








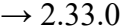
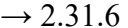
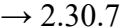
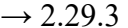


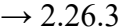
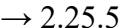







[Latest version](#) ▾ git last updated in 2.39.1

Changes in the **git** manual

1. 2.39.1 no changes
2. [2.39.0](#)  [12/12/22](#)
3. 2.38.3 no changes
4. [2.38.2](#)  [12/11/22](#)
5. 2.38.1 no changes
6. [2.38.0](#)  [10/02/22](#)
7. 2.37.3 → 2.37.5 no changes
8. [2.37.2](#)  [08/11/22](#)
9. 2.37.1 no changes
10. [2.37.0](#)  [06/27/22](#)
11. 2.36.1 → 2.36.4 no changes
12. [2.36.0](#)  [04/18/22](#)
13. 2.35.1 → 2.35.6 no changes
14. [2.35.0](#)  [01/24/22](#)
15. 2.34.1 → 2.34.6 no changes
16. [2.34.0](#)  [11/15/21](#)
17. 2.33.3 → 2.33.6 no changes
18. [2.33.2](#)  [03/23/22](#)
19. [2.33.1](#)  [10/12/21](#)
20. 2.32.1 → 2.33.0 no changes
21. [2.32.0](#)  [06/06/21](#)
22. 2.31.1 → 2.31.6 no changes
23. [2.31.0](#)  [03/15/21](#)
24. 2.30.1 → 2.30.7 no changes
25. [2.30.0](#)  [12/27/20](#)
26. 2.29.1 → 2.29.3 no changes
27. [2.29.0](#)  [10/19/20](#)
28. 2.28.1 no changes
29. [2.28.0](#)  [07/27/20](#)
30. 2.27.1 no changes
31. [2.27.0](#)  [06/01/20](#)
32. 2.26.1 → 2.26.3 no changes
33. [2.26.0](#)  [03/22/20](#)
34. 2.25.2 → 2.25.5 no changes
35. [2.25.1](#)  [02/17/20](#)
36. [2.25.0](#)  [01/13/20](#)
37. 2.23.1 → 2.24.4 no changes
38. [2.23.0](#)  [08/16/19](#)
39. 2.22.2 → 2.22.5 no changes
40. [2.22.1](#)  [08/11/19](#)
41. [2.22.0](#)  [06/07/19](#)
- 42.

Check your version of git by running

```
git --version
```

# NOME

git - o monitor de conteúdo estúpido

# RESUMO

```
git [--version] [--help] [-C <caminho>] [-c <nome>=<valor>]
  [--exec-path[=<caminho>]] [--html-path] [--man-path] [--info-path]
  [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]
  [--git-dir=<caminho>] [--work-tree=<caminho>] [--namespace=<nome>]
  [--super-prefix=<caminho>]
  <comando> [<args>]
```

## DESCRIÇÃO

O Git é um sistema de controle de revisão distribuído, rápido, escalável e com um conjunto de comandos incommumente rico que oferece operações de alto nível e acesso completo aos seus recursos.

Para começar, consulte [gittutorial\[7\]](#) e depois [giteveryday\[7\]](#) para conhecer um conjunto mínimo de comandos uteis. Para uma introdução mais aprofundada acesse o [Manual do Usuário do Git](#).

Depois de dominar os conceitos básicos, é possível voltar para esta página para aprender quais os outros comandos o Git oferece. É possível aprender mais sobre os comandos individuais do Git com o comando "git help nome-do-comando". A página do manual [giteli\[7\]](#) fornece uma visão geral da sintaxe de comandos da linha de comando.

Uma cópia formatada e com hiperlink da documentação mais recente do Git pode ser visualizada em <https://git.github.io/htmldocs/git.html> ou <https://git-scm.com/docs>.

## OPÇÕES

--version

Imprime a versão do pacote Git exibindo a sua origem.

--help

Imprime a sinopse e uma lista dos comandos mais usados. Caso a opção --all ou -a seja usada, todos os comandos disponíveis serão impressos. Caso um comando Git seja informado, esta opção exibirá a página do manual deste comando.

Outras opções estão disponíveis para controlar como a página do manual é exibida. Para mais informações, consulte [git-help\[1\]](#), pois o comando git --help ... é convertido internamente em git help ....

-C <caminho>

Execute como se o git tivesse sido iniciado em <caminho> em vez do diretório de trabalho atual. Quando várias opções -C são usadas, cada -C <caminho> não absoluto subsequente é interpretado com relação ao -C <caminho> anterior. Se <caminho> estiver presente, porém vazio, por exemplo, -C "", o diretório de trabalho atual permanecerá inalterado.

Esta opção afeta as opções que esperam o nome do caminho, como --git-dir e work-tree, pois as suas interpretações dos nomes dos caminhos seriam feitas em relação ao diretório de trabalho causado pela opção -C. Como, por exemplo, as seguintes invocações são equivalentes:

```
git --git-dir=a.git --work-tree=b -C c status
git --git-dir=c/a.git --work-tree=c/b status
```

-c <nome>=<valor>

Encaminhe um parâmetro de configuração para o comando O valor informado substituirá os valores dos arquivos de configuração. Um <nome> é esperado no mesmo formato listado pelo comando git config (sub chaves separadas por pontos).

Note que ao omitir = no comando `git -c foo.bar ...` é permitido e define `foo.bar` com o valor booleano verdadeiro (assim como `[foo]bar` faria em um arquivo de configuração). Incluindo os iguais, porém com um valor vazio (como `git -c foo.bar= ...`) define `foo.bar` para a string vazia que `git config --type=bool` converterá para `false`.

