Git Commit: Boas Práticas

Matheus Tavares

https://matheustavares.gitlab.io/

Parece uma boa ideia...

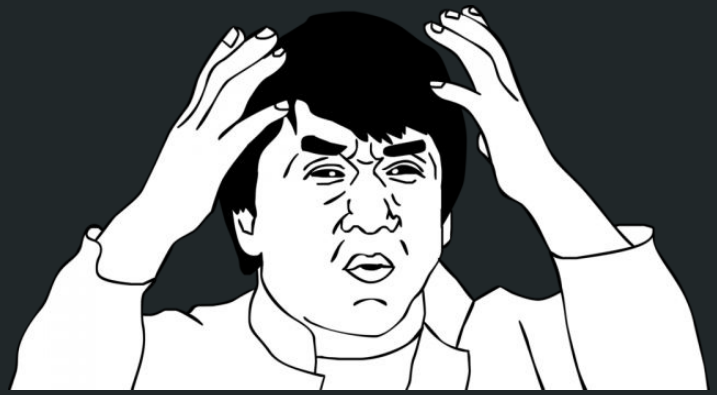
$ vim file.c

$ git add .

$ git commit -m “Update file.c”

$ git push origin master

... até que você/alguém precise relembrar o que foi feito.



https://blog.tiagopariz.com/wp-content/uploads/2018/10/jackie-chan-meme.png

```
commit 5e2faaea31dceaad23a3331cd9ad20afad6c719b
Author: joh123 <>

    Minor changes

commit ae95825114ccac3a996251bd1de4da5e83c95172
Author: joh123 <>

    Fixed some bugs

commit 6e639b014f82521138025091fac660cd2474b7a0
Author: joh123 <>

    Now it compiles!

commit f717b52ddd1e95dbbdd5a1476051b842468cb195
Author: jonh123 <>

    Actually it didn't. NOW IT DOES

commit 90384aec06d44a5d40d9c299f082334503943b32
Author: john123 <>

    Update README
```

Além disso, bons commits:

- São de grande ajuda para futuros contribuidores.
- Ajudam a entender **as motivações** por trás de uma dada linha de código (“*git archeology*”):
  - $ git bisect
  - $ git blame
  - $ git log -S/-G

Dissecando um bom commit

1. Mudanças **não correlacionadas** pertencem a **commits separados**.
2. Invista na escrita de mensagens de commit **informativas**.
3. **Não *commitar*** blocos de trabalho **incompletos**. (*soft rule*)

1. Mudanças **não correlacionadas** pertencem a **commits separados**.

- Mais fácil de **revisar**
  ➜ melhores revisões ➜ melhor código, menos bugs, etc.
- Mais fácil de **reverter**, caso necessário.
- Mais fácil de **integrar (merge)** com outros commits e branches.
  (conflitos menores e/ou mais fáceis de resolver)

1. Mudanças **não correlacionadas** pertencem a **commits separados**.

2. Invista na escrita de mensagens de commit **informativas**.

> “a well-crafted Git commit message is the best way to communicate context about a change to fellow developers (and indeed to [your future self]). **A diff will tell you what changed, but only the commit message can properly tell you why**.”

- Chris Beams (https://cbea.ms/git-commit/)

2. Invista na escrita de mensagens de commit **informativas**.

Passos:

1. **Descreva o problema**: o que não está legal no código atual?
2. **Justifique como as mudanças resolvem o problema**: porque o estado do projeto após este commit é melhor do que o atual?
3. **Alternativas descartadas [opcional]**: existem outros modos de implementar a mudança? Se sim, porque este foi escolhido?

Dica:

git commit -v ou

git config --global commit.verbose true

2. Invista na escrita de mensagens de commit **informativas**.

Foque no **porquê** não no **como**.

```
Adquira mutex ao escrever na variável X

Chame a função pthread_mutex_lock()
antes de escrever na variável X, usada
para contar o número de arquivos, e
também chame pthread_mutex_unlock()
após a escrita.
```

```
Proteja a variável X contra escritas em
paralelo

A variável X, utilizada para contar o
número de arquivos, é escrita
concorrentemente por múltiplas threads.
Adquira mutex para garantir que não
haja condição de corrida (e portanto,
erros de sobrescrita) em X.
```

3. **Não *commitar*** blocos de trabalho **incompletos**. *(soft rule)*

- É legal garantir que **cada commit** seja compilável e passe os testes.
  - Mais fácil encontrar bugs / mudanças de comportamento com ***git bisect***
  - Mais fácil justificar, revisar e relembrar (git archeology) cada mudança auto-contida.

