commit规范方案探讨

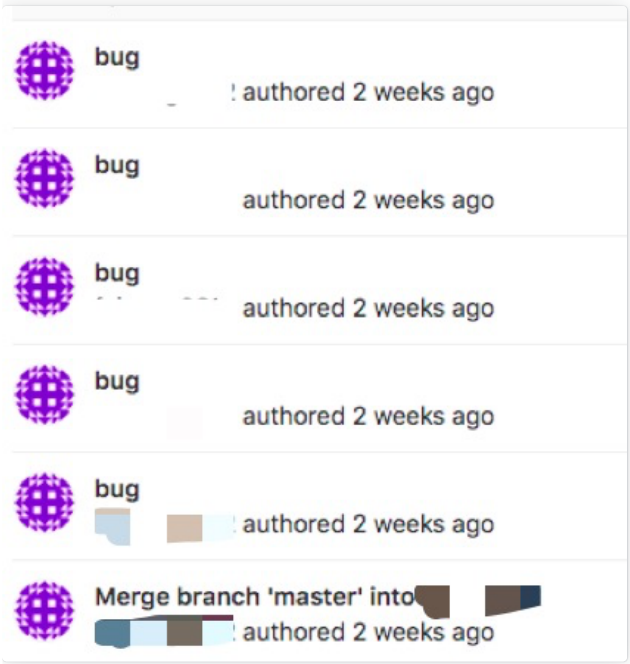
缘由

> 为什么要对commit进行规范化，每一次commit msg写啥，又不会影响我的代码运行?

确实，Commit的频率和信息多少，并不会影响你代码的执行，但是当你遇到需要上线分支中的一部分功能，想要cherry-pick的时候，就会很难受，commit msg清一色都是`update xxx`，也不知道要把哪些commit 找出来。

在项目开发开发中或许我们能经常看到

- 说不出所以然的一连串的 commit 提交日志
- commit 信息写的很简单，根本没有办法从 commit 信息中获知该 commit 用意的
- commit 信息写的很随意，commit 信息和变更代码之间不能建立联系的
- commit 信息写的过于冗余的



相信或多或少大家都曾碰到过。一旦涉及代码回滚，issue 回溯，changelog，语义化版本发布等操作时，你基本是无从下手的。

那，我们要做的是什么呢?

> 将commit 和 代码之间能建立起联系，并和相关的 issue 予以关联，做到任何代码都能区域性的解决问题。而 changelog，语义化版本发布这更像是合理化 commit 后水到渠成之事。

最佳实践

1) One Thing，One Commit；

> 在提交 commit 的时候尽量保证这个 commit 只做一件事情，比如实现某个功能或者修改了配置文件。

2）不要commit一半的工作；

> 当开发任务没有完整的完成的时候，不要commit。这不是说每次commit都需要开发完成一个非常完整的大功能，而是当把功能切分成许多小的但仍然具备完整性的功能点的时候，开发人员需要完整完成这个功能点之后才能commit。必要时可以使用stash命令对修改进行记录。

3）经常commit；

> 经常使用commit能够使你的commit（里的修改内容）越小，并且能使你commit相关的修改，多次commit允许你 推送自己代码到远程分支上的频率增加，能有效的减少merge代码时出现的代码冲突问题，因为多次 commit能使你的同事的代码库得到及时的更新。