`--exec-path[=<caminho>]`

O caminho para onde os seus principais programas Git estão instalados. Isso também pode ser controlado configurando a variável de ambiente `GIT_EXEC_PATH`. Caso nenhum caminho seja informado, o *git* imprimirá a configuração atual e encerrará.

`--html-path`

Imprima o caminho, sem barra, onde a documentação HTML do Git está instalada e encerre.

`--man-path`

Imprima o manpath (consulte `man(1)`) para as páginas do manual desta versão do Git e encerre.

`--info-path`

Imprima o caminho onde os arquivos Info que documentam esta versão do Git estão instalados e encerre.

`-p`

`--paginate`

Canalize toda a saída para *less* (ou caso esteja definido, `$PAGER`) caso a saída padrão seja um terminal. Isso substitui as opções de configuração `pager.<cmd>` (consulte a seção "Mecanismo de Configuração" abaixo).

`-p`

`--no-pager`

Não canalize a saída do Git para um pager.

`--git-dir=<caminho>`

Define o caminho para o repositório (o diretório `".git"`). Isso também pode ser controlado pela configuração da variável de ambiente `GIT_DIR`. Pode ser um caminho absoluto ou relativo ao diretório de trabalho atual.

A especificação do local do diretório `".git"` usando esta opção (ou a variável de ambiente `GIT_DIR`) desativa a descoberta do repositório que tenta localizar um diretório com o `".git"` dentro (que é como o repositório e o nível mais alto da descoberta da árvore de trabalho) e informa ao Git que você está no nível mais alto da árvore de trabalho. Caso não esteja no diretório no nível mais alto da árvore de trabalho, deve informar ao Git onde está este nível da árvore de trabalho, com a opção `--work-tree=<caminho>` (ou a variável de ambiente `GIT_WORK_TREE`)

Caso queira executar o git como se tivesse sido iniciado em `<caminho>`, utilize `git -C <caminho>`.

`--work-tree=<caminho>`

Define o caminho para a árvore de trabalho. Pode ser um caminho absoluto ou relativo ao diretório de trabalho atual. Também pode ser controlado através da configuração da variável de ambiente `GIT_WORK_TREE` e da variável de configuração `core.worktree` (consulte `core.worktree` no [git-config\[1\]](#) para uma discussão com mais detalhes).

`--namespace=<caminho>`

Define o espaço de nomes no Git. Para mais detalhes consulte [gitnamespaces\[7\]](#). É o mesmo que configurar a variável de ambiente GIT\_NAMESPACE.

--super-prefix=<caminho>

Apenas para uso interno. Set a prefix which gives a path from above a repository down to its root. One use is to give submodules context about the superproject that invoked it.

--bare

Trate o repositório como um repositório simples. Caso o ambiente GIT\_DIR não estiver definido, ele será definido no diretório de trabalho atual.

--no-replace-objects

Não utilize substituições "ref" para substituir os objetos Git. Para mais informações consulte [git-replace\[1\]](#).

--literal-pathsspecs

Trate os pathspecs literalmente (ou seja, sem "globbing", sem a magia do "pathspec"). É o mesmo que definir a variável de ambiente GIT\_LITERAL\_PATHSPECS como 1.

--glob-pathsspecs

Adicione a magia *glob* para todos os *pathspec*. É como definir a variável de ambiente GIT\_GLOB\_PATHSPECS como 1. A desativação do caractere curinga nos pathspecs individuais podem ser feitos utilizando a mágica do pathspec ":(literal)"

--noglob-pathsspecs

Adicione a magia *literal* a todos os "pathspec". É equivalente a definir a variável de ambiente GIT\_NOGLOB\_PATHSPECS para 1. A ativação dos caracteres curinga nos "pathspecs" individuais podem ser feitos utilizando a mágica do pathspec ":(glob)"

--icase-pathsspecs

Adicione a magia *icase* em todos os pathspec. É como definir a variável de ambiente GIT\_ICASE\_PATHSPECS como 1.

--no-optional-locks

Não execute operações opcionais que exijam bloqueios. Isso é equivalente que definir o GIT\_OPTIONAL\_LOCKS como 0.

--list-cmds=group[,group...]

Liste os comandos por grupo. Essa é uma opção interna/experimental e pode mudar ou ser removido no futuro. Os grupos compatíveis são: *builtins*, *parseopt* (comandos internos que utilizam *parse-options*), *main* (todos os comandos no diretório *'libexec'*), *others* (todos os outros comandos no \$PATH que possuem um prefixo git), *list- <categoria>* (consulte as categorias no *command-list.txt*), *nohelpers* (exclua os comandos auxiliares), *alias* e *config* (recupera a lista dos comandos da variável *completion.commands*)

## OS COMANDOS DO GIT

Dividimos o Git em comandos de alto nível ("porcelana") e de baixo nível ("encanamento").

# Comandos de alto nível (porcelana)

Separamos os comandos porcelana nos comandos principais e em alguns utilitários auxiliares do usuário.

## Os principais comandos porcelana

[git-add\[1\]](#)

Adiciona o conteúdo dos arquivos ao índice.

[git-am\[1\]](#)

Aplica uma série de patches vindos de uma caixa de e-mails.

[git-archive\[1\]](#)

Cria um histórico dos arquivos a partir de uma determinada árvore.

[git-bisect\[1\]](#)

Utilize a procura binária para localizar o commit que introduziu um bug.

[git-branch\[1\]](#)

Lista, cria ou exclui os ramos.

[git-bundle\[1\]](#)

Mova os objetos e as refs através do histórico.

[git-checkout\[1\]](#)

Mova os ramos ou restaura os arquivos da árvores de trabalho.

[git-cherry-pick\[1\]](#)

Aplica as mudanças introduzidas por alguns commits já existentes.