Dica: você pode *arrumar* os commits depois.

Possível *workflow*:

1. Criar nova branch a partir da principal.
2. Desenvolver + *commitar* frequentemente (mesmo incompleto)
3. "*git rebase -i*" para arrumar a branch.
4. Fazer o merge ou pull request.
5. [opcional]: se houver pedidos de revisão (PR), retornar ao 2. e 3.

O formato da mensagem também é importante

- **Cada organização/grupo define suas regras.**
- Lint: veja *pre-commit hooks*, *commitlint*, etc.
- Vejamos algumas das regras mais comuns em repos como do Linux e Git.

O formato da mensagem também é importante

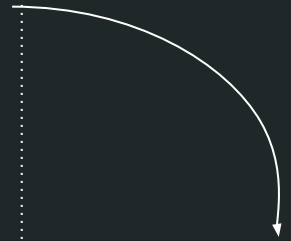
```
Título resumindo as mudanças em até 50 chars (soft rule)

Corpo, separado do título por uma linha em branco, e
justificado em 72 colunas (soft rule). Explica a mudança
em mais detalhes, como descrito no slide anterior. O
corpo pode ser omitido quando as mudanças são triviais.

O corpo pode ter múltiplos parágrafos e também:

  - Bullet points
  - Tabelas, links (mas cuidado com links efêmeros) e
    outros

Trailers
```



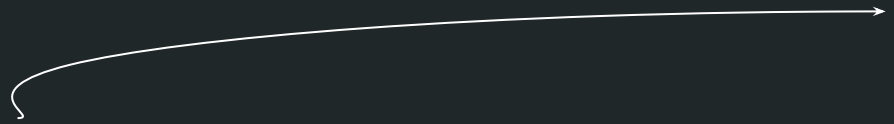
Normalmente:

- No imperativo
- Não pontuado
- Pode ter um prefixo **“area: ”** para indicar a área/subsistema modificada/o

O formato da mensagem também é importante

Trailers:

- Closes #21
- Fix #53
- Signed-off-by: A U Thor <author@example.com>
- Co-authored-by: A U Thor <author@example.com>
- Reviewed-by: A U Thor <author@example.com>
- Reported-by: A U Thor <author@example.com>
- etc.



Vem do Developer Certificate of Origin

Dissecando um bom commit

Exemplo do Git:

```
commit 7655b4119d49844e6ebc62da571e5f18528dbde8
Author: René Scharfe <l.s.r@web.de>
Date:   Tue Mar 3 21:55:34 2020 +0100

remote-curl: show progress for fetches over dumb HTTP

Fetching over dumb HTTP transport doesn't show any progress, even with
the option --progress.  If the connection is slow or there is a lot of
data to get then this can take a long time while the user is left to
wonder if git got stuck.

We don't know the number of objects to fetch at the outset, but we can
count the ones we got.  Show an open-ended progress indicator based on
that number if the user asked for it.

Signed-off-by: René Scharfe <l.s.r@web.de>
Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

Outras dicas

- Verifique erros de espaçamento antes do commit: *$ git diff --check*
- Use *git commit -v* (ou commit.verbose) para ajudar a escrever mensagens.
- Use *git rebase -i* para organizar sua branch pessoal.
- Considere utilizar githooks.
- Se contribuindo para um projeto, leia commits feitos no mesmo arquivo para entender o padrão utilizado pela comunidade.
    - *$ git log --no-merges <arquivo>*

Outras dicas

- Obviamente, para projetos pessoais/menores, estas regras podem ser afrouxadas.
- Para entender melhor a importância de boas mensagens de commit: https://bmuskalla.github.io/blog/2021-01-12-git-archeology/

Referências

1. **How to Write a Git Commit Message**, Criss Beans: https://chris.beams.io/posts/git-commit/
2. **Developer Tip: Keep Your Commits "Atomic"**, Sean Patterson: https://www.freshconsulting.com/atomic-commits/
3. **Pro Git**, Scott Chacon and Ben Straub: https://git-scm.com/book/en/v2
4. **SubmittingPatches,** Git community https://git-scm.com/docs/SubmittingPatches