Audityzer Branching Strategy - Trunk-Based Development

Overview

Audityzer implements a trunk-based development workflow optimized for security auditing and fuzzing tool development. This strategy accelerates integration cycles, reduces branch management overhead, and maintains high code quality for our security-focused platform.

Core Principles

1. Trunk-Based Development

- **Main Branch**: `safe-improvements` serves as our trunk and source of truth
- **Short-lived Feature Branches**: Maximum 2-3 days before integration
- **Rapid Integration**: Multiple daily integrations to maintain development velocity
- **Continuous Testing**: Automated security scanning and testing on all branches

2. Security-First Approach

- All branches undergo automated security analysis
- Vulnerability scanning before merge approval
- Fuzzing tests for security plugin integrations
- Code quality gates focused on security best practices

Branch Types

Trunk Branch: `safe-improvements`

- Primary development branch for security auditing features
- Always deployable and production-ready
- Receives all feature branch merges
- Protected with automated quality gates

Feature Branches

- **Naming Convention**: `feature/security-[component]-[description]`
- **Examples**:
- `feature/security-plugin-framework`
- `feature/security-vulnerability-scanner`
- `feature/security-fuzzing-engine`
- **Lifecycle**: Create → Develop → Test → Merge → Delete
- **Maximum Duration**: 72 hours

Integration Branches (Temporary)

- `feature/community-portal` - Community engagement for security researchers
- `feature/marketing-automation` - Outreach for security auditing bounty program
- These will be integrated sequentially into trunk

Workflow Process

1. Feature Development

```
# Create feature branch from trunk
git checkout safe-improvements
git pull origin safe-improvements
git checkout -b feature/security-new-scanner

# Develop with frequent commits
git add .
git commit -m "feat: implement vulnerability detection engine"

# Push and create PR
git push origin feature/security-new-scanner
```

2. Integration Process

- Automated CI/CD pipeline triggers on push
- Security scanning (SAST, dependency check)
- Automated testing of security plugins
- Code review focusing on security implications
- Merge to trunk upon approval

3. Automated Cleanup

- Merged branches automatically deleted
- Stale branch identification after 7 days
- Local branch pruning recommendations

Quality Gates

Pre-Merge Requirements

1. **Security Scan**: No high/critical vulnerabilities
2. **Plugin Tests**: All security plugin tests pass
3. **Fuzzing Tests**: Basic fuzzing validation
4. **Code Review**: Security-focused peer review
5. **Documentation**: Security implications documented

Automated Checks

- Dependency vulnerability scanning
- Static Application Security Testing (SAST)
- Security plugin compatibility tests
- Performance impact assessment

Release Management

Release Branches

- Created from trunk for version releases
- **Naming**: `release/v[major].[minor].[patch]`

- **Purpose**: Final testing and production deployment
- **Lifecycle**: 24-48 hours maximum

Hotfix Process

- Critical security fixes bypass normal workflow
- Direct commits to trunk with immediate deployment
- Post-deployment validation and documentation

Branch Protection Rules

Trunk Protection (`safe-improvements`)

- Require pull request reviews (minimum 1)
- Require status checks to pass
- Require branches to be up to date
- Restrict pushes to administrators only
- Require signed commits for security

Feature Branch Guidelines

- Regular rebasing against trunk
- Atomic commits with clear security context
- Comprehensive testing before PR creation

Migration Strategy

Phase 1: Current State (Complete)

- Merged `roadmap-exec` into `safe-improvements`
- Established trunk-based workflow
- Updated documentation for security focus

Phase 2: Integration (In Progress)

- Integrate `feature/community-portal`
- Integrate `feature/marketing-automation`
- Archive legacy branches

Phase 3: Optimization

- Implement advanced automated testing
- Enhanced security scanning pipeline
- Performance monitoring integration

Best Practices

For Security Plugin Development

1. **Isolated Testing**: Test security plugins in sandboxed environments
2. **Vulnerability Disclosure**: Follow responsible disclosure for discovered issues
3. **Documentation**: Comprehensive security impact documentation
4. **Validation**: Multi-stage validation for security tools

For Contributors

1. **Small Commits**: Atomic changes for easier security review
2. **Clear Messages**: Commit messages indicating security implications
3. **Regular Sync**: Daily synchronization with trunk
4. **Testing**: Local security testing before push

Monitoring and Metrics

Development Velocity

- Average feature branch lifetime
- Integration frequency
- Time to production deployment

Security Metrics

- Vulnerability detection rate
- False positive reduction
- Security plugin effectiveness
- Code quality improvements

Tools and Automation

CI/CD Pipeline

- GitHub Actions for automated testing
- Security scanning integration
- Automated branch cleanup
- Performance monitoring

Local Development

- Pre-commit hooks for security checks
- Local fuzzing test capabilities
- Automated dependency updates

This branching strategy ensures rapid development of security auditing capabilities while maintaining the highest standards of code quality and security for the Audityzer platform.