[git-citool\[1\]](#)

Uma alternativa gráfica ao git-commit.

[git-clean\[1\]](#)

Remove os arquivos da árvore de trabalho sem monitoramento.

[git-clone\[1\]](#)

Clona um repositório em um novo diretório.

[git-commit\[1\]](#)

Grava as alterações para o repositório.

[git-describe\[1\]](#)

Dá a um objeto um nome em um formato legível para humanos com base em um "ref" disponível.

[git-diff\[1\]](#)

Exiba as alterações entre os commits, o commit, árvore de trabalho, etc.

[git-fetch\[1\]](#)

Faz o download dos objetos e dos refs vindo de outro repositório.

[git-format-patch\[1\]](#)

Prepara os patches para serem enviados por e-mail.

[git-gc\[1\]](#)

Exclui os arquivos desnecessários e otimiza o repositório local.

[git-grep\[1\]](#)

Imprima as linhas que coincidam com um padrão.

[git-gui\[1\]](#)

Uma interface gráfica portátil para o Git.

[git-init\[1\]](#)

Cria um repositório vazio para o Git ou reinicializa um repositório já existente.

[git-log\[1\]](#)

Exibe os registros logs de um commit.

[git-merge\[1\]](#)

Une dois ou mais históricos de desenvolvimento juntos.

[git-mv\[1\]](#)

Move ou renomeia um arquivo, um diretório ou um link simbólico.

[git-notes\[1\]](#)

Adiciona ou inspeciona as anotações dos objetos.

[git-pull\[1\]](#)

Capture de e o integre com um outro repositório ou em um outro ramo local.

[git-push\[1\]](#)

Atualiza os refs remotos através da associação dos objetos.

[git-range-diff\[1\]](#)

Compara os dois intervalos de um commit (duas versões de um ramo por exemplo).

[git-rebase\[1\]](#)

Reaplica os commits no topo do outro cume da base.

[git-reset\[1\]](#)

Redefine o HEAD atual para o para a condição determinada.

### [git-restore\[1\]](#)

Restaura as árvores de trabalho.

### [git-revert\[1\]](#)

Reverte alguns commits já existentes.

### [git-rm\[1\]](#)

Remove os arquivos da árvore de trabalho e do índice.

### [git-shortlog\[1\]](#)

Faça um resumo da saída do *git log*.

### [git-show\[1\]](#)

Exiba os vários tipos de objetos.

### [git-spars checkout\[1\]](#)

Alter e inicialize a averiguação esparsa.

### [git-stash\[1\]](#)

Armazene as alterações em um diretório fora do diretório de trabalho.

### [git-status\[1\]](#)

Exiba a condição atual da árvore de trabalho.

### [git-submodule\[1\]](#)

Inicializa, atualiza ou inspeciona submódulos.

### [git-switch\[1\]](#)

Alterna entre os ramos.

### [git-tag\[1\]](#)

Cria, lista, exclui ou verifica uma tag do objeto com assinatura GPG.

### [git-worktree\[1\]](#)

Manipula as diversas árvores de trabalho.

### [gitk\[1\]](#)

O navegador do repositório Git.

## **Comandos Auxiliares**

Manipuladores:

### [git-config\[1\]](#)

Obtém e define os repositórios ou as opções globais.

[git-fast-export\[1\]](#)

Exportador de dados do Git.

[git-fast-import\[1\]](#)

Estrutura para os importadores de dados rápidos do Git.

[git-filter-branch\[1\]](#)

Reescreve os ramos.

[git-mergetool\[1\]](#)

Executa as ferramentas de resolução de problemas para resolver os conflitos de mesclagem.

[git-pack-refs\[1\]](#)

Empacota os cabeçalhos e as tags para um acesso eficiente ao repositório.

[git-prune\[1\]](#)

Corta todos os objetos fora de alcance do banco de dados de objetos.

[git-reflog\[1\]](#)

Gerencia as informações do reflog.

[git-remote\[1\]](#)

Gerencia o conjunto de repositórios monitorados.

[git-repack\[1\]](#)

Empacota os objetos não empacotados em um repositório.

[git-replace\[1\]](#)

Cria, lista e exclui os refs para a reposição dos objetos.

Interrogadores:

[git-annotate\[1\]](#)

Anota as linhas com as informações do commit.

[git-blame\[1\]](#)

Exibe quais foram as últimas modificações feitas em cada linha de um arquivo separados pela versão da revisão e do autor da modificação.

[git-bugreport\[1\]](#)

Colete informações para que o usuário envie um relatório de erro.

[git-count-objects\[1\]](#)

Conta a quantidade dos objetos que não foram empacotados e seu consumo no disco.

[git-difftool\[1\]](#)



Exibe as mudanças utilizando ferramentas diff tradicionais.

[git-fsck\[1\]](#)

Verifica a conectividade e validade dos objetos em um banco de dados.

[git-help\[1\]](#)

Exiba a informação de ajuda sobre o Git.

[git-instaweb\[1\]](#)

Navegue instantaneamente no seu repositório de trabalho com o gitweb.

[git-merge-tree\[1\]](#)

Exiba as três maneiras de mesclagem sem mexer no índice.

[git-rerere\[1\]](#)

Reutilize uma resolução gravada das mesclagens conflitantes.

[git-show-branch\[1\]](#)

Exiba os ramos e seus respectivos commits.

[git-verify-commit\[1\]](#)

Verifique a assinatura GPG dos commits.

[git-verify-tag\[1\]](#)

Verifique a assinatura GPG das tags.

[git-whatchanged\[1\]](#)

Exiba os registros logs com a diferença entre a introdução de cada commit.

