Debugando com git

Matheus Tavares
https://matheustavares.gitlab.io/

Arqueologia com git

Matheus Tavares
https://matheustavares.gitlab.io/

[Arqueologia é] o **estudo do passado** através da recuperação e **análise** sistemática da **cultura material**

Paul G. Bahn, 1992

pathspecs

- Padrões para especificar *paths*.
- Coringas
  - *.js
  - **/src/Makefile
  - test/**/*.c
- Colchetes angular
  - logger.[ch]
- *Magic signatures*
  - :/README
  - :^*.md

- **Dica**: use ‘...’ para evitar a expansão do shell.
- Mais info aqui .

grep

- Semelhante ao GNU grep
- Respeita *.gitignore* e *.git*
- Recursivo por padrão
- **Procura em commits/trees**

Alternativa interessante: ripgrep

grep

```
$ git grep show_submodule
builtin/ls-files.c:static void show_submodule(struct repository *superproject,
builtin/ls-files.c:             show_submodule(repo, dir, ce->name);
diff.c:         show_submodule_diff_summary(o, one->path ? one->path : two->path,
diff.c:         show_submodule_inline_diff(o, one->path ? one->path : two->path,
submodule.c:static void show_submodule_header(struct diff_options *o,
submodule.c:void show_submodule_diff_summary(struct diff_options *o, const char *path,
submodule.c:    show_submodule_header(o, path, one, two, dirty_submodule,
submodule.c:void show_submodule_inline_diff(struct diff_options *o, const char *path,
submodule.c:    show_submodule_header(o, path, one, two, dirty_submodule,
submodule.h:void show_submodule_diff_summary(struct diff_options *o, const char *path,
submodule.h:void show_submodule_inline_diff(struct diff_options *o, const char *path,
```

grep

```
$ git grep show_submodule 752c0c24
752c0c24:diff.c:                show_submodule_summary(o->file, one ? one->path : two->path,
752c0c24:submodule.c:void show_submodule_summary(FILE *f, const char *path,
752c0c24:submodule.h:void show_submodule_summary(FILE *f, const char *path,
```

grep

```
$ git grep -n show_submodule
builtin/ls-files.c:221:static void show_submodule(struct repository *superproject,
builtin/ls-files.c:313:                 show_submodule(repo, dir, ce->name);
diff.c:3473:                    show_submodule_diff_summary(o, one->path ? one->path : two->path,
diff.c:3481:                    show_submodule_inline_diff(o, one->path ? one->path : two->path,
submodule.c:546:static void show_submodule_header(struct diff_options *o,
submodule.c:617:void show_submodule_diff_summary(struct diff_options *o, const char *path,
submodule.c:627:                show_submodule_header(o, path, one, two, dirty_submodule,
submodule.c:657:void show_submodule_inline_diff(struct diff_options *o, const char *path,
submodule.c:669:                show_submodule_header(o, path, one, two, dirty_submodule,
submodule.h:77:void show_submodule_diff_summary(struct diff_options *o, const char *path,
submodule.h:80:void show_submodule_inline_diff(struct diff_options *o, const char *path,
```

grep

```
$ git grep -n --heading --break show_submodule
builtin/ls-files.c
221:static void show_submodule(struct repository *superproject,
313:            show_submodule(repo, dir, ce->name);

diff.c
3473:           show_submodule_diff_summary(o, one->path ? one->path : two->path,
3481:           show_submodule_inline_diff(o, one->path ? one->path : two->path,

submodule.c
546:static void show_submodule_header(struct diff_options *o,
617:void show_submodule_diff_summary(struct diff_options *o, const char *path,
627:    show_submodule_header(o, path, one, two, dirty_submodule,
657:void show_submodule_inline_diff(struct diff_options *o, const char *path,
669:    show_submodule_header(o, path, one, two, dirty_submodule,

submodule.h
77:void show_submodule_diff_summary(struct diff_options *o, const char *path,
80:void show_submodule_inline_diff(struct diff_options *o, const char *path,
```

grep

```
$ git grep -W show_submodule
builtin/ls-files.c:221:static void show_submodule(struct repository *superproject,
builtin/ls-files.c-222-                    struct dir_struct *dir, const char *path)
builtin/ls-files.c-223-{
builtin/ls-files.c-224- struct repository subrepo;
builtin/ls-files.c-225-
builtin/ls-files.c-226- if (repo_submodule_init(&subrepo, superproject, path, null_oid()))
builtin/ls-files.c-227-         return;
builtin/ls-files.c-228-
builtin/ls-files.c-229- if (repo_read_index(&subrepo) < 0)
builtin/ls-files.c-230-         die("index file corrupt");
builtin/ls-files.c-231-
builtin/ls-files.c-232- show_files(&subrepo, dir);
builtin/ls-files.c-233-
builtin/ls-files.c-234- repo_clear(&subrepo);
builtin/ls-files.c-235-}
--
```