4）commit之前的测试；

> 保证你所开发的功能是完整无误的。在commit代码之前的对代码充分测试是非常重要的，可以避免有问题的代码被其他开发人员使用。

5）编写规范的commit message；

> 规范的Commit Message，就是奥利给！

规范

> Angular Commit 规范

参考资料：Commit message 和 Change log 编写指南

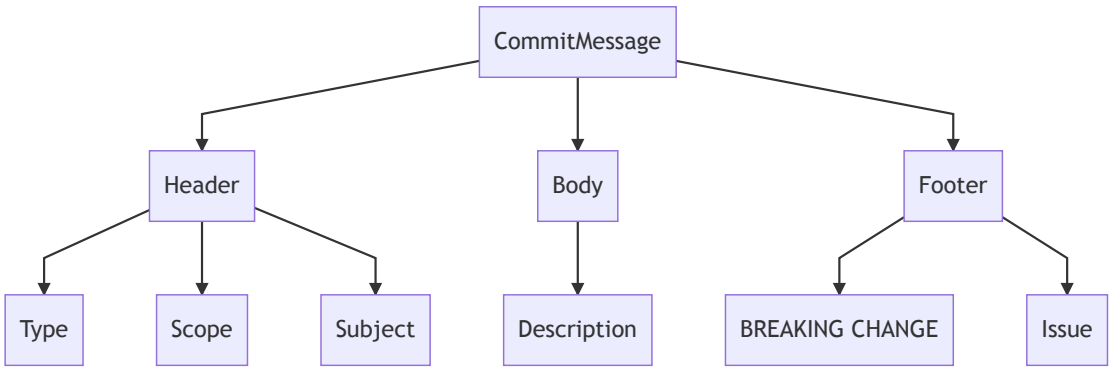
每次提交，Commit message 都包括三个部分：Header，Body 和 Footer。

```
<type>(<scope>): <subject>
// 空一行
<body>
// 空一行
<footer>
```

Header部分只有一行，包括三个字段：`type`（必需）、`scope`（可选）和`subject`（必需）。

Body 部分是对本次 commit 的详细描述，可以分成多行。

Footer 部分是对不兼容变动和关联的issue进行描述。



> thoughtbot 规范

地址：https://github.com/thoughtbot/dotfiles/blob/master/gitmessage

```
# 50-character subject line
#
# 72-character wrapped longer description. This should answer:
#
# * Why was this change necessary?
# * How does it address the problem?
# * Are there any side effects?
#
# Include a link to the ticket, if any.
```

1. 第一行不超过 50 个字符
2. 第二行空一行
3. 第三行开始是描述信息，每行长度不超过 72 个字符，超过了自己换行。
    1. 描述信息主要说明：
    2. 这个改动为什么是必要的?
    3. 这个改动解决了什么问题?
    4. 会影响到哪些其他的代码?
4. 最后最好有一个相应 ticket 的链接

Commit 这种格式有点类似邮件，主题不超过50个字符，然后空一行，这样显示的下面的部分都会折叠起来，类似下面的样子。我们使用命令 git log --oneline的时候就只显示第一行。

工具

参考资料：commitizen + husky 规范git提交信息

> 输入提示信息

commitizen

cz-conventional-changelog

cz-conventional-emoji

> lint工具

@commitlint/config-conventional

@commitlint/cli

> 自定义lint

commitlint-config-cz

cz-customizable

> 捕获commit

husky

> 生成changelog

standard-version

方案1

交互式提交+Commit规范

> 第一步：安装

```
yarn add -D commitizen cz-conventional-changelog @commitlint/config-conventional @commitlint/cli husky
```

> 第二步：配置

package.json

```
"scripts": {
  "commit":"git-cz",
},


"config":{
    "commitizen":{
        "path":"node_modules/cz-conventional-changelog"
    }
},

"husky": {
    "hooks": {
      "commit-msg": "commitlint -E HUSKY_GIT_PARAMS"
    }
}
```

commitlint.config.js

```
module.exports = {
    extents:[
        "@commitlint/config-conventional"
    ],
    rules:{
        'body-leading-blank': [1, 'always'],
        'footer-leading-blank': [1, 'always'],
        'header-max-length': [2, 'always', 72],
        'scope-case': [2, 'always', 'lower-case'],
        'subject-case': [
            2,
            'never',
            ['sentence-case', 'start-case', 'pascal-case', 'upper-case']
        ],
        'subject-empty': [2, 'never'],
        'subject-full-stop': [2, 'never', '.'],
        'type-case': [2, 'always', 'lower-case'],
        'type-empty': [2, 'never'],
        'type-enum': [
            2,
            'always',
            [
                'build',
                'chore',
                'ci',
                'docs',
                'feat',
                'fix',
                'improvement',
                'perf',
                'refactor',
                'revert',
                'style',
                'test'
            ]
        ]
    }
}
```