[gitweb\[1\]](#)

Interface web do Git (frontend web para os repositórios Git).

## **Interagindo com os outros**

Estes comandos são para interagir com um SCM externo e com as outras pessoas através de patch por e-mail.

[git-archimport\[1\]](#)

Importa um repositório GNU Arch no Git.

[git-cvsexportcommit\[1\]](#)

Exporta um único commit para uma averiguação do CVS.

[git-cvsimport\[1\]](#)

Recupera os seus dados vindos de outros SCM que as pessoas adoram odiar.

[git-cvsserver\[1\]](#)

Um emulador de um servidor CVS para o Git.

[git-imap-send\[1\]](#)

Envia uma coleção de patches da entrada padrão para um diretório IMAP.

[git-p4\[1\]](#)

Importa de e submete aos repositório Perforce.

[git-quiltimport\[1\]](#)

Aplica um conjunto de patches no ramo atual.

[git-request-pull\[1\]](#)

Gera um resumo com as modificações pendentes.

[git-send-email\[1\]](#)

Envia uma coleção de patches como sendo e-mails.

[git-svn\[1\]](#)

Operação bidirecional entre um repositório do Subversion e do Git.

## Redefina, restaure e reverta

Existem três comandos com nomes semelhantes: `git reset`, `git restore` e o `git revert`.

- [git-revert\[1\]](#) trata de fazer um novo commit que reverte as alterações feitas por outros commit.
- [git-restore\[1\]](#) trata da restauração dos arquivos na árvore de trabalho do índice ou de outro commit. Este comando não atualiza o seu ramo. O comando também pode ser usado para restaurar os arquivos no índice do outro commit.
- [git-reset\[1\]](#) trata da atualização do seu ramo, movendo o topo para adicionar ou remover os commits do ramo. Esta operação altera o histórico do commit.

O comando `git reset` também pode ser usado para restaurar o índice, sobrepondo com `git restore`.

## Comandos de baixo nível (encanamento *plumbing*)

Embora o Git inclua a sua própria camada de porcelana, os seus comandos de baixo nível são suficientes para apoiar o desenvolvimento de porcelanas alternativas. Os desenvolvedores destas porcelanas podem começar lendo sobre [git-update-index\[1\]](#) e [git-read-tree\[1\]](#).

A interface (entrada, saída, conjunto de opções e as semânticas) para esses comandos de baixo nível deve ser muito mais estável que os comandos porcelana, porque estes comandos são principalmente para uso com um script. A interface para os comandos Porcelana, por outro lado, está sujeita a alterações para melhorar a experiência do usuário final.

A descrição a seguir divide os comandos de baixo nível em comandos que manipulam os objetos (no repositório, índice e árvore de trabalho), comandos que interrogam, comparam objetos, comandos que movem objetos e suas referências entre os repositórios.

## Comandos de manipulação

[git-apply\[1\]](#)

Aplicar um patch nos arquivos e/ou ao índice.

[git-checkout-index\[1\]](#)

Copiar os arquivos do índice para a árvore de trabalho.

[git-commit-graph\[1\]](#)

Escrever e verificar os arquivos commit-graph do Git.

[git-commit-tree\[1\]](#)

Criar um novo objeto commit.

[git-hash-object\[1\]](#)

Calcular o ID do objeto e opcionalmente criar uma bolha vinda de um arquivo.

[git-index-pack\[1\]](#)

Construir um pacote índice do arquivo para um arquivo de pacote já existente.

[git-merge-file\[1\]](#)

Executar a mesclagem de um arquivo de três vias.

[git-merge-index\[1\]](#)

Executar uma mesclagem para os arquivos que precisam ser mesclados.

[git-mktag\[1\]](#)

Criar um objeto tag.

[git-mktree\[1\]](#)

Criar uma árvore-objeto de um texto com formatação ls-tree.

[git-multi-pack-index\[1\]](#)

Escrever e verificar os diversos índices dos pacotes.

[git-pack-objects\[1\]](#)

Criar um histórico empacotado dos objetos.

[git-prune-packed\[1\]](#)

Remover os objetos extras que estejam atualmente nos arquivos empacotados.

[git-read-tree\[1\]](#)

Ler a informação da árvore no índice.

[git-symbolic-ref\[1\]](#)

Ler, modificar e excluir os refs simbólicos.

[git-unpack-objects\[1\]](#)

Desempacota os objetos de um arquivo empacotado.

[git-update-index\[1\]](#)

Registra o conteúdo de um arquivo na árvore de trabalho para o índice.

[git-update-ref\[1\]](#)

Atualiza o nome do objeto armazenado em um "ref" de forma segura.

[git-write-tree\[1\]](#)

Cria um objeto árvore com base no índice atual.

## Comandos de interrogação

[git-cat-file\[1\]](#)

Proporciona o conteúdo, o tipo e a informação sobre o tamanho dos objetos no repositório.

[git-cherry\[1\]](#)

Procura os commits que ainda serão aplicados ao "upstream".

[git-diff-files\[1\]](#)

Compara os arquivos na árvore de trabalho e no índice.

[git-diff-index\[1\]](#)

Compara uma árvore com o diretório de trabalho ou índice.

[git-diff-tree\[1\]](#)

Compara o conteúdo e o modo das bolhas encontrados através de dois objetos da árvore.

[git-for-each-ref\[1\]](#)

Informação de saída de cada "ref".

[git-get-tar-commit-id\[1\]](#)

Extrai o ID do commit de um arquivo criado utilizando o *git-archive*.

[git-ls-files\[1\]](#)

Exiba a informação sobre os arquivos no índice e na árvore de trabalho.

