**Patch Creation Script - Complete Usage Guide**

**1. Prerequisites for Setup**

**System Requirements**

# Required tools (check if installed)

git --version # Git version control

bash --version # Bash shell (usually pre-installed)

diffstat --version # Optional: for patch statistics

# Install missing tools (Ubuntu/Debian)

sudo apt-get update

sudo apt-get install git diffstat

# Install missing tools (macOS)

brew install git diffstat

# Install missing tools (CentOS/RHEL)

sudo yum install git diffstat

**Git Repository Setup**

# Initialize or clone your SDK repository

git clone https://github.com/yourcompany/sdk-repo.git

cd sdk-repo

# Or initialize new repository

git init

git remote add origin https://github.com/yourcompany/sdk-repo.git

# Configure git (if not done already)

git config --global user.name "Your Name"

git config --global user.email "your.email@company.com"

**Script Setup**

# Download and setup the script

curl -o create\_patch.sh https://your-script-location/create\_patch.sh

chmod +x create\_patch.sh

# Or create manually and copy the script content

touch create\_patch.sh

chmod +x create\_patch.sh

# Copy the script content into create\_patch.sh

**2. Sample Usage Examples with Real Input/Output**

**Example 1: Creating Patch from Feature Branch (SDK New Feature)**

**Scenario**: You've developed a new authentication feature in SDK

$ ./create\_patch.sh

**Output:**

=== Automated Patch Creation Script ===

[INFO] Repository: my-sdk

[INFO] Current branch: feature/oauth-integration

=== Select Patch Creation Method ===

1) Create patch from specific commits

2) Create patch from branch comparison

3) Create patch from last N commits

4) Create patch from uncommitted changes

Choose method (1-4): 2

=== Create Patch from Branch Comparison ===

Available branches:

\* feature/oauth-integration

main

remotes/origin/main

remotes/origin/develop

Enter base branch (default: main): main

Enter feature branch (default: feature/oauth-integration):

[INFO] Creating patch comparing feature/oauth-integration with main...

[INFO] Patch created: patch-feature/oauth-integration-vs-main.patch

Add patch description? (y/n): y

Enter patch description:

OAuth 2.0 integration for SDK authentication - includes token management, refresh logic, and error handling

[INFO] Metadata added to patch.

=== Patch Summary ===

Patch file: patch-feature/oauth-integration-vs-main.patch

Size: 15K

Lines: 487

Diffstat:

src/auth/oauth.js | 156 ++++++++++++++++++++++++++++++++++++++++++++

src/auth/token-manager.js | 89 ++++++++++++++++++++++++++

tests/auth/oauth.test.js | 134 +++++++++++++++++++++++++++++++++++++

README.md | 12 +++-

package.json | 3 +-

5 files changed, 392 insertions(+), 2 deletions(-)

View patch content? (y/n): n

[INFO] Patch creation completed successfully!

Open directory containing the patch? (y/n): n

**Example 2: Creating Patch from Last N Commits (Bug Fix)**

**Scenario**: Fixed critical bugs in last 3 commits for hotfix release

$ ./create\_patch.sh

**Output:**

=== Automated Patch Creation Script ===

[INFO] Repository: my-sdk

[INFO] Current branch: hotfix/critical-fixes

=== Select Patch Creation Method ===

1) Create patch from specific commits

2) Create patch from branch comparison

3) Create patch from last N commits

4) Create patch from uncommitted changes

Choose method (1-4): 3

=== Create Patch from Last N Commits ===

Recent commits:

a1b2c3d Fix memory leak in HTTP client

e4f5g6h Resolve null pointer exception in parser

h7i8j9k Update error handling for network timeouts

k9l0m1n Add logging for debugging API calls

n2o3p4q Initial commit for feature

Enter number of commits to include: 3

[INFO] Creating patch from last 3 commits...

[INFO] Patch created: patch-last-3-commits.patch

Add patch description? (y/n): y

Enter patch description:

Critical hotfix patch containing memory leak fix, null pointer resolution, and improved error handling

[INFO] Metadata added to patch.

=== Patch Summary ===

Patch file: patch-last-3-commits.patch

Size: 8.2K

Lines: 256

View patch content? (y/n): y

**Example 3: Creating Patch from Uncommitted Changes (Development)**

**Scenario**: Working on SDK performance improvements, want to create patch before committing

$ ./create\_patch.sh

**Output:**

=== Automated Patch Creation Script ===

[INFO] Repository: my-sdk

[INFO] Current branch: develop

[WARNING] You have uncommitted changes.

Uncommitted files:

M src/core/client.js

M src/utils/performance.js

A src/benchmarks/perf-test.js

?? temp-notes.txt

Do you want to commit these changes first? (y/n): n

Continue without committing? (y/n): y

=== Select Patch Creation Method ===

1) Create patch from specific commits

2) Create patch from branch comparison

3) Create patch from last N commits

4) Create patch from uncommitted changes

Choose method (1-4): 4

=== Create Patch from Uncommitted Changes ===

Uncommitted changes:

M src/core/client.js

M src/utils/performance.js

A src/benchmarks/perf-test.js

?? temp-notes.txt

[INFO] Creating patch from uncommitted changes...

[INFO] Patch created: patch-uncommitted-20250625-143022.patch

Add patch description? (y/n): y

Enter patch description:

Performance optimization patch - improved HTTP client efficiency and added benchmark testing

[INFO] Metadata added to patch.

