How commit policies enable end-to-end traceability and faster code reviews

Code commit rules your developer team will follow
Who is Midori?

Experience
10+ years in the Atlassian Ecosystem

Top Vendor
(earlier was called “Atlassian Verified”) Reliable maintenance, guaranteed support and top-notch docs

Powering 4000+ customers
Bank of America, BMW, Northrop Grumman, Lenovo, etc.

What is Better Commit Policy?

On the market since 2015

The must-have app for every developer team

“Set it and forget it” commit rules
Code change, tag and branch verification for Git, Bitbucket, GitHub, GitLab, Subversion & Mercurial
How commit policies enable end-to-end traceability and faster code reviews

- Why do you need commit policies?
- How do commit policies work?
- Defining commit policies
- Installing commit policies
- Commit policies in any environment
- Introducing commit policies to your team
How do commit policies work?

Why do you need commit policies?

How do commit policies work?

Defining commit policies

Installing commit policies

Commit policies in any environment

Introducing commit policies to your team

end-to-end traceability and faster code reviews
Why do you need commit policies?

Key reasons for using commit policies

- Code repositories need to follow processes/regulations
  - Commit policies ensure compliance in industries like:
    - Avionics
    - Automotive
    - Financial
    - Defence
    - Medical
    - Pharmaceutical
    - (Other safety-critical)

- Changes to source code need to be connected to requirements
  - The relation needs must be traceable between code changes and:
    - Requirements
    - Test executions
    - Bug reports
    - Version releases
    - User stories

- A well-controlled code base makes reviews and audits easy
  - Commit Policies help answer questions like:
    - Why was this changed?
    - Was it tested?
    - Is this within the scope of the current release?
    - Who approved this merge?
    - Is there a programmatic test for this story?
Why do you need commit policies?
What are the risks of an uncontrolled repository?
How commit policies enable end-to-end traceability and faster code reviews

Why do you need commit policies?
How do commit policies work?
Defining commit policies
Installing commit policies
Commit policies in any environment
Introducing commit policies to your team
How do commit policies work?

The commit verification process

Developer machine

Commit rejected with descriptive message

Commit Policy hook script

Successful commit

Target repository

Better Commit Policy for Jira

Changes sent to Jira for verification

Rejected changes

Accepted changes

Commited changes

Changes sent to Jira for verification

Jira
How do commit policies work?
The KILLER feature you won’t find elsewhere: local commit verification with Git

Usual approach:
Verify remotely when pushing

 commit, commit, commit
Local repository
push
Remote repository
all rejected! too late!

The Midori approach:
Verify locally when committing

 commit
Local repository
push
Remote repository
...will always succeed!

Okay, let me just quickly re-commit the very last one.

Arghhh!
I must use interactive rebasing to fix all those...

level of frustration when rejected

one rejected!
How commit policies enable end-to-end traceability and faster code reviews

- Why do you need commit policies?
- How do commit policies work?
- Defining commit policies
- Installing commit policies
- Commit policies in any environment
- Introducing commit policies to your team
**Defining commit policies**
Commit policy = rules for who can change what under what conditions in repositories

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Application design guidelines</td>
<td>This policy is for the application (Android and iOS) design guidelines and documentation</td>
<td>Apply to a repository</td>
</tr>
<tr>
<td>7</td>
<td>Crypto Exchange Policy</td>
<td>Commits accepted only against unresolved issues in the CEP project.</td>
<td>Apply to a repository</td>
</tr>
<tr>
<td>12</td>
<td>Current CLD Sprint</td>
<td>Commits only against the “Cloud Hosting” Sprint will be accepted.</td>
<td>Apply to a repository</td>
</tr>
<tr>
<td>20</td>
<td>EVM - Solidity</td>
<td>This policy ensures that only solidity files are committed and changed</td>
<td>Apply to a repository</td>
</tr>
<tr>
<td>9</td>
<td>FREEZE DISABLED</td>
<td>Commit policy to freeze the repository by rejecting all changes.</td>
<td>Apply to a repository</td>
</tr>
<tr>
<td>8</td>
<td>Image files only</td>
<td>Only image files in the “.jpg” “jpeg” “.svg” “.png” “.gif” formats are accepted.</td>
<td>Apply to a repository</td>
</tr>
<tr>
<td>19</td>
<td>Sounds and music</td>
<td>This rule only allows audio files to be changed.</td>
<td>Apply to a repository</td>
</tr>
<tr>
<td>1</td>
<td>TOMCAT</td>
<td>Commit policy for Apache Tomcat, an open-source server and servlet container.</td>
<td>Apply to a repository</td>
</tr>
</tbody>
</table>
# Defining commit policies