[git-ls-remote\[1\]](#)

Lista as referências em um repositório remoto.

[git-ls-tree\[1\]](#)

Lista o conteúdo de uma árvore de objetos.

[git-merge-base\[1\]](#)

Localize os melhores ancestrais possíveis para fazer uma mesclagem.

[git-name-rev\[1\]](#)

Localize os nomes simbólicos para os "revs" que foram informados.

[git-pack-redundant\[1\]](#)

Localiza os arquivos "pack" que forem redundantes.

[git-rev-list\[1\]](#)

Lista os objetos de commit em ordem cronológica reversa.

[git-rev-parse\[1\]](#)

Escolha e modele os parâmetros.

[git-show-index\[1\]](#)

Exiba o índice do arquivo empacotado.

[git-show-ref\[1\]](#)

Lista as referências em um repositório local.

[git-unpack-file\[1\]](#)

Cria um arquivo temporário com conteúdos bolha.

[git-var\[1\]](#)

Exiba uma variável lógica local para o Git.

[git-verify-pack\[1\]](#)

Valide os arquivos empacotados do Git.

Em geral, os comandos de interrogação não tocam nos arquivos da árvore de trabalho.

## **Sincronizando os repositórios**

[git-daemon\[1\]](#)

Um servidor realmente simples para os repositórios Git.

[git-fetch-pack\[1\]](#)

Recebe os objetos que faltam de um outro repositório.

[git-http-backend\[1\]](#)

Implementação do lado do servidor do Git através do HTTP.

[git-send-pack\[1\]](#)

impulsiona os objetos sob o protocolo Git de um outro repositório.

[git-update-server-info\[1\]](#)

Atualiza a informação auxiliar sobre o arquivo para ajudar os servidores burros.

A seguir, são apresentados os comandos auxiliares utilizados acima; os usuários finais normalmente não os utilizam diretamente.

[git-http-fetch\[1\]](#)

Faz o download de um repositório remoto Git através do HTTP.

[git-http-push\[1\]](#)

Impulsiona (push) os objetos através do HTTP/DAV para um outro repositório.

[git-parse-remote\[1\]](#)

Rotinas para auxiliar na análise dos parâmetros de acesso ao repositório remoto.

[git-receive-pack\[1\]](#)

Receba o que é impulsionado ao repositório.

[git-shell\[1\]](#)

Shell de login restrito para acesso somente com o SSH do Git.

[git-upload-archive\[1\]](#)

Envia um arquivo de volta ao git-archive.

[git-upload-pack\[1\]](#)

Envia os objetos compactados para o git-fetch-pack.

## Comandos auxiliares internos

Estes são comandos auxiliares internos usados por outros comandos; os usuários finais normalmente não os utilizam diretamente.

[git-check-attr\[1\]](#)

Exiba a informação do gitattributes.

[git-check-ignore\[1\]](#)

Depure o gitignore / exclua os arquivos.

[git-check-mailmap\[1\]](#)

Exiba os nomes canônicos e os endereços de e-mail dos contatos.

[git-check-ref-format\[1\]](#)

Certifique que um nome de uma referência está bem formado.

[git-column\[1\]](#)

Exiba os dados em colunas.

[git-credential\[1\]](#)

Obtém e guarda as credenciais dos usuários.

[git-credential-cache\[1\]](#)

Auxiliar para armazenar as senhas temporariamente na memória.

[git-credential-store\[1\]](#)

Auxiliar para armazenar as credenciais no disco.

[git-fmt-merge-msg\[1\]](#)

Gera uma mensagem de mesclagem do commit.

[git-interpret-trailers\[1\]](#)

Adiciona ou analisa as informações estruturadas nas mensagens do commit.

[git-mailinfo\[1\]](#)

Extraí a correção e o autor de uma única mensagem de e-mail.

[git-mailsplit\[1\]](#)

Programa simples para dividir o mbox UNIX.

[git-merge-one-file\[1\]](#)

O programa assistente predefinido de ajuda para usar com o `git-merge-index`.

[git-patch-id\[1\]](#)

Compute um ID único para um patch.

[git-sh-i18n\[1\]](#)

Código da configuração do i18n do Git para scripts shell.

[git-sh-setup\[1\]](#)

Código da configuração comum do script shell do Git.

[git-strip-space\[1\]](#)

Remove os espaços desnecessários.

## Mecanismo de Configuração

O Git utiliza um formato de texto simples para armazenar as personalizações por repositório e por usuário. Tal arquivo de configuração pode ficar assim:

```
#
# Os caracteres '#' ou ';' indicam um comentário.
#

; variáveis principais
[core]
    ; Não confie nos modos dos arquivos
    filemode = false

; identidade do usuário
[user]
    name = "Junio C Hamano"
    email = "gitster@pobox.com"
```

Vários comandos são lidos no arquivo de configuração e ajustam a sua operação de acordo. Para obter uma lista e mais detalhes sobre o mecanismo de configuração, consulte [git-config\[1\]](#).

## Terminologia do Identificador

<objeto>

Indica o nome do objeto para qualquer tipo de objeto.

<blob>

Indica um nome de um objeto bolha.

<árvore>

Indica um nome de um objeto árvore.

<commit>

Indica um nome de um objeto commit.

<tree-ish>

Indica uma árvore, nome de um objeto commit ou tag. Um comando que aceite um argumento <commit-ish> e queira operar em um objeto <commit>, porém remove a referência automaticamente dos objetos <tag> que apontem para um <commit>.

<commit-ish>

Indica um nome do objeto commit ou tag. Um comando que aceite um argumento <commit-ish> e queira operar em um objeto <commit>, porém remove a referência automaticamente dos objetos <tag> que apontem para um <commit>.

