Solo Git & Heaven Interface - Testing Guide

Version: 0.4.0
Last Updated: October 17, 2025
Completion Level: >97%

📋 Table of Contents

Overview

This guide covers how to test Solo Git and the Heaven Interface at all levels:

- **Unit Tests**: Individual components
- **Integration Tests**: Component interaction
- **System Tests**: End-to-end workflows
- **Manual Tests**: User interface verification

Prerequisites

```
# Install test dependencies
pip install -e .[dev]

# Or install individually
pip install pytest pytest-cov pytest-asyncio
pip install textual-dev  # For TUI testing
```

Testing Philosophy

Tests Are The Review

In Solo Git, tests replace code review:

✅ **Green tests** = Safe to merge
❌ **Red tests** = Stay in workpad
⏳ **No tests** = Not ready

Test Levels

1. **Fast Tests** (`--target fast`): Unit + integration (~10s)
2. **Full Tests** (`--target full`): Everything including E2E (~60s)
3. **Smoke Tests**: Post-merge verification (~30s)

Coverage Goals

- **Core Engines**: >95% coverage
- **CLI Commands**: >90% coverage
- **UI Components**: >85% coverage
- **Integration**: >80% coverage

Testing the CLI

Manual CLI Testing

1. Basic Commands

```
# Test help system
evogitctl --help
evogitctl repo --help
evogitctl pad --help
evogitctl test --help

# Test version
evogitctl version

# Test hello
evogitctl hello
```

Expected: All commands show help text with proper formatting.

2. Repository Operations

```
# Create test repo
cd /tmp
zip -r test.zip myapp/

# Initialize repository
evogitctl repo init --zip test.zip --name "TestRepo"

# Verify
evogitctl repo list
evogitctl repo info <repo-id>
```

Expected:
- Repository appears in list
- Info shows correct details
- Rich table formatting visible

3. Workpad Operations

```
# Create workpad
evogitctl pad create "test feature" --repo <repo-id>

# Verify
evogitctl pad list
evogitctl pad info <pad-id>

# View diff
evogitctl pad diff <pad-id>
```

Expected:
- Workpad created successfully
- List shows workpad with status
- Diff shows changes (if any)

4. Test Execution

```
# Run fast tests
evogitctl test run <pad-id> --target fast

# Run full tests
evogitctl test run <pad-id> --target full
```

Expected:
- Progress indicator appears
- Results table shows pass/fail
- Summary panel displays totals

5. AI Operations

```
# Test pair command
evogitctl pair "add hello world function"

# Test AI commands
evogitctl ai generate "simple function"
evogitctl ai review <pad-id>
```

Expected:
- AI responds with plan
- Code generated/reviewed
- Tests run automatically

6. Interactive Mode

```
# Launch interactive shell
evogitctl interactive

# In shell:
> repo list
> pad create "test"
> <Tab>  # Test autocomplete
> <Up>   # Test history
> <Ctrl+R>  # Test search
> <Ctrl+D>  # Exit
```

Expected:
- Autocomplete works
- History navigable
- Commands execute
- Search finds commands

Automated CLI Testing

Run Test Suite

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

# Run specific test file
pytest tests/cli/test_commands.py -v

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

Test Rich Formatting

```
# Test table rendering
pytest tests/cli/test_formatting.py::test_repo_list_formatting

# Test panel rendering
pytest tests/cli/test_formatting.py::test_repo_info_panel

# Test progress indicator
pytest tests/cli/test_formatting.py::test_test_run_progress
```

Test Interactive Features

```
# Test autocomplete
pytest tests/ui/test_autocomplete.py

# Test command history
pytest tests/ui/test_history.py
```

Testing the TUI

Manual TUI Testing

1. Launch and Layout

```
# Launch Heaven TUI
evogitctl heaven

# Or with repo
evogitctl heaven --repo /path/to/repo
```

Verify:
- [ ] TUI launches without errors
- [ ] All panels visible
- [ ] Layout matches design (30-40-30 split)
- [ ] Header shows title and time
- [ ] Footer shows shortcuts
- [ ] Status bar shows context

2. Keyboard Navigation

Test these shortcuts:

```
✓ Ctrl+P      - Command palette opens
✓ ?           - Help screen appears
✓ R           - Panels refresh
✓ Ctrl+Q      - Quit confirmation
✓ Ctrl+Z/Y    - Undo/redo
✓ Ctrl+T      - Test runner activates
✓ Tab         - Panel focus cycles
```

