# Clang-Format Implementation Spec

April 16, 2015
mark benvenuto



## Introduction

MongoDB engineers spend valuable time discussing code formatting style during code reviews. That time could be better spent discussing other facets of the proposed changes. Standardization of code appearance and formatting across the project can lead to accelerated comprehension of unfamiliar code, especially by less experienced engineers. This project proposes introducing clang-format into our development workflow to mitigate these issues.

## Goals

- Standardize on Clang-Format 3.6
- Migrate the MongoDB code base to one coding style standard by 3.2 release
- Enforce coding convention in MCI via "`scons format`" target on Linux
- Enable developers to format files (via `mongo-clang-format --update`)
- Provide developers with a simple workflow to ensure their changes match the coding style during code reviews (via optional switch in `upload.py`)
- Support developers on Linux, Mac OS X, and Windows

## Non-Goals

- Enforce WiredTiger style (BSD Kernel Normal Form (KNF) style) - this is already done by dist/s_all
- Have a rich per-directory customized PRESUBMIT.py like Chromium

## The Coding Style

Clang-format does not offer a way to exactly represent our current style. To do so we would need to distribute a custom version of the tool which we would be a long term burden to the development team, and company.

A new file called ".clang-format" will be added to "src/mongo". By placing the file here, we will be able to enforce coding style for only the mongodb code, and ignore third-party.

The format in the file is different from our existing style in two ways:

1. Zero indentation under namespaces instead of 4 space (called NamespaceIndentationKind in clang-format)

Old Style

```
namespace mongo {
    void main() {
    }
} // mongo
```

New Style

```
namespace mongo {
void main() {
}
} // namespace mongo
```

2. Brackets before else instead of on a new line

Old Style

```
if (foo()) {
}
else {
}
```

New Style

```
if (foo()) {
} else {
}
```

.clang-format

```
BasedOnStyle: Google
BreakBeforeBraces: Attach
AccessModifierOffset: -4
AllowShortFunctionsOnASingleLine: Inline
AllowShortIfStatementsOnASingleLine: false
BinPackArguments: false
BinPackParameters: false
ColumnLimit:     100
IndentWidth:     4
TabWidth:        4
```

## Implementation & Enforcement

During the development lifecycle, codes goes through several steps, and for each step we need to consider how clang-format will work with that step.

| Step | Mandatory | Clang-Format Integration |
|---|---|---|
| 1. Developer Writes Code in Editor | Yes | Developer can optionally have clang-format update in-progress code |
| 2. Developer Sends Code Review | Yes (but can be skipped) | Developer can optionally validate changes pre-upload |
| 3. Developer Sends Patch Request to MCI | Recommended | None. Developers may send CRs before MCI patches. It offers little value over CR. |
| 4. Developer Commits Code Locally to Git | Yes | None. Do not want to add additional developer steps or impact developer's incremental commits. |
| 5. Developer Pushes Code to remote Github repo | Yes | None. GitHub does not support server-side pre-commit hooks. |
| 6. MCI Periodic Run Validates Developer Change | Yes | Yes. "`scons format`" |

To support this development lifecycle, we will add a script to buildscripts. This script will support two modes of operation:

lint - check the style of existing files

```
mongo-clang-format.py lint [--clang-format|-cf=<file>] [file|glob|directory]
```

update - update the style of existing files

```
mongo-clang-format.py update [--clang-format|-cf=<file>] [file|glob|directory]
```

that will handle both linting and updating of files.

The Python script will be compatible with Python 2.7. It will not target older versions unless strictly needed.

1. Get a list of files that should be validated, and enforced. It will use `git ls-tree` and only include files in src/mongo.

2. Validate each file in the working tree that is not grandfathered by comparing before and after diffs of a clang-format run for that file. If there are only changes, it is considered incorrectly formatted.

## Getting Clang-Format

Since mongo-clang-format will enforce that clang-format is v3.6 and while this may represent an undue burden on developer productivity, mongo-clang-format will automatically download it, and cache a local copy of clang-format into the build directory if the correct version cannot be found automatically.

These files will be on a HTTP server at https://s3.amazonaws.com/boxes.10gen.com with files in `clang-format/<platform>/clang-format[.exe]`

To document this procedure, we are going to add a new task in MCI, to download clang-format from LLVM.org, install the binaries to /opt/clang-format, and push these binaries to boxes.10gen.com.