<tipo>

Indica que um tipo do objeto seja necessário. Atualmente um dos: blob, tree, commit, ou tag.

<arquivo>

Indica um nome do arquivo - quase sempre em relação à raiz da estrutura da árvore que o GIT\_INDEX\_FILE descreve.

## Identificadores Simbólicos

Qualquer comando Git que aceite qualquer <objeto> também pode utilizar a seguinte notação simbólica:

HEAD

indica o cabeçalho do ramo atual.

<tag>

uma tag válida *nome* (como, por exemplo, uma referência refs/tags/<tag>).

<head>

um cabeçalho válido *nome* (como, por exemplo, uma referência refs/heads/<head>).



Para obter uma lista mais completa de maneiras de soletrar os nomes dos objetos, consulte a seção "DEFININDO AS REVISÕES" em [gitrevisions\[7\]](#).

## A Estrutura dos Arquivos/Diretórios

Favor consultar o documento [gitrepository-layout\[5\]](#).

Para mais detalhes sobre cada gancho, consulte [githooks\[5\]](#).

Os SCMs de alto nível podem fornecer e gerenciar informações adicionais no \$GIT\_DIR.

## Terminologia

Favor consultar [gitglossary\[7\]](#).

## As Variáveis do Ambiente

Vários comandos Git usam as seguintes variáveis de ambiente:

### O Repositório Git

Essas variáveis de ambiente se aplicam a *todos* os comandos principais do Git. Nb: é importante notar que eles podem ser usados/substituídos pelo SCMS acima do Git, portanto, tenha cuidado caso esteja usando um front-end externo.

#### GIT\_INDEX\_FILE

Este ambiente permite a definição de um arquivo índice alternativo. Caso não seja definido, a predefinição do \$GIT\_DIR/index é utilizado.

#### GIT\_INDEX\_VERSION

Essa variável de ambiente permite a especificação de uma versão de índice para os novos repositórios. Não afetará os arquivos de índice existentes. Por predefinição, a versão 2 ou 3 do arquivo de índice é utilizado. Para mais informações consulte [git-update-index\[1\]](#).

#### GIT\_OBJECT\_DIRECTORY

Caso o diretório de armazenamento dos objetos seja informado através desta variável de ambiente, os diretórios `sh1` serão criados embaixo - caso contrário, o diretório predefinido \$GIT\_DIR/objects será utilizado.

#### GIT\_ALTERNATE\_OBJECT\_DIRECTORIES

Devido à natureza imutável dos objetos Git, os objetos antigos podem ser arquivados em diretórios compartilhados com somente leitura apenas. Esta variável especifica uma lista ":" separada (no Windows ";") dos diretórios dos objetos Git que podem ser utilizados para localizar objetos Git. Os novos objetos não serão gravados nestes diretórios.

As entradas que começam com " (aspas duplas) serão interpretadas como caminhos entre as aspas no estilo C, removendo as aspas duplas iniciais e finais, respeitando as escapes da barra invertida. Como por exemplo, o valor "path-with-\"-and-:-in-it":vanilla-path possui dois caminhos: path-with-\"-and-:-in-it e vanilla-path.

#### GIT\_DIR

Caso a variável de ambiente `GIT_DIR` esteja definida, ela definirá um caminho que será utilizado em vez do `.git` predefinido como sendo a base do repositório. A opção da linha de comando `--git-dir` também define este valor.

#### `GIT_WORK_TREE`

Defina o caminho para a raiz da árvore de trabalho. Isso também pode ser controlado pela opção da linha de comando `--work-tree` e pela variável de configuração `core.worktree`.

#### `GIT_NAMESPACE`

Define o espaço de nomes no Git; para mais detalhes consulte [gitnamespaces\[7\]](#). A opção da linha de comando `--namespace` também define este valor.

#### `GIT_CEILING_DIRECTORIES`

Essa deve ser uma lista separada por dois pontos com caminhos absolutos. Caso seja definido, é uma lista dos diretórios onde o Git não deve mudar de diretório (`chdir`) enquanto procura um diretório do repositório (útil para excluir os diretórios da rede com carregamento lento). Ele não excluirá o diretório de trabalho atual, um `GIT_DIR` definido na linha de comandos ou no ambiente. Normalmente, o Git precisa ler as entradas nesta lista e resolver qualquer link simbólico que possa estar presente para compará-las com o diretório atual. No entanto, mesmo que esse acesso seja lento, você pode adicionar uma entrada vazia à lista para informar ao Git que as entradas subsequentes não são links simbólicos e não precisam ser resolvidos; como, por exemplo, `GIT_CEILING_DIRECTORIES=/maybe/symlink::/very/slow/non/symlink`.

#### `GIT_DISCOVERY_ACROSS_FILESYSTEM`

Quando executado em um diretório que não possui o diretório do repositório `".git"`, o Git tenta encontrar esse diretório nos diretórios pais para encontrar o cume da árvore de trabalho, porém, é predefinido que, ele não cruze os limites do sistema de arquivos. Essa variável de ambiente pode ser configurada como `true` para dizer ao Git para não parar nos limites do sistema de arquivos. Como o `GIT_CEILING_DIRECTORIES`, isso não afetará explicitamente um diretório do repositório definido através do `GIT_DIR` ou na linha de comando.

#### `GIT_COMMON_DIR`

Se essa variável estiver definida para um caminho, os arquivos que não são da árvore de trabalho, estão normalmente em `$GIT_DIR` e serão obtidos desse caminho. Os arquivos específicos da árvore de trabalho, como `HEAD` ou índice serão obtidos de `$GIT_DIR`. Para mais detalhes consulte [gitrepository-layout\[5\]](#) e [git-worktree\[1\]](#). Essa variável tem precedência mais baixa do que outras variáveis de caminho como `GIT_INDEX_FILE`, `GIT_OBJECT_DIRECTORY`...

