

GitKraken CLI & GitLens Developer Workflow Guide

This comprehensive guide shows you how to leverage GitKraken CLI and GitLens to dramatically improve your Git workflow efficiency, with special focus on interactive rebasing and graph visualization.

What You'll Learn

This documentation covers essential developer workflow optimizations:

- **GitKraken CLI Setup** - Command-line Git power tools
 - **GitLens Extension** - VS Code Git integration and visualization
 - **Interactive Rebase Mastery** - Clean up commit history like a pro
 - **Graph Visualization** - Understand complex branch relationships
 - **Advanced Workflows** - Real-world development scenarios
 - **Best Practices** - Pro tips for optimal Git workflows
-

GitKraken CLI Setup

Installation

macOS (Homebrew):

```
brew install gitkraken-cli
```

Manual Installation:

```
# Download and install from GitKraken website  
# Or use npm  
npm install -g @gitkraken/gk-cli
```

Authentication

```
# Login to GitKraken  
gk auth login  
  
# Verify authentication  
gk auth status
```

Basic Configuration

```
# Set up GitKraken CLI with your repositories
gk repo add /path/to/your/repo

# List configured repositories
gk repo list
```

GitLens Extension

Installation in VS Code

1. Open VS Code
2. Go to Extensions (Ctrl+Shift+X)
3. Search for "GitLens"
4. Install "GitLens — Git supercharged" by GitKraken

Key Features to Enable

- **Blame annotations** - See who changed what line
- **Code lens** - Inline Git information
- **File history** - Track file changes over time
- **Branch comparison** - Compare changes between branches

Interactive Rebase Workflows

GitKraken CLI Interactive Rebase

Start an interactive rebase:

```
# Rebase last 3 commits
gk rebase -i HEAD~3

# Rebase from a specific commit
gk rebase -i <commit-hash>

# Rebase onto a different branch
gk rebase -i main
```

Common rebase operations:

- **pick** - Keep the commit as-is
- **edit** - Pause to modify the commit
- **squash** - Combine with previous commit
- **drop** - Remove the commit entirely
- **reword** - Change commit message

GitLens Interactive Rebase

Using GitLens in VS Code:

1. Open Interactive Rebase View:

- Command Palette (**Cmd+Shift+P**)
- Type "GitLens: Open Interactive Rebase Editor"

2. Visual Rebase Operations:

- Drag and drop commits to reorder
- Right-click for context menu options
- Edit commit messages inline

Example Workflow:

```
# In terminal, start rebase
git rebase -i HEAD~5

# VS Code will automatically open GitLens rebase editor
# - Reorder commits by dragging
# - Change actions using dropdown menus
# - Edit messages inline
# - Save and close to apply changes
```

Advanced Rebase Scenarios

Splitting a commit:

```
# Mark commit for edit in rebase
# When paused at the commit:
git reset HEAD~1
git add -p file1.js # Stage parts of changes
git commit -m "First part of changes"
git add .
git commit -m "Second part of changes"
git rebase --continue
```

Squashing multiple commits:

```
# In GitKraken CLI
gk rebase -i HEAD~4
# Change 'pick' to 'squash' for commits to combine
# Edit the combined commit message
```

Graph Views & Visualization

GitKraken CLI Graph Commands

View repository graph:

```
# ASCII graph in terminal
gk log --graph --oneline

# Detailed graph with dates
gk log --graph --pretty=format:"%h %ad %s" --date=short

# Show all branches
gk log --graph --all --oneline
```

Branch visualization:

```
# Show branch relationships
gk branch --graph

# Compare branches visually
gk diff branch1..branch2 --stat
```

GitLens Graph Features

Commits Graph View:

1. Open Command Palette (**Cmd+Shift+P**)
2. Type "GitLens: Show Commits View"
3. Features available:
 - Interactive timeline
 - Branch visualization
 - Commit details on hover
 - Search and filter commits

Repository Graph:

1. Command Palette → "GitLens: Show Graph"
2. Benefits:
 - Visual branch merging
 - Commit relationships
 - Easy navigation between commits
 - Branch comparison tools

Customizing Graph Views

GitLens Settings for Better Graphs:

```
{  
  "gitlens.graph.layout": "editor",  
  "gitlens.graph.showGhostRefsOnRowHover": true,  
  "gitlens.graph.pullRequests.enabled": true,  
  "gitlens.graph.minimap.enabled": true,  
  "gitlens.graph.scrollRowPadding": 3  
}
```

Advanced Workflows

Feature Branch Workflow with GitKraken CLI

Creating and managing feature branches:

```
# Create feature branch  
gk checkout -b feature/new-authentication  
  
# Work on feature...  
git add .  
git commit -m "Add OAuth integration"  
  
# Interactive rebase to clean up history  
gk rebase -i main  
  
# Push and create PR  
gk push --set-upstream origin feature/new-authentication  
gk pr create --title "Add OAuth Authentication" --body "Implements OAuth2  
flow"
```