How to test:
1. Press each key combination
2. Verify expected action occurs
3. Check status bar updates
4. Verify no error messages

3. Command Palette

```
# In TUI, press Ctrl+P
```

Test searches:
- Type: `test` → Should show test commands
- Type: `cr` → Should show “create” commands
- Type: `ai` → Should show AI commands
- Use arrow keys → Selection moves
- Press Enter → Command executes
- Press Esc → Palette closes

Verify:
- [ ] Fuzzy matching works
- [ ] Results update in real-time

- [ ] Shortcuts displayed
- [ ] Categories shown
- [ ] Execution works

4. Panel Functionality

Left Panel (Commit Graph):
- [ ] Shows recent commits
- [ ] Visual graph renders
- [ ] Commit messages visible
- [ ] Test status icons show

Left Panel (File Tree):
- [ ] File structure visible
- [ ] Git status indicators work
- [ ] Can expand/collapse folders
- [ ] File selection works

Center Panel (Workpad):
- [ ] Active workpad shown
- [ ] Status updates
- [ ] Checkpoint count accurate

Center Panel (AI Activity):
- [ ] Operations listed
- [ ] Cost tracking visible
- [ ] Status updates real-time

Right Panel (Test Runner):
- [ ] Tests can be triggered
- [ ] Output streams in real-time
- [ ] Results color-coded
- [ ] Summary accurate

Right Panel (Diff Viewer):
- [ ] Shows file changes
- [ ] Additions in green
- [ ] Deletions in red
- [ ] Syntax highlighting

5. Real-Time Updates

Test update flow:
1. Launch TUI in one terminal
2. In another terminal: `evogitctl pad create "test"`
3. Switch back to TUI
4. Press `R` to refresh
5. Verify new workpad appears

Verify:
- [ ] State updates after refresh
- [ ] No stale data shown
- [ ] Panels synchronize

6. Error Handling

Test error scenarios:
- Invalid pad ID in command palette
- Network error during AI operation
- File not found in file tree
- Test execution failure

Verify:
- [ ] Error messages appear
- [ ] UI remains stable
- [ ] Can recover gracefully

Automated TUI Testing

Using Textual Dev Tools

```
# Install textual dev tools
pip install textual-dev

# Run with console
textual console

# In another terminal
evogitctl heaven

# Watch logs in console
```

Snapshot Testing

```
# Run TUI snapshot tests
pytest tests/ui/test_tui.py --snapshot-update

# Compare snapshots
pytest tests/ui/test_tui.py
```

Component Testing

```
# Test individual widgets
pytest tests/ui/test_file_tree.py
pytest tests/ui/test_test_runner.py
pytest tests/ui/test_command_palette.py
```

Testing the GUI

Manual GUI Testing

1. Build and Launch

```
# Build frontend
cd heaven-gui
npm install
npm run build

# (Optional) Build Tauri if Rust installed
npm run tauri:build

# Launch dev server
npm run dev
```

Verify:
- [ ] GUI window opens
- [ ] No console errors
- [ ] All components load
- [ ] Theme applied correctly

2. Component Testing

Monaco Editor:
- [ ] Opens files
- [ ] Syntax highlighting works
- [ ] Can edit and save
- [ ] Find/replace works
- [ ] Line numbers visible

Commit Graph:
- [ ] Renders without errors
- [ ] Commits visible
- [ ] Can zoom/pan
- [ ] Click shows details
- [ ] Colors indicate status

Test Dashboard:
- [ ] Charts render
- [ ] Data updates
- [ ] Interactive tooltips
- [ ] Export works

AI Assistant:
- [ ] Chat interface works
- [ ] Can send messages
- [ ] Responses stream
- [ ] Code blocks formatted
- [ ] Cost displayed

File Browser:
- [ ] Tree structure visible

- [ ] Can expand/collapse
- [ ] File icons shown
- [ ] Git status colors
- [ ] Double-click opens

Settings Panel:
- [ ] All options visible
- [ ] Changes save
- [ ] Validation works
- [ ] Reset to defaults

3. Keyboard Shortcuts

Test in GUI:
- `Ctrl+P` - Command palette
- `Ctrl+N` - New workpad
- `Ctrl+T` - Run tests
- `Ctrl+S` - Save file
- `Ctrl+F` - Find
- `Ctrl+/` - Toggle comment