#### `GIT_DEFAULT_HASH`

Caso esta variável esteja definida, o algoritmo hash predefinido para os novos repositórios será definido com este valor. Este valor é atualmente ignorado durante a clonagem; em vez disso, a configuração do repositório remoto é utilizada. O valor predefinido é `sha1`.

## Os Commits do Git

#### `GIT_AUTHOR_NAME`

O endereço legível do endereço de e-mail utilizado na identidade do autor ao criar os commits, na tag dos objetos ou ao gravar os reflogs. Substitui as definições de configuração `user.name` e `author.name`.

#### `GIT_AUTHOR_EMAIL`

O endereço de email utilizado na identidade do autor ao criar os commits, na marcação dos objetos ou ao gravar os reflogs. Substitui as definições da configuração `user.email` e `author.email`.

## GIT\_AUTHOR\_DATE

A data utilizada para a identidade do autor ao criar objetos commit ou tags ou quando escrever "reflogs". Para conhecer os formatos válidos, consulte [git-commit\[1\]](#).

## GIT\_COMMITTER\_NAME

O endereço legível do nome utilizado na identidade do autor do commit ao criar os commits, na tag dos objetos ou ao gravar os reflogs. Substitui as definições de configuração `user.name` e `committer.name`.

## GIT\_COMMITTER\_EMAIL

O endereço de email utilizado na identidade do autor ao criar os commits, na marcação dos objetos ou ao gravar os reflogs. Overrides the `user.email` and `committer.email` configuration settings.

## GIT\_COMMITTER\_DATE

A data utilizada para a identidade de quem fez o commit durante a criação dos objetos as tags do commit ou ao gravar os reflogs. Para mais formatos válidos, consulte [git-commit\[1\]](#).

## EMAIL

O endereço de e-mail usado nas identidades do autor e do commit, caso nenhuma outra variável de ambiente ou da configuração relevante tiver sido definida.

# Os Diffs do Git

## GIT\_DIFF\_OPTS

A única opção válida é "--unified=??" ou "-u??" para definir o número de linhas de contexto mostradas quando um diff unificado for criado. Isso tem precedência sobre qualquer valor da opção "-U" ou "--unified" passado na linha de comando diff do Git.

## GIT\_EXTERNAL\_DIFF

Quando a variável de ambiente `GIT_EXTERNAL_DIFF` é configurada, o programa informado nele é chamado, em vez do "diff" descrito acima. Para um caminho que é adicionado, removido ou modificado, a variável `GIT_EXTERNAL_DIFF` é chamado com 7 parâmetros:

```
path old-file old-hex old-mode new-file new-hex new-mode
```

onde:

<old|new>-file

são os arquivos que `GIT_EXTERNAL_DIFF` pode utilizar para ler o conteúdo do <antigo|novo>,

<old|new>-hex

são os hashes SHA-1 com 40 hexadecimais,

<old|new>-mode

são a representação octais dos modos dos arquivos.

Os parâmetros do arquivo podem apontar para o arquivo de trabalho do usuário (`new-file` em "git-diff-files" por exemplo), `/dev/null` (`old-file` quando um novo arquivo for adicionado por exemplo) ou um arquivo temporário (um arquivo antigo no índice por exemplo). A variável `GIT_EXTERNAL_DIFF` não deve se preocupar com o desvinculamento do arquivo temporário --- ele é removido quando a variável `GIT_EXTERNAL_DIFF` termina.

Para um caminho que não foi mesclado, `GIT_EXTERNAL_DIFF` é chamado com 1 parâmetro, <caminho>.

Para cada caminho `GIT_EXTERNAL_DIFF` que é chamado, duas variáveis de ambiente, `GIT_DIFF_PATH_COUNTER` e `GIT_DIFF_PATH_TOTAL` são definidas.

`GIT_DIFF_PATH_COUNTER`

Um contador com base 1 incrementado por um em cada caminho.

`GIT_DIFF_PATH_TOTAL`

A quantidade total dos caminhos.

## Outros

`GIT_MERGE_VERBOSE`

Um número que controla a quantidade de saída demonstrada pela estratégia de mesclagem recursiva. Substitui o `merge.verbose`. Consulte [git-merge\[1\]](#)

`GIT_PAGER`

Essa variável de ambiente substitui o `$PAGER`. Caso esteja definido como uma string vazia ou com o valor "cat", o Git não iniciará um pager. Consulte também a opção `core.askPass` no [git-config\[1\]](#).

`GIT_PROGRESS_DELAY`

Um número que controla quantos segundos atrasar antes de mostrar os indicadores opcionais do progresso. A predefinição retorna para 2.

`GIT_EDITOR`

Essa variável de ambiente substitui o `$EDITOR` e o `$VISUAL`. É usado por vários comandos Git quando, no modo interativo, um editor deve ser iniciado. Consulte também [git-var\[1\]](#) e a opção `core.editor` no [git-config\[1\]](#).

`GIT_SSH`

`GIT_SSH_COMMAND`

Caso alguma destas variáveis de ambiente esteja definida, o comando *git fetch* e o *git push* utilizarão o comando informado em vez do *ssh* quando precisarem se conectar com um sistema remoto. Os parâmetros da linha de comando passados para o comando configurado, são determinados pela variante *ssh*. Para mais detalhes consulte a opção de configuração `ssh.variant` no [git-config\[1\]](#).