Hotfix Workflow

Emergency fixes with clean history:

```
# Create hotfix branch from main  
gk checkout main  
gk pull  
gk checkout -b hotfix/critical-security-fix  
  
# Make fix  
git add security-patch.js  
git commit -m "Fix: Resolve XSS vulnerability in user input"  
  
# Clean rebase if needed  
gk rebase -i main  
  
# Merge back to main  
gk checkout main
```

```
gk merge hotfix/critical-security-fix
gk push
```

GitLens Blame and History Investigation

Finding the root cause of issues:

1. Blame View:

- Right-click any line → "Toggle Line Blame"
- See who changed what and when
- Click commit hash for full details

2. File History:

- Command Palette → "GitLens: Show File History"
- See all changes to current file
- Compare different versions

3. Line History:

- Right-click line → "Show Line History"
- Track how a specific line evolved

Best Practices






Interactive Rebase Best Practices

Before rebasing:

```
# Always work on a backup branch
gk checkout -b backup-branch

# Ensure working directory is clean
git status
```

Rebase rules:

-  Never rebase public/shared branches
-  Use rebase for private feature branches
-  Squash related commits together
-  Keep commit messages descriptive
-  Don't rebase if others are working on the branch

Graph View Optimization

Settings for better performance:

```
{
  "gitlens.graph.pageItemLimit": 2500,
  "gitlens.graph.searchItemLimit": 100,
  "gitlens.views.commits.showBranchComparison": "working",
  "gitlens.views.repositories.showBranchComparison": "working"
}
```

Workflow Automation

GitKraken CLI aliases:

```
# Add to your shell profile
alias gks="gk status"
alias gkl="gk log --graph --oneline"
alias gkp="gk pull --rebase"
alias gkpu="gk push --set-upstream origin"
```

VS Code keybindings for GitLens:

```
{
  "key": "cmd+shift+g h",
  "command": "gitlens.showFileHistory"
},
{
  "key": "cmd+shift+g b",
  "command": "gitlens.toggleFileBlame"
},
{
  "key": "cmd+shift+g g",
  "command": "gitlens.showGraphPage"
}
```

Practical Examples

Scenario 1: Cleaning Up Feature Branch History

Problem: Feature branch has messy commit history with WIP commits

Solution:

```
# Start interactive rebase
gk rebase -i main

# In GitLens rebase editor:
# 1. Squash WIP commits with meaningful commits
```

```
# 2. Reword commit messages to be descriptive
# 3. Drop any debugging commits
# 4. Reorder commits logically
```

Scenario 2: Finding When a Bug Was Introduced

Problem: Bug exists but unsure when it was introduced

Solution using GitLens:

1. Open the problematic file
2. Use "Toggle Line Blame" on the buggy line
3. Click the commit hash to see full change
4. Use "Show File History" to see evolution
5. Use git bisect if needed:

```
git bisect start
git bisect bad HEAD
git bisect good <last-known-good-commit>
```

Scenario 3: Reviewing Team Changes

Using GitLens for code review:

1. Open "Repositories" view in GitLens
2. Expand "Branches" → "Remotes"
3. Right-click colleague's branch → "Compare with Working Tree"
4. Review changes in GitLens comparison view
5. Use graph view to understand branch relationships

Pro Tips

GitKraken CLI Power Commands

```
# Quick commit with message
gk commit -am "Quick fix for navigation bug"

# Stash with description
gk stash push -m "Work in progress on user auth"

# Cherry-pick with conflict resolution
gk cherry-pick <commit-hash> --strategy=recursive -X theirs

# Interactive add for precise staging
gk add -i
```


GitLens Hidden Gems

- **Compare Working Tree:** See all uncommitted changes at once
- **Search Commits:** Find commits by message, author, or hash
- **Worktrees:** Manage multiple working directories
- **Autolinks:** Automatic linking to issues and PRs

Integration Tips

Combine GitKraken CLI with GitLens:

1. Use CLI for heavy Git operations (rebase, merge)
2. Use GitLens for exploration and investigation
3. Use CLI scripts for automation
4. Use GitLens for visual confirmation

Summary

By combining GitKraken CLI's powerful command-line tools with GitLens's rich VS Code integration, you can:

- **Speed up development** with efficient Git operations
- **Debug faster** using blame and history views
- **Visualize complex Git graphs** for better understanding
- **Perform safe interactive rebases** with visual feedback
- **Collaborate better** with clean commit history

The key is to use CLI for bulk operations and GitLens for detailed investigation and visualization. This combination provides both power and usability for modern Git workflows.

Happy coding!