Комментирование коммитов должно оформляться по соглашению оформления [AngularJS](https://docs.google.com/document/d/1QrDFcIiPjSLDn3EL15IJygNPiHORgU1_OOAqWjiDU5Y/edit#heading=h.uyo6cb12dt6w)

# Format of the commit message

**<type>(<scope>): <subject>**

**<BLANK LINE>**

**<body>**

**<BLANK LINE>**

**<footer>**

Any line of the commit message cannot be longer **100 characters**! This allows the message to be easier to read on github as well as in various git tools.

A commit message consists of a header, a body and a footer, separated by a blank line.

## Revert

If the commit reverts a previous commit, its header should begin with `**revert:** `, followed by the header of the reverted commit. In the body it should say: `This reverts commit <hash>.`, where the hash is the SHA of the commit being reverted.

## Message header

The message header is a single line that contains succinct description of the change containing a **type**, an optional **scope** and a **subject**.

### Allowed <type>

This describes the kind of change that this commit is providing.

* **feat** (feature)
* **fix** (bug fix)
* **docs** (documentation)
* **style** (formatting, missing semi colons, …)
* **refactor**
* **test** (when adding missing tests)
* **chore** (maintain)

### Allowed <scope>

Scope can be anything specifying place of the commit change. For example **$location**, **$browser**, **$compile**, **$rootScope**, **ngHref**, **ngClick**, **ngView**, etc...

You can use **\*** if there isn't a more fitting scope.

### <subject> text

This is a very short description of the change.

* use imperative, present tense: “change” not “changed” nor “changes”
* don't capitalize first letter
* no dot (.) at the end

## Message body

* just as in <subject> use imperative, present tense: “change” not “changed” nor “changes”
* includes motivation for the change and contrasts with previous behavior

<http://365git.tumblr.com/post/3308646748/writing-git-commit-messages>

<http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html>

## Message footer

### Breaking changes

All breaking changes have to be mentioned as a breaking change block in the footer, which should start with the word BREAKING CHANGE: with a space or two newlines. The rest of the commit message is then the description of the change, justification and migration notes.

BREAKING CHANGE: isolate scope bindings definition has changed and

    the inject option for the directive controller injection was removed.

    To migrate the code follow the example below:

    Before:

    scope: {

      myAttr: 'attribute',

      myBind: 'bind',

      myExpression: 'expression',

      myEval: 'evaluate',

      myAccessor: 'accessor'

    }

    After:

    scope: {

      myAttr: '@',

      myBind: '@',

      myExpression: '&',

      // myEval - usually not useful, but in cases where the expression is assignable, you can use '='

      myAccessor: '=' // in directive's template change myAccessor() to myAccessor

    }

    The removed `inject` wasn't generaly useful for directives so there should be no code using it.

### Referencing issues

Closed bugs should be listed on a separate line in the footer prefixed with "Closes" keyword like this:

Closes #234

or in case of multiple issues:

Closes #123, #245, #992

# Examples

**feat($browser): onUrlChange event (popstate/hashchange/polling)**

Added new event to $browser:

- forward popstate event if available

- forward hashchange event if popstate not available

- do polling when neither popstate nor hashchange available

**Breaks $browser.onHashChange, which was removed (use onUrlChange instead)**

**fix($compile): couple of unit tests for IE9**

Older IEs serialize html uppercased, but IE9 does not...

Would be better to expect case insensitive, unfortunately jasmine does

not allow to user regexps for throw expectations.

**Closes #392**

**Breaks foo.bar api, foo.baz should be used instead**

**feat(directive): ng:disabled, ng:checked, ng:multiple, ng:readonly, ng:selected**

New directives for proper binding these attributes in older browsers (IE).

Added coresponding description, live examples and e2e tests.

**Closes #351**

**style($location): add couple of missing semi colons**

**docs(guide): updated fixed docs from Google Docs**

Couple of typos fixed:

- indentation

- batchLogbatchLog -> batchLog

- start periodic checking

- missing brace

**feat($compile): simplify isolate scope bindings**

Changed the isolate scope binding options to:

  - @attr - attribute binding (including interpolation)

  - =model - by-directional model binding

  - &expr - expression execution binding

This change simplifies the terminology as well as

number of choices available to the developer. It

also supports local name aliasing from the parent.

**BREAKING CHANGE:** isolate scope bindings definition has changed and

the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

scope: {

  myAttr: 'attribute',

  myBind: 'bind',

  myExpression: 'expression',

  myEval: 'evaluate',

  myAccessor: 'accessor'

}

After:

scope: {

  myAttr: '@',

  myBind: '@',

  myExpression: '&',

  // myEval - usually not useful, but in cases where the expression is assignable, you can use '='

  myAccessor: '=' // in directive's template change myAccessor() to myAccessor

}

The removed `inject` wasn't generaly useful for directives so there should be no code using it.

Названия веток.