blame

- Qual commit alterou uma linha por último.

```
$ git blame log-tree.c
...
959a26231f5 (Brandon Casey           2013-02-12 02:17:39 -0800   18) #include "sequencer.h"
12da1d1f6ff (Thomas Rast             2013-03-28 17:47:32 +0100   19) #include "line-log.h"
3ac68a93fd2 (Nguyễn Thái Ngọc Duy    2018-05-26 15:55:24 +0200   20) #include "help.h"
40ce41604da (Eric Sunshine           2018-07-22 05:57:17 -0400   21) #include "range-diff.h"
20323d104ec (Elijah Newren           2022-02-02 02:37:35 +0000   22) #include "strmap.h"
5f1c3f07b7f (Junio C Hamano          2006-04-09 01:11:11 -0700   23)
2608c24940c (Jeff King               2014-08-26 06:23:54 -0400   24) static struct decoration name_decoration = { "object names" };
76c61fbdbab (Junio C Hamano          2015-05-13 10:25:18 -0700   25) static int decoration_loaded;
76c61fbdbab (Junio C Hamano          2015-05-13 10:25:18 -0700   26) static int decoration_flags;
a7524128750 (Nazri Ramliy            2010-06-19 09:37:34 +0800   27)
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   28) static char decoration_colors[][COLOR_MAXLEN] = {
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   29)    GIT_COLOR_RESET,
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   30)    GIT_COLOR_BOLD_GREEN,   /* REF_LOCAL */
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   31)    GIT_COLOR_BOLD_RED,     /* REF_REMOTE */
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   32)    GIT_COLOR_BOLD_YELLOW,  /* REF_TAG */
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   33)    GIT_COLOR_BOLD_MAGENTA, /* REF_STASH */
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   34)    GIT_COLOR_BOLD_CYAN,    /* REF_HEAD */
76f5df305be (Nguyễn Thái Ngọc Duy    2011-08-18 19:29:37 +0700   35)    GIT_COLOR_BOLD_BLUE,    /* GRAFTED */
67a4b5864f9 (Nazri Ramliy            2010-06-19 09:37:35 +0800   36) };
...
```

blame

- -w: ignora mudanças de “*whitespaces*”
- tig blame permite um blame iterativo/recursivo.
- Limitações: **última mudança** nem sempre é a **mais interessante**
  - Renomear variável
  - Adicionar parâmetro à função
  - Remover espaços
  - etc.

log

- **pathspec**
- **--since= / --after=** limita por data
- **--grep=** procura regex na mensagem de commit
- ***hashA…hashB*** limita entre dois commits
- ***--graph*** mostra o grafo
- ***-p*** mostra o diff
- ***--format*** altera o que vai ser mostrado

pickaxe

- log -G: commits cujo **diff contém uma string**
- log -S: commits cujo **diff muda o número de ocorrências de uma string**

```
$ git log -S show_submodule --oneline
188dce131f ls-files: use repository object
fd47ae6a5b diff: teach diff to display submodule difference with an inline diff
8e6df65015 submodule: refactor show_submodule_summary with helper function
752c0c2492 Add the --submodule option to the diff option family
```

- Dica: combine com *--reverse*

Recuperando dados

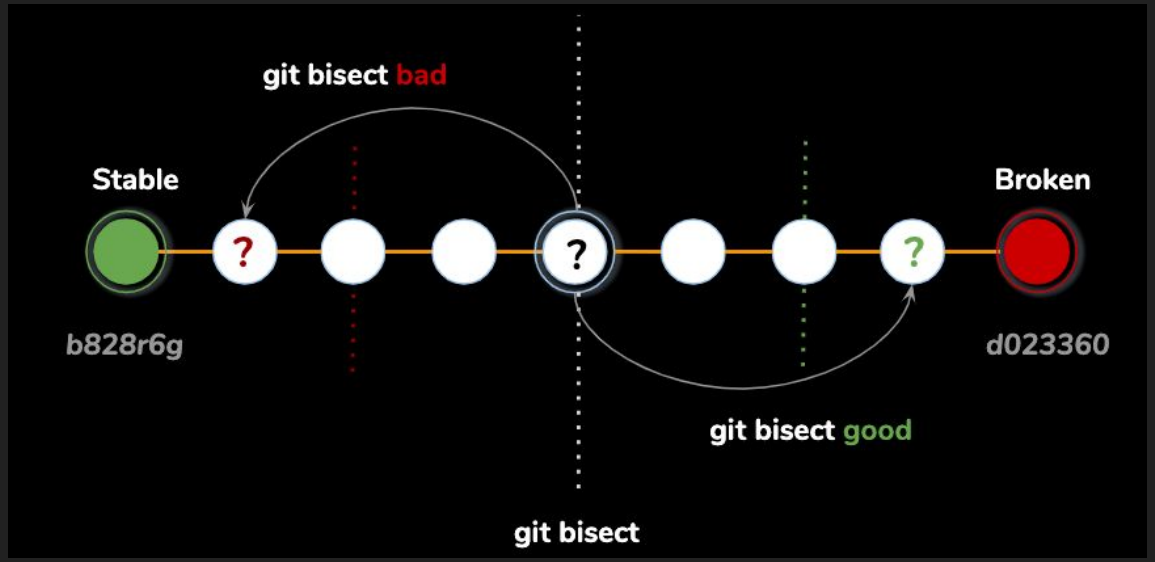
- reflog: log de referencias (branches, tags, HEAD, etc.)

```
$ git reflog HEAD
b1485644f9 (HEAD -> master, origin/master, origin/main, origin/HEAD) HEAD@{0}: pull: Fast-forward
7c2ef319c5 (mailmap) HEAD@{1}: checkout: moving from mailmap to master
7c2ef319c5 (mailmap) HEAD@{2}: rebase (finish): returning to refs/heads/mailmap
7c2ef319c5 (mailmap) HEAD@{3}: pull: Fast-forward
bd5df96b79 HEAD@{4}: checkout: moving from master to mailmap
bd5df96b79 HEAD@{5}: pull: Fast-forward
79f2338b37 HEAD@{6}: checkout: moving from 87d8ec31e8cb14c4076abc93bf73f070f3951834 to master
```