方案2

交互式提交+自定义提示文案+Commit规范

> 第一步：安装

```
yarn add -D commitizen cz-conventional-changelog @commitlint/config-conventional @commitlint/cli commitlint-config-cz cz-customizable husky
```

> 第二步：配置

和方案1类似，只是在其基础上修改如下配置：

package.json

```
"scripts": {
  "commit":"git-cz",
},

"config": {
    "commitizen": {
      "path": "node_modules/cz-customizable"
    }
},


"husky": {
    "hooks": {
      "commit-msg": "commitlint -E HUSKY_GIT_PARAMS"
    }
}
```

commitlint.config.js

```
// 增加cz
{
  extends: ["@commitlint/config-conventional", "cz"],
}
```

.cz-config.js

```
module.exports = {
  types: [
    {      value: 'init',      name: 'init:      初始提交'    },
    {      value: 'feat',      name: 'feat:      增加新功能'    },
    {      value: 'fix',      name: 'fix:      修复bug'    },
    {      value: 'ui',      name: 'ui:       更新UI'    },
    {      value: 'refactor',      name: 'refactor: 代码重构'    },
    {      value: 'release',      name: 'release:  发布'    },
    {      value: 'deploy',      name: 'deploy:   部署'    },
    {      value: 'docs',      name: 'docs:     修改文档'    },
    {      value: 'test',      name: 'test:     增删测试'    },
    {      value: 'chore',      name: 'chore:    更改配置文件'    },
    {      value: 'style',      name: 'style:    代码样式修改不影响逻辑'    },
    {      value: 'revert',      name: 'revert:   版本回退'    },
    {      value: 'add',      name: 'add:      添加依赖'    },
    {      value: 'minus',      name: 'minus:    版本回退'    },
    {      value: 'del',      name: 'del:      删除代码/文件'    }
  ],
  scopes: [],
  messages: {
    type: '选择更改类型:\n',
    scope: '更改的范围:\n',
    // 如果allowcustomscopes为true，则使用
    // customScope: 'Denote the SCOPE of this change:',
    subject: '简短描述:\n',
    body: '详细描述，使用"|"换行:\n',
    breaking: 'Breaking Changes列表:\n',
    footer: '关闭的issues列表，E.g.: #31, #34:\n',
```

```
      confirmCommit: '确认提交?'
    },
    allowCustomScopes: true,
    allowBreakingChanges: ["feat", "fix"]
  };
```

交互界面

方案3

交互式提交+emoji表情包

> 第一步：安装

```
yarn add -D commitizen cz-conventional-emoji
```

> 第二步：配置package.json，增加以下配置

```
"scripts": {
  "commit":"git-cz",
},
"config":{
  "commitizen":{
    "path":"node_modules/cz-conventional-emoji"
  }
},
```

方案4

全局模式，等同于方案1，只是cz全局安装

> 第一步：安装

```
# 全局安装
npm install -g commitizen cz-conventional-changelog
# 项目内安装
yarn add -D @commitlint/config-conventional @commitlint/cli husky
```

> 第二步：配置

.czrc

需要在全局根目录（在命令行工具，输入 `cd ~` ，就可以找到）下建立.czrc文件,然后文件中输入内容 `{“path”:"cz-conventional-changelog"}` 或者键入如下命令：

```
echo '{"path":"cz-conventional-changelog"}' > ~/.czrc
```

package.json

```
"husky": {
    "hooks": {
      "commit-msg": "commitlint -E HUSKY_GIT_PARAMS"
    }
}
```

commitlint.config.js 配置等同于方案1.