## Editing the details of commit policies

![Jira Software interface showing commit policy settings](image)

## Edit Commit Policy

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy ID</td>
<td>1</td>
</tr>
<tr>
<td>Name</td>
<td>TOMCAT</td>
</tr>
<tr>
<td>Description</td>
<td>Commit policy for Apache Tomcat, an open-source server and servlet container.</td>
</tr>
<tr>
<td>Rejection message</td>
<td>All change sets pushed into this repository must conform to our coding policy. See the manual page for details: <a href="https://tomcat.apache.org/tomcat-7.0-doc/appdev/source.html">https://tomcat.apache.org/tomcat-7.0-doc/appdev/source.html</a></td>
</tr>
</tbody>
</table>

**Tips:** If your Version Control System or Operating System console fails with international characters, consider using English-only characters.

### Options

- Accept the commits that **already exist** in the repository (on another branch) without verification.
- Accept the merge commits without verification (for Git and Mercurial).

### Tag rules

The tag must satisfy **all** of these rules:
## Defining commit policies

Adding tag, branch and commit rules to commit policies

<table>
<thead>
<tr>
<th>Tag rules</th>
<th>The tag must satisfy <strong>all</strong> of these rules:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No rules added yet.</td>
</tr>
</tbody>
</table>

+ Add rule

<table>
<thead>
<tr>
<th>Branch rules</th>
<th>The branch must satisfy <strong>all</strong> of these rules:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No rules added yet.</td>
</tr>
</tbody>
</table>

+ Add rule

<table>
<thead>
<tr>
<th>Commit rules</th>
<th>The commit must satisfy <strong>all</strong> of these rules:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No rules added yet.</td>
</tr>
</tbody>
</table>

+ Add rule

[Save] [Cancel]
Defining commit policies
Customizing the conditions within a tag rule
## Defining commit policies

Customizing the conditions within a branch rule

<table>
<thead>
<tr>
<th>Rule 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>The branch must satisfy <strong>all</strong> of these rules:</td>
</tr>
</tbody>
</table>

**Condition 2.1**

- **Branch name must match a pattern**
  - [ ] Apply this rule only if the branch name matches
  - [ ] Additional rejection message for this rule
  - The commit must satisfy **all** of these conditions:

  - **Branch name must match a pattern**
    - (feature|bugfix)/(BBB)-\d+\.\S+
    - [ ] Exactly one issue
      - [ ] in
      - [ ] status = "In Progress"
      - [ ] Allow and ignore the issue keys that don’t match the JQL
      - Leave the JQL blank to accept any existing issue key
      - Non-existing issue keys are strictly rejected. Other references with similar syntax can be ignored by re-configuring the issue key pattern.
      - [ ] Examples

**Condition 2.2**

- **Branch name must contain issue keys from a JQL query**
  - [ ] Add condition
  - [ ] Add rule
Defining commit policies
Customizing the conditions for commit messages

Commit rules The commit must satisfy all of these rules:

Rule 1
- Limit scope
  - Apply this rule only if the branch name matches
  - Apply this rule only for the files whose path matches

Additional rejection message for this rule
Enter the message shown for the committer when his commit is rejected by this specific rule (consider including an example that would be accepted)

TIP If your Version Control System or Operating System console fails with international characters, consider using English-only characters.

The commit must satisfy all of these conditions:

Condition 1.1 Commit message must match a pattern
- Glob

+ Add condition
Defining commit policies
Customizing the conditions for commit messages

Condition 2.1

Commit message must match a pattern

\((\S+|\s*)\[10,\] \)

Examples
- Require at least 10 character long commit messages (excluding whitespace):
  - \((\S+|\s*)\[10,\] \)
- Require a Jira issue key from the FOO or BAR projects in the start of the commit message:
  - \((FOO|BAR) - \d+.*\)
- Enforce the 50/72 rule:
  - \5.\{0,49\} ((\r\n|\n) |\r\n|\n) .\{3,72\} ((\r\n|\n) |\r\n|\n) .\{1,72\} ((\r\n|\n) |\r\n|\n) \)
Defining commit policies
Customizing the conditions for commit messages

**Condition 2.2**

Commit message must contain issue keys from a JQL query

- **exactly one issue** in **project = "Beem for Business"** and **status = "In Progress"** and **assignee = current**

- **Allow and ignore the issue keys that don’t match the JQL**

Leave the JQL blank to accept any existing issue key. Non-existing issue keys are strictly rejected. Other references with similar syntax can be ignored by re-configuring the issue key pattern.