- git fsck --lost-found: procura e salva objetos inalcançáveis
  (i.e. “a serem deltados em breve™”)

bisect

- Procura o commit que introduziu um bug
- Ou, de forma mais generica: **mudou um comportamento**

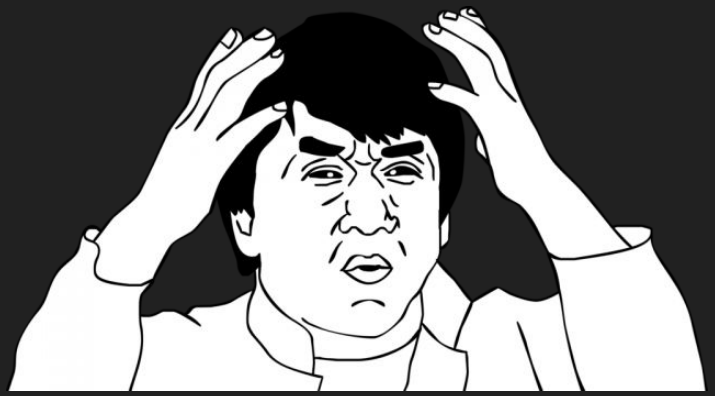


Fonte: https://www.datree.io/resources/git-bisect-debugging-with-git

Detalhes sobre o algoritmo do bisect aqui e aqui (paper).

Pra que tudo isso funcione bem….

… é preciso ter um bom histórico.



```
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
```

Dissecando um bom commit

1. Mudanças **não correlacionadas** pertencem a **commits separados**.
2. Escreva 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**
- Mais fácil de **integrar (merge)**
  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**

“Uma mensagem de commit do Git bem elaborada é a melhor maneira de comunicar o contexto sobre uma mudança para colegas desenvolvedores (e, na verdade, para [seu futuro eu]). **Um diff lhe dirá o que mudou, mas apenas a mensagem do commit poderá lhe dizer o porquê.**”

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

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

Passos:

1. **Descreva o problema.**
2. **Justifique como as mudanças resolvem o problema**.
3. **Alternativas descartadas [opcional]**.

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 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.**
  (veja docs e o log)
- Lint: veja *pre-commit hooks*, *commitlint*, etc.

Formato comum

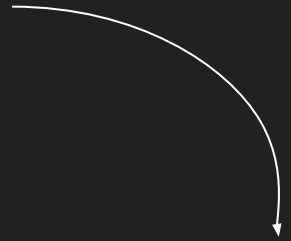
Título resumindo as mudanças em até 50 chars *(soft rule)*

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>
```

Dicas extras

- *$ git help glossary*
- *$ git help revisions*
- *Git “under the hood” (slides)*
- *visualizing-git e* git-sim *(tools)*
- Theoretical analysis of git bisect (paper)

Referências

1. **Git Archeology,** Benjamin Muskalla https://bmuskalla.github.io/blog/2021-01-12-git-archeology/
2. **Git como ferramenta de debug,** Lucas Oshiro https://lucasoshiro.github.io/posts/2023-02-13-git-debug/
3. **Pro Git**, Scott Chacon and Ben Straub: https://git-scm.com/book/en/v2
4. **Git Docs:** https://git-scm.com/docs/
5. **How to Write a Git Commit Message**, Criss Beans: https://chris.beams.io/posts/git-commit/

Obrigado
Comentários? :)