方案5

> 基于方案2再扩展

如果我们希望使用emoji，可以接受部分commitlint规范关闭，我们其实可以基于方案二直接扩展emoji。

> 第一步：安装

```
# 全局安装
npm install -g commitizen cz-conventional-changelog
# 项目内安装
yarn add -D  @commitlint/config-conventional @commitlint/cli commitlint-config-cz cz-customizable husky
```

> 第二步：配置

package.json

```
# 配置自定义commit
"config": {
    "commitizen": {
      "path": "node_modules/cz-customizable"
    }
},
# 配置commit信息校验
"husky": {
    "hooks": {
      "commit-msg": "commitlint -E HUSKY_GIT_PARAMS"
    }
}
```

.czrc

需要在全局根目录下建立.czrc文件,然后文件中输入内容 `{“path”:"cz-conventional-changelog"}` 或者键入如下命令：

```
echo '{"path":"cz-conventional-changelog"}' > ~/.czrc
```

.cz-config.js

```
module.exports = {
  types: [
    { value: "✨feat",          name: "feat:          增加新功能" },
    { value: "🐾fix",           name: "fix:           修复bug" },
    { value: "📝docs",          name: "docs:          修改文档" },
    { value: "⚡perf",           name: "perf:          性能优化" },
```

```
      { value: "🎉init",          name: "init:          初始提交" },
      { value: "➕add",           name: "add:           添加依赖" },
      { value: "🔨build",         name: "build:         打包" },
      { value: "🔧chore",         name: "chore:         更改配置文件" },
      { value: "👷ci",            name: "ci:            CI部署" },
      { value: "🔥del",           name: "del:           删除代码/文件" },
      { value: "♻️refactor",      name: "refactor:      代码重构" },
      { value: "⏪revert",        name: "revert:        版本回退" },
      { value: "💄style",         name: "style:         样式修改不影响逻辑" },
      { value: "✅test",          name: "test:          增删测试" },
    ],
    scopes: [],
    messages: {
      type: "选择更改类型:\n",
      scope: "更改的范围:\n",
      // 如果allowcustomscopes为true，则使用
      // customScope: 'Denote the SCOPE of this change:',
      subject: "简短描述:\n",
      body: '详细描述，使用"|"换行:\n',
      breaking: "Breaking Changes列表:\n",
      footer: "关闭的issues列表，E.g.: #31, #34:\n",
      confirmCommit: "确认提交?",
    },
    allowCustomScopes: true,
    allowBreakingChanges: ["feat", "fix"],
  };
```

commitlint.config.js

```
module.exports = {
  extends: ["@commitlint/config-conventional", "cz"],
  rules: {
    "body-leading-blank": [1, "always"],
    "footer-leading-blank": [1, "always"],
    "header-max-length": [2, "always", 72],
    "scope-case": [2, "always", "lower-case"],
    "subject-case": [
      2,
      "never",
      ["sentence-case", "start-case", "pascal-case", "upper-case"],
    ],
    "subject-empty": [0],
    "subject-full-stop": [2, "never", "."],
    "type-case": [0],
    "type-empty": [0],
    "type-enum": [0],
  },
};
```

使用

我们采用了 方案5 作为最终方案。

> 本地模式

`git add` 后，执行 `npm run commit` 或者 `yarn commit`，就可以开始使用了。

> 全局模式

`git add` 后，执行 `git cz`，就可以开始使用了。

常用type

```
feat: 新功能（feature）
fix: 修改Bug
refactor: 重构（即不是新增功能，也不是修改bug的代码变动）
docs: 文档修改
style: 代码格式（风格）修改，注意不是 css 修改（不影响代码运行的变动）
test: 测试用例修改
chore: 其他修改，比如构建流程，依赖管理.
```

standard-version

> 第一步：安装

```
yarn add standard-version -D
```

> 第二步：配置

配置package.json

```
"scripts": {
    "release": "standard-version"
}
```

> 第三步：运行

执行 `yarn release` 生成CHANGELOG.md。

参考

http://legendtkl.com/2016/12/22/git-good-practice-commit-msg/

https://juejin.im/post/5cc7be8bf265da036c579597

https://segmentfault.com/a/1190000009048911

https://shimo.im/docs/18AIXO42yGlgFxAB/read