**Examples**

- Issues from project FOO:
  
  ```jql
  project = FOO
  ```

- Issues from project FOO and assigned to the committer (for Git):
  
  ```jql
  project = FOO and assignee = currentUser()
  ```

- Issues from project FOO and assigned to the committer (for Subversion):
  
  ```jql
  project = FOO and assignee = "${committer.userName}"
  ```

- Issues from project FOO and assigned to the committer or to anyone if the committer is the project lead (for Git):
  
  ```jql
  project = FOO and (assignee = currentUser() or project in projectsLeadByUser())
  ```

- Issues from project FOO and assigned to the committer or to anyone if the committer is the project lead (for Subversion):
  
  ```jql
  project = FOO and (assignee = "${committer.userName}" or project in projectsLeadByUser("${committer.userName}"))
  ```

- User stories in the current sprint from project FOO:
  
  ```jql
  project = FOO and issuetype = Story and sprint in openSprints()
  ```
Defining commit policies
Customizing the conditions for changed files

**Condition 2.3**

**Changed paths (files) must match a pattern**

Using a glob pattern: `{obj, tmp, class}`

**Examples**

- Avoid checking in `obj`, `tmp`, `class` files:
  - `COPY GLOB` `!(*.obj, *.tmp, *.class)`

- Allow files with restricted types in a directory, e.g., `~/images` must contain image files:
  - `COPY GLOB` `~/images/*.jpg,*.png,*.gif`

- Lock a file, i.e., no changes are allowed on a file:
  - `COPY GLOB` `!(*.LICENSE.txt, LICENSE.txt)`

- Lock a directory, i.e., no changes are allowed on a directory and its descendants:
  - `COPY GLOB` `!(*production-config/)`

- Lock (freeze) the whole repository, i.e., to reject all changes in a repository:
  - `COPY REGEX` `(!.*)`

- Enforce naming conventions on Java files (tip: implement any naming convention similarly):
  - `COPY REGEX` `((\./\?)((A-Z)([a-z0-9-]+)\.(java))(((?<!\.java)\/(?!\-))/)+)`

- Protect the Subversion repository structure (`/trunk`, `/branches`, and `/tags`):
  - `COPY REGEX` `([^/]*(trunk|branches|tags)/.*)`
Defining commit policies
Customizing the conditions for changed files

**Condition 2.4**

**Changed paths (files) must contain issue keys from a JQL query**

- exactly one issue in project = "Fitband App" and assignee = "$committer.userName"

- Allow and ignore the issue keys that don’t match the JQL

Leave the JQL blank to accept any existing issue key.
Non-existing issue keys are strictly rejected. Other references with similar syntax can be ignored by re-configuring the issue key pattern.

**Examples**

**Issues from project FOO:**

- `COPY JQL project = FOO`

**Issues from project FOO and assigned to the committer (for Git):**

- `COPY JQL project = FOO and assignee = currentUser()`

**Issues from project FOO and assigned to the committer (for Subversion):**

- `COPY JQL project = FOO and assignee = "$committer.userName"`

**Issues from project FOO and assigned to the committer or to anyone if the committer is the project lead (for Git):**

- `COPY JQL project = FOO and (assignee = currentUser() or project in projectsLeadByUser())`

**Issues from project FOO and assigned to the committer or to anyone if the committer is the project lead (for Subversion):**

- `COPY JQL project = FOO and (assignee = "$committer.userName" or project in projectsLeadByUser("$committer.userName"))`

**User stories in the current sprint from project FOO:**

- `COPY JQL project = FOO and issuetype = Story and sprint in openSprints()`
Defining commit policies
Customizing the conditions for committer users

Condition 2.5

**Committer must have a valid JIRA account**

Committer's **email** must identify a valid Jira user account

☑️ User must be member in any of these Jira groups: [jira-software-users](#)

e.g. the 'jira-developers' group only

+ Add condition -
Defining commit policies

Customizing the conditions for committer users

**Committer attribute must match a pattern**

- Committer's email must match the pattern: *@midori-global.com*

Examples:
- Accept from a specific email domain:
  - COPY GLOB: *@gmail.com*
- Accept from a list of usernames:
  - COPY GLOB: *@john,jack,com*
- Accept nothing (freeze the repository):
  - COPY REGEX: *(?!.*)*
How commit policies enable end-to-end traceability and faster code reviews

- Why do you need commit policies?
- How do commit policies work?
- Defining commit policies
- Installing commit policies
- Commit policies in any environment
- Introducing commit policies to your team
Installing commit policies

Installing and applying commit policies to repositories
Installing commit policies
Installing and applying commit policies to repositories

Hook Script Wizard

I want to apply the **TOMCAT** commit policy to a **Git** repository so that the commits are verified hosted on **Linux, Mac OS X (All U*X variants)**.