Verify:
- [ ] All shortcuts work
- [ ] No conflicts
- [ ] Help shows shortcuts

4. State Synchronization

Test sync:
1. Open GUI
2. Note current workpad
3. In terminal: `evogitctl pad create "test"`
4. In GUI: Click refresh or wait for auto-refresh
5. Verify new workpad appears

Verify:
- [ ] State syncs from CLI to GUI
- [ ] State syncs from GUI to CLI
- [ ] No conflicts
- [ ] Real-time updates work

Automated GUI Testing

Frontend Unit Tests

```
# Run React component tests
cd heaven-gui
npm test

# With coverage
npm test -- --coverage
```

E2E Testing (if Playwright configured)

```
# Run E2E tests
npm run test:e2e

# Specific test
npm run test:e2e -- --grep "commit graph"
```

Integration Testing

CLI-TUI Integration

Test scenario:
1. Create repo via CLI
2. Launch TUI
3. Create workpad in TUI (via command palette)
4. Switch back to CLI
5. Verify workpad visible: `evogitctl pad list`

Verify:
- [ ] State shared correctly
- [ ] No data loss
- [ ] Timestamps accurate

CLI-GUI Integration

Test scenario:
1. Launch GUI
2. Note active repo/workpad
3. In terminal: `evogitctl test run <pad-id>`
4. Watch GUI test dashboard
5. Verify results appear

Verify:
- [ ] Test results sync to GUI
- [ ] Live updates work
- [ ] Dashboard updates

AI Integration

Test scenario:

```
# Ensure Abacus.ai credentials configured
evogitctl config setup

# Test AI pair
evogitctl pair "create hello world function"
```

Verify:
- [ ] AI responds
- [ ] Code generated
- [ ] Tests run

- [ ] Cost tracked
- [ ] Results logged

Git Integration

Test scenario:

```
# Create and promote workpad
evogitctl pad create "test"
# ... make changes ...
evogitctl test run <pad-id>
evogitctl pad promote <pad-id>

# Verify in git
cd ~/.sologit/data/repos/<repo-id>
git log
git diff HEAD~1
```

Verify:
- [ ] Commits appear in git
- [ ] Branch operations work
- [ ] Merges are fast-forward
- [ ] Tags created

Test Suite Reference

Running All Tests

```
# Full test suite
pytest

# With verbose output
pytest -v

# With coverage
pytest --cov=sologit --cov-report=html

# Parallel execution
pytest -n auto
```

Test Organization

```
tests/
├── unit/                 # Unit tests
│   ├── test_git_engine.py
│   ├── test_patch_engine.py
│   └── test_state_manager.py
├── cli/                  # CLI tests
│   ├── test_commands.py
│   ├── test_formatting.py
│   └── test_main.py
├── ui/                   # UI tests
│   ├── test_tui.py
│   ├── test_file_tree.py
│   ├── test_command_palette.py
│   └── test_autocomplete.py
├── integration/          # Integration tests
│   ├── test_cli_tui.py
│   ├── test_state_sync.py
│   └── test_ai_workflow.py
└── e2e/                  # End-to-end tests
    ├── test_basic_workflow.py
    ├── test_ai_pairing.py
    └── test_test_workflow.py
```

Running Specific Tests

```
# By directory
pytest tests/unit/
pytest tests/cli/
pytest tests/ui/

# By file
pytest tests/cli/test_commands.py

# By test function
pytest tests/cli/test_commands.py::test_repo_list

# By marker
pytest -m "slow"
pytest -m "not slow"

# By keyword
pytest -k "repo"
pytest -k "test_run"
```

Test Markers

```
# Available markers
@pytest.mark.slow           # Slow tests (>1s)
@pytest.mark.integration    # Integration tests
@pytest.mark.cli            # CLI-specific
@pytest.mark.tui            # TUI-specific
@pytest.mark.gui            # GUI-specific
@pytest.mark.ai             # Requires AI API
@pytest.mark.smoke          # Smoke tests
```

Coverage Reports

```
# Generate HTML coverage
pytest --cov=sologit --cov-report=html

# Open in browser
open htmlcov/index.html

# Generate terminal report
pytest --cov=sologit --cov-report=term-missing

# Generate XML for CI
pytest --cov=sologit --cov-report=xml
```

CI/CD Integration

GitHub Actions

```
# .github/workflows/test.yml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: |
          pip install -e .[dev]

      - name: Run tests
        run: |
          pytest --cov=sologit --cov-report=xml

      - name: Upload coverage
        uses: codecov/codecov-action@v3
```