=== Patch Summary ===

Patch file: patch-uncommitted-20250625-143022.patch

Size: 5.4K

Lines: 178

[INFO] Patch creation completed successfully!

**Example 4: Creating Patch from Specific Commits (Release Preparation)**

**Scenario**: Creating patch for specific features to include in release

$ ./create\_patch.sh

**Input Values:**

* Method: 1 (specific commits)
* Starting commit: abc123f (start of feature development)
* Ending commit: def456a (end of feature development)
* Description: "Feature release patch v2.1.0 - includes new API endpoints and improved documentation"

**3. Values to Fill in Scripts**

**Dynamic Input Fields:**

| **Field** | **Purpose** | **Example Values** | **Notes** |
| --- | --- | --- | --- |
| **Method Selection** | Choose patch creation type | 1, 2, 3, or 4 | Based on your workflow |
| **Commit Hashes** | Specify commit range | abc123f, def456a | Use git log --oneline to find |
| **Branch Names** | Source/target branches | main, develop, feature/api-v2 | Use git branch -a to list |
| **Number of Commits** | Last N commits | 1, 3, 5, 10 | Based on logical grouping |
| **Patch Description** | Metadata for patch | "OAuth integration feature" | Descriptive, include ticket numbers |
| **Commit Message** | If committing first | "Fix: resolve API timeout issues" | Follow conventional commits |

**SDK Development Context Values:**

# Feature Development

Base Branch: "main" or "develop"

Feature Branch: "feature/new-sdk-method"

Description: "Add new payment processing methods to SDK v3.2"

# Bug Fixes

Number of Commits: 2-3 (targeted fixes)

Description: "Hotfix: Critical security patch for API authentication"

# Release Preparation

Starting Commit: "v2.0.0" (tag)

Ending Commit: "HEAD"

Description: "Release candidate v2.1.0 - all features and fixes"

**4. How to Confirm Patch Creation Properly**

**A. Immediate Validation**

# Check if patch file exists and has content

ls -la \*.patch

-rw-r--r-- 1 user user 15234 Jun 25 14:30 patch-feature-oauth-vs-main.patch

# Verify patch is not empty

wc -l patch-feature-oauth-vs-main.patch

487 patch-feature-oauth-vs-main.patch

# Check patch format is valid

head -20 patch-feature-oauth-vs-main.patch

**B. Content Validation**

# Use git to validate patch format

git apply --check patch-feature-oauth-vs-main.patch

# No output = patch is valid

# Check what files would be affected

git apply --stat patch-feature-oauth-vs-main.patch

# Preview changes without applying

git apply --check --verbose patch-feature-oauth-vs-main.patch

**C. Test Application (Recommended)**

# Create test branch for validation

git checkout -b test-patch-validation

# Apply the patch

git apply patch-feature-oauth-vs-main.patch

# Verify changes

git status

git diff --cached # if patch was applied with --index

# Test SDK functionality

npm test # or your test command

npm run build # ensure it builds

# Clean up test branch

git checkout main

git branch -D test-patch-validation

**D. Advanced Validation for SDK Development**

# Check patch statistics

diffstat patch-feature-oauth-vs-main.patch

# Verify no sensitive data in patch

grep -i "password\|secret\|key\|token" patch-feature-oauth-vs-main.patch

# Check file permissions

grep "^old mode\|^new mode" patch-feature-oauth-vs-main.patch

# Validate patch can be emailed (no binary data)

file patch-feature-oauth-vs-main.patch

# Should output: ASCII text

**5. Efficient Usage for Dynamic SDK Development**

**A. Integration with CI/CD Pipeline**

# Automated patch creation in CI

./create\_patch.sh --method=2 --base=main --feature=$CI\_BRANCH --desc="Auto-generated patch for $CI\_COMMIT\_MESSAGE"

# Store patches as artifacts

cp \*.patch $CI\_ARTIFACTS\_DIR/

**B. Code Review Workflow**

# Create patch for code review

./create\_patch.sh # Method 4 for uncommitted changes

# Email patch to reviewers or attach to PR

# Apply reviewer suggestions

git apply reviewer-suggestions.patch

**C. Release Management**

# Create release patches

./create\_patch.sh # Method 1, from last release tag to HEAD

# Archive patches with release notes

# Hotfix distribution

./create\_patch.sh # Method 3, last 2-3 critical commits

# Distribute to customers for immediate application

**D. Multi-Repository SDK Development**

# Core SDK changes

cd sdk-core && ./create\_patch.sh

# Language bindings

cd ../sdk-python && ./create\_patch.sh

cd ../sdk-javascript && ./create\_patch.sh

# Combine related patches for coordinated releases

**E. Error Recovery and Debugging**

# Before major changes

./create\_patch.sh # Create backup patch

# After issues discovered

git apply backup-patch.patch # Restore previous state

# Incremental development

./create\_patch.sh # Create small, focused patches for each feature

**6. Best Practices for SDK Development**

1. **Semantic Patch Naming**: Include version numbers, feature names, and dates
2. **Descriptive Metadata**: Always add meaningful descriptions with ticket numbers
3. **Small, Focused Patches**: Create patches for logical units of work
4. **Test Before Distribution**: Always validate patches in clean environment
5. **Version Control**: Track patches in separate branch or documentation
6. **Security Review**: Check patches for sensitive information before sharing
7. **Backward Compatibility**: Note any breaking changes in patch descriptions

This comprehensive approach ensures reliable patch creation and distribution for dynamic SDK development workflows.