## Coding Style Enforcement - Pre-Code Review

```
upload.py --mongo-code-format[=<path_to_mongoformat.py]
```

One new optional switch will be added to upload.py to run the verification against the diff that is about to be uploaded to the code review server. This flag will be optional as many non-kernel projects also use `upload.py`.

The script will run `mongo-clang-format.py` against the patch. It will verify the contents of each “new file” as outlined above only. Upload.py will look for this file in two places:

```
<GIT_ROOT>\buildscripts\mongo-clang-format.py
<GIT_ROOT>\..\..\..\..\..\buildscripts\mongo-clang-format.py
```

```
<GIT_ROOT> = git rev-parse --show-toplevel
```

If it cannot find the script, it will ask the user to specify the location of the script. If the diff fails to meet the style, the upload will fail, and the user will be advised to run mongo-clang-format.py to reformat the patch.

### Coding Style Enforcement - Post-Commit

There will be a new target for to check format in scons

```
scons format
```

For MCI purposes, we will add this to the lint task on Linux-64 builds. If the download binaries from LLVM.org do not work, we will use “ubuntu-1404” instead.

A new target is added instead of just making it the default for the benefit of downstream packagers who will not need to run this check while making packages.

## Upgrading to future versions of clang-format

Each new version of clang-format makes subtle changes to the formatting rules. For instance, if we chose the Google Code style, this code in clang-format 3.6:

```
  options->addOptionChaining(
              "net.ssl.disabledProtocols", "sslDisabledProtocols",moe::String,
              "Comma separated list of disabled protocols").hidden();
```

would be changed in clang-format 3.7 (ie, trunk) to

```
  options->addOptionChaining("net.ssl.disabledProtocols",
                             "sslDisabledProtocols", moe::String,
                             "Comma separated list of disabled protocols")
      .hidden();
```

simply because the line breaking rules are different. This is just one example of many. For now, we have chosen to standardize on version 3.6. When we choose to upgrade to a newer version, we will have to bulk format the eligible code. There is no alternative if an upgrade is desired. We want to do these upgrades irregularly since they are disruptive to the code base.

## Developer Instructions & FAQ

As part of easing adoption, we will make a FAQ available on the website documenting these tools, and steps on the website.

### How to check code matches coding style before code review?

Call `upload.py` with `--check-clang-format`

### How do I just reformat my changes?

Run `mongo-clang-format.py update` in your local git repo.

### Can I reformat existing files?

No need. Mark will do this in bulk.

### Can I reformat other official branches?

We will not reformat released branches like v2.6 or v3.0. Feel free to reformat private branches at your own discretion.

### How to reformat files added in 3.2?

```
mongo-clang-format.py update <filename>
```

### How to do Git Blame?

Git blame can ignore whitespace differences, and in some case moved lines, but does not do a structural difference, only textual.

```
 -w -- ignores whitespace
 -M -- detects moved or copied lines

git blame -w -M
```

Also, git gui blame or other like tools are creating for skipping over this giant reformat change. See http://jfire.io/blog/2012/03/07/code-archaeology-with-git/

### How do I install Clang-Format?

`mongo-clang-format.py` will automatically download files from LLVM as approriate for the 3.6 release.

Clang 3.6 binaries can be downloaded from the LLVM website.
http://llvm.org/releases/download.html#3.6.0

## Implementation Plan

We will make a series of changes across 4 different repos to deploy the code changes.

- 10gen/kernel_tools
    - Update upload.py to learn about mongo-clang-format.py
- mongodb/mongo
    - Add new buildscripts/mongo-clang-format.py, and related python modules.
    - Modify SConstruct for new targets
- 10gen/toolchain-builder
    - Generate clang-format static binary on RHEL 55
- 10gen/evergreen-packer
    - Install clang-format on Linux-64
- MCI YAML changes
    - Update lint tasks for Linux-64 in mongodb/mongo
- Reformat code
- Give presentation to kernel team in kernel meeting

- Write wiki pages
- 10gen/mongodb-www-about
    - Update mongodb.org website for code style changes

## Editor Integration

Emacs CC Mode

Below are a collection of sample plugins. There are many others available on the web.
Visual Studio
Vim
XCode
Emacs
Eclipse
Sublime
Atom

## Reference

Clang format project scope document
Google C++ Style Guide
MongoDB C++ Style