Jenkins Integration

```
pipeline {
    agent any

    stages {
        stage('Install') {
            steps {
                sh 'pip install -e .[dev]'
            }
        }

        stage('Test') {
            steps {
                sh 'pytest --cov=sologit --cov-report=xml --junitxml=junit.xml'
            }
        }

        stage('Coverage') {
            steps {
                publishCoverage adapters: [coberturaAdapter('coverage.xml')]
            }
        }
    }

    post {
        always {
            junit 'junit.xml'
        }
    }
}
```

Pre-commit Hooks

```
# .pre-commit-config.yaml
repos:
  - repo: local
    hooks:
      - id: pytest-fast
        name: Run fast tests
        entry: pytest tests/unit/ -x
        language: system
        pass_filenames: false
```

Troubleshooting Tests

Common Issues

Tests Hanging

```
# Run with timeout
pytest --timeout=10

# Show slow tests
pytest --durations=10
```

Import Errors

```
# Reinstall in editable mode
pip install -e .

# Check Python path
python -c "import sys; print(sys.path)"
```

Flaky Tests

```
# Run multiple times
pytest --count=10 tests/integration/

# Show flaky tests
pytest --flake-finder --flake-runs=10
```

Coverage Not Updating

```
# Clean old data
rm -rf .coverage htmlcov/

# Run fresh
pytest --cov=sologit --cov-report=html
```

Debug Mode

```
# Run with pdb on failure
pytest --pdb

# Run with verbose logging
pytest -v --log-cli-level=DEBUG

# Show local variables on failure
pytest -l
```

Test Data Cleanup

```
# Clean test repositories
rm -rf /tmp/sologit-test-*

# Clean state
rm -rf ~/.sologit/state/test_*

# Reset to clean state
pytest tests/ --setup-show
```

Test Checklist

Before Release

- [ ] All unit tests pass
- [ ] All integration tests pass

- [ ] Coverage >90% for core
- [ ] CLI manually tested
- [ ] TUI manually tested
- [ ] GUI manually tested (if built)
- [ ] State sync verified
- [ ] AI integration tested
- [ ] Documentation updated
- [ ] Changelog updated

Quick Smoke Test

```
# 2-minute smoke test
evogitctl --version
evogitctl hello
evogitctl repo list
evogitctl heaven  # Launch and press Ctrl+Q
pytest tests/unit/ -x  # Stop on first failure
```

Full Regression

```
# 10-minute full test
pytest --cov=sologit --cov-report=term-missing
pytest tests/integration/
python examples/demo_basic.py
evogitctl heaven  # Manual TUI test
```

Continuous Testing

Watch Mode (Development)

```
# Auto-run tests on file changes
pytest-watch

# Or with coverage
ptw -- --cov=sologit
```

Test-Driven Development

1. **Write test first:**

```
def test_new_feature():
    result = new_feature()
    assert result == expected
```

1. **Run and watch it fail:**

```
pytest tests/unit/test_new.py -x
```

1. **Implement feature:**

```python
def new_feature():
    # Implementation
    return expected
```

1. **Run and watch it pass:**

```
pytest tests/unit/test_new.py -x
```

Performance Testing

Benchmarking

```bash
# Install pytest-benchmark
pip install pytest-benchmark

# Run benchmarks
pytest tests/benchmarks/ --benchmark-only

# Compare results
pytest tests/benchmarks/ --benchmark-compare
```

Load Testing

```bash
# Test concurrent operations
pytest tests/load/ --workers=10

# Test with large repositories
pytest tests/load/ --repo-size=large
```

Summary

Quick Reference

Fast check:

```
pytest tests/unit/ -x
```

Full check:

```
pytest --cov=sologit --cov-report=term-missing
```

Manual check:

```
evogitctl repo list
evogitctl pad list
evogitctl heaven
```

Key Points

1. **Tests Are The Review** - Green tests = safe to merge
2. **Test at all levels** - Unit, integration, E2E
3. **Automate everything** - CI/CD pipeline
4. **Manual testing matters** - UI/UX verification
5. **Coverage goals** - >90% for core code

Getting Help

- **Test failures**: Check logs with `-v`
- **Coverage gaps**: Use `--cov-report=html`
- **Flaky tests**: Run with `--count=10`
- **Debugging**: Use `--pdb`

Heaven Interface - Where tests are the review.