Select the OS of the server which hosts the Git repositories and hooks.
Installing commit policies

Installing and applying commit policies to repositories
Installing commit policies
Installing and applying commit policies to repositories
Installing commit policies
Installing and applying commit policies to repositories

Commit Policies

Hook Script Wizard

Install Python for the hook scripts

If this is the very first hook script for this Git server, make sure that the software required for executing the hook scripts are installed to the server.
You only need to execute this step once, by clicking the link below and following the instructions. If you already completed this step before, you can safely skip it now.
Show instructions

Back | I have installed Python

Commit policy for Apache Tomcat, an open-source server and servlet container.
Installing commit policies
Installing and applying commit policies to repositories

Hook Script Wizard

Download the hook script package
Click the button, then a ZIP archive with all necessary files will be downloaded to your browser.

Download
Installing commit policies

Installing and applying commit policies to repositories

Hook Script Wizard

Install the hook script

1. Unpack the ZIP file to the location where Git expects it

You will need write access to the filesystem where Git stores the repositories and hooks. Ask for the help of your system administrator when in doubt.

The destination directory is here by default:

```bash
<CENTRAL_GIT_REPO>/git/hooks
```

...or here for a Git bare repository (if you don't know what it is, then use the first location).

Back I have installed the hook script
## Installing commit policies

Installing and applying commit policies to repositories

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>applypatch-msg.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>commit-msg.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jcp-git-local-windows-1-hook-scripts.zip</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>post-update.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pre-apply-patch.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pre-commit.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prepare-commit-msg.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pre-push.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pre-rebase.sample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update-sample</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Select a Destination and Extract Files

Files will be extracted to this folder:

C:\repo\local\git\hooks\jcp-git-local-windows-1-hook-scripts

- [ ] Show extracted files when complete

![Extracting files dialog box](image)
Installing commit policies
Installing and applying commit policies to repositories

Hook Script Wizard

Test your configuration

1. Try to commit something that should be rejected
   Ex: if your policy requires at least one issue key, then make a commit with the message "No issue key here". Check if it is rejected.

2. Try to commit something that should be accepted
   Ex: if your policy requires at least one issue key, include one in the commit message: "Fix for FOO-17". Check if it is accepted.
Installing commit policies

Testing your commit rule

Make a change, and commit it without linking to a Jira issue.

Rejected (immediately, not when pushing to the server).

Commit again including a Jira issue in the commit message.

Rejected, as the issue key is not an unresolved in-progress issue.

Commit again including a Jira issue that meets the requirements.

Accepted.
Installing commit policies

Testing your tag rule

Create the tag "v1.2" and push it.

Rejected as its format is not allowed.

Rename it to "1.2.0" (a valid semantic version name), and push again.

Accepted.
Installing commit policies

Testing your branch rule
Installing commit policies

Installing and applying commit policies to repositories
How commit policies enable end-to-end traceability and faster code reviews

- Why do you need commit policies?
- How do commit policies work?
- Defining commit policies
- Installing commit policies
- Commit policies in any environment
- Introducing commit policies to your team
Installing commit policies
Supported Version Control Systems

Any custom VCS or environment via REST API
Installing commit policies

Supported clients & IDE integrations
REST API
Integrate with custom VCSs and custom environments

New REST end-points are added to the standard Jira REST API!

For commit verification:
/commit-policy/{policy-id}/verification

For listing existing commit policies:
/commit-policy

For generating hook scripts:
/hook-script/{vcs}/{os}/{policy-id}

– Learn more:
http://www.midori-global.com/products/better-commit-policy-for-jira/documentation/rest-api
How commit policies enable end-to-end traceability and faster code reviews

- Why do you need commit policies?
- How do commit policies work?
- Defining commit policies
- Installing commit policies
- Commit policies in any environment
- Introducing commit policies to your team
Introducing commit policies to your team

Making your job easier by getting everyone on board

Don’t surprise the team with commit policies

Bring real-life examples to support your case

Encourage using local commit verification
Introducing commit policies to your team
Use the Team Playbook by Atlassian

Use the 5 “Whys” Analysis
https://www.atlassian.com/team-playbook
Thank you!

Levente Szabo • levente.szabo@midori-global.com
Try our other apps, too (free)!

Better PDF Exporter for Jira
- Easy emailing, sharing, archiving, printing for Jira data

Better Excel Exporter for Jira
- Full-blown native Excel exports, spreadsheet reports and Business Intelligence for Jira

Better Content Archiving for Confluence
- Usage tracking, expiration, review workflow, retention and clean-up for your Confluence pages