Мы будем использовать систему тегов:

1) Для реализации Issue: Issue\_<number>\_<addiction issue name>

2) Для "тестирования": Test\_<test name>

Пример: Issue\_123\_Implement\_class\_ComputerState

Оформление merge request:

Когда создавать merge request:

* Merge request можно создавать сразу после создания первого коммита при работе над конкретным issue. Тогда следует указать номер issue над которым ведётся работа (по поводу оформления смотрите ниже)
* При завершении работы, когда получается «рабочий прототип» результата. Тогда необходимо предоставить ваш код на обзор кода(code review) ответственным за проект (maintainer)

Оформление merge request:

<Title>:

<Body>:

* <What was released>
* <Solving info>
* <Solved issues list>

Обозначения:

<Title> - заголовок должен кратко излагать над чем ведётся работа. Если merge request создаётся после первого коммита для обозначения того, что задача принята в работу, тогда в начале заголовка необходимо указать «Draft:».

<Body> - информация о том, что было реализовано, при необходимости должны быть указаны средства, которыми это было реализовано.

Содержит следующий макет изложения:

* <What was released> - первый абзац должен содержать краткую информацию о том, что было реализовано. При реализации только всех требований issue и если в issue предоставлено подробное описание задачи данный раздел может быть пропущен. Если реализовались «элементы» по UML диаграммам (которые были составлены вами или иными участниками проекта и присутствующие в соответственной директории проекта), то достаточно перечислить какие «блоки» или классы были реализованы,с обязательным условием указания файла, где располагается данная схема.
* <Solving info> - второй абзац при необходимости должен содержать методы решения задачи, если они не были прописаны в issue или обговорены с ответственными за проект лицами, в особенности должны быть отмечены «костыли» и «узкие» моменты которые были применены по ходу решения задачи.
* <Solved issues list> - строка содержащая перечисления тех issue решение которых подразумевает данный merge request. В данный список попадают только те issue, которые ПОЛНОСТЬЮ нивелируются предоставленным вами решением. Список выглядит следующим образом «Closes: #<номер issue>»

Оформление issue:

Когда создавать issue:

* Сразу при формировании задачи. Любые задачи, имеющие конкретную формулировку должны быть представлены в виде issue (это касается не только задач на программирование, но и любых других). При составление загруженной формулировками и аспектами задачи желательно разбить её на несколько задач.
* Если во время решения задачи её решение было достигнуто, за исключением какого-либо аспекта, который требует значительных трудозатрат на решение, разрешается закрыть текущее issue существующим merge request, при этом создав новое issue для решение вырожденной проблемы. (этот же критерий подходит под условие, если для решение текущей задачи необходим значительный дополнительный функционал, реализация которого не подразумевался текущим issue, то так же необходимо создать отдельное issue, не закрывая предыдущее)
* Если во время тестирования\использования программы были обнаружены некоторые ошибки, то необходимо каждую отдельную ошибку предоставить в виде отдельного issue

Оформление issue:

<Title>:

<Body>:

* <What should be released>
* <Solving info>

Обозначения:

<Title> - заголовок должен кратко излагать задачу. Перед самим заглавием должны быть выставлены соответсвующие задаче теги:

* «Feature:» - если данная задача подразумевает создание дополнительного функционала.
* «Bug:» - если задача подразумевает исправление ошибки.
* «Frontend:» - если задача подразумевает изменение стилистических настроек программы (иконки и UI/UX стили, а так же изменение XML файлов).

<Body> - информация о том, что должно быть реализовано, при необходимости должны быть указаны средства, которыми это было реализовано.

Содержит следующий макет изложения:

* <What should be released> - первый абзац должен содержать краткую информацию о том, что должно быть реализовано. Если задача состоит в реализации «элементов» по UML диаграммам (которые были составлены вами или иными участниками проекта и присутствующие в соответственной директории проекта), то достаточно перечислить какие «блоки» или классы должны быть реализованы, с обязательным условием указания файла, где располагается данная схема и дополнительного прописывания «функционирования» при необходимости.
* <Solving info> - второй абзац при необходимости должен содержать методы решения задачи. Может быть предоставлен в виде словесных замечаний к выполнению или списка инструментов, если в этом есть необходимость.

Документирование (комментирование функций):

Обязательно должно присутствовать для каждого класса, метода, функции, перечисления и т. д.

Комментирование осуществляется по шаблонам [Javadoc](https://en.wikipedia.org/wiki/Javadoc#Technical_architecture) и [Doxygen](https://www.doxygen.nl/manual/docblocks.html) (так же рекомендуется для наилучшего оформления длинных описаний с [Markdown](https://ru.wikipedia.org/wiki/Markdown))

Вкратце:

Комментирование начинается в виде многострочного комментария перед комментируемым объектом(Автоматическая генерация поля: /\*\* + Enter).

Если это функция или метод, то комментирование быдет выглядеть следующим образом:

<Body> - информация по особенностям использования данного функционала, а так же информация о том, для чего создавался данный функционал

<Parameters> - список параметров с описанием того, что вы ожидаете на вход для каждого параметра. Список выглядит следующим образом:

@param <parameter name> <description>

…

<Return> - описание того, что данный функционал возвращает в качестве результата. Описание содержит следующий стиль.

@return <return description>

Вот пример идеального комментария:

