.5** (Phase 4): Heaven Interface + cleanup

See Also

- ARCHITECTURE.md (ARCHITECTURE.md) - System architecture
- README.md (README.md) - Main documentation
- docs/SETUP.md (docs/SETUP.md) - Installation guide
- docs/API.md (docs/API.md) - API reference