A variável `$GIT_SSH_COMMAND` tem precedência sobre a variável `$GIT_SSH` que é interpretado pelo shell, permite que argumentos adicionais sejam incluídos. A variável `$GIT_SSH`, por outro lado, deve ser apenas o caminho para um programa (que pode ser um script shell do wrapper, caso as opções adicionais sejam necessárias).

Geralmente é mais fácil configurar as opções desejadas através do seu arquivo pessoal `.ssh/config`. Consulte a documentação do *ssh* para obter mais detalhes.

`GIT_SSH_VARIANT`

Se esta variável de ambiente estiver configurada, ela substitui a detecção automática do Git, caso `GIT_SSH/GIT_SSH_COMMAND/core.sshCommand` se refere ao OpenSSH, plink ou tortoiseplink. Esta variável substitui a configuração `ssh.variant` que serve ao mesmo propósito.

`GIT_ASKPASS`

Caso esta variável do ambiente esteja definida, os comandos Git que precisam obter as senhas ou as frases secretas (para a autenticação HTTP ou IMAP por exemplo) chamarão esse programa com um

prompt adequado como argumento da linha de comando e irão ler a senha em seu STDOUT. Consulte também a opção `core.askPass` no [git-config\[1\]](#).

#### GIT\_TERMINAL\_PROMPT

Caso esta variável de ambiente esteja definida como `0`, o git não será solicitado no terminal (ao solicitar uma autenticação HTTP por exemplo).

#### GIT\_CONFIG\_NOSYSTEM

Independente se você ignore as configurações de leitura do arquivo `$(prefix)/etc/gitconfig` do sistema. Essa variável de ambiente pode ser usada junto com `$HOME` e o `$XDG_CONFIG_HOME` para criar um ambiente previsível para um script exigente, ou você pode configurá-la temporariamente para evitar o uso de um arquivo `/etc/gitconfig` com problemas, enquanto aguarda alguém com permissões suficientes para corrigi-lo.

#### GIT\_FLUSH

Caso esta variável de ambiente estiver definida como `"1"`, então os comandos como *git blame* (no modo incremental), *git rev-list*, *git log*, *git check-attr* e *git check-ignore* serão impor uma descarga do fluxo gerado depois que cada registro tenham sido esvaziado. Se essa variável for definida como `0`, a saída destes comandos será feita utilizando toda a E/S na memória intermédia (buffer). Se essa variável de ambiente não seja definida, o Git escolherá a descarga em uma memória intermédia ou orientada no esvaziamento dos registros, para ver se o stdout parece ter sido redirecionado para um arquivo ou não.

#### GIT\_TRACE

Ativa o rastreo geral das mensagens, como por exemplo expansão do pseudônimo, execução interna dos comandos e a execução externa dos comandos.

Caso esta variável esteja definida como `1`, `2` ou `true` (a comparação não diferencia as maiúsculas das minúsculas), as mensagens de rastreo serão impressas no stderr.

Caso a variável seja configurada com um valor inteiro maior que `2` e menor que `10` (estritamente), o Git interpretará este valor como um descritor de arquivo aberto e tentará gravar as mensagens de monitoramento neste descritor do arquivo.

Como alternativa, caso a variável estiver definida como um caminho absoluto (começando com um caractere `/`), o Git interpretará isso como um caminho do arquivo e tentará anexar as mensagens de rastreo nelas.

Desativar a variável ou defini-la como vazia `0` ou *false* (não faz distinção entre maiúsculas e minúsculas) desativa as mensagens de monitoramento.

#### GIT\_TRACE\_FSMONITOR

Ativa as mensagens de rastreamento para a extensão do monitor do sistema de arquivos. Consulte `GIT_TRACE` para conhecer opções de saída de rastreo disponíveis.

#### GIT\_TRACE\_PACK\_ACCESS

Permite rastrear as mensagens para todos os acessos para qualquer pacote. Para cada acesso, é registrado o nome do arquivo do pacote e um *offset*. Pode ser útil para solucionar alguns problemas de desempenho relacionados ao pacote. Consulte `GIT_TRACE` para conhecer opções de saída de rastreo disponíveis.

#### GIT\_TRACE\_PACKET

Ativa o rastreo das mensagens para todos os pacotes que entram ou saem de um determinado programa. Isso pode ajudar na depuração da negociação dos objetos ou de outros problemas de protocolo. O rastreamento é desativado em um pacote que comece com `"PACK"` (porém

consulte `GIT_TRACE_PACKFILE` abaixo). Consulte `GIT_TRACE` para conhecer opções de saída de rastreo disponíveis.

#### `GIT_TRACE_PACKFILE`

Permite o monitoramento dos arquivos dos pacotes enviados ou recebidos através de um determinado programa. Diferente de outras saídas monitoradas, esse monitoramento é literalmente: sem cabeçalhos e sem a citação dos dados binários. Você quase que certamente vai querer direcionar para um arquivo (`GIT_TRACE_PACKFILE=/tmp/my.pack` por exemplo) em vez de exibi-lo no terminal ou misturá-lo com uma outra saída monitorada.

Observe que atualmente isso é implementado apenas para o lado do cliente dos clones e das buscas.

#### `GIT_TRACE_PERFORMANCE`

Ativa as mensagens de rastreamento relacionadas ao desempenho, como por exemplo, o tempo total da execução de cada comando Git. Consulte `GIT_TRACE` para conhecer opções de saída de rastreo disponíveis.

#### `GIT_TRACE_SETUP`

Permite que as mensagens de rastreamento imprimam o `.git`, a árvore de trabalho e o diretório de trabalho atual após o Git concluir a sua fase de configuração. Consulte `GIT_TRACE` para conhecer opções de saída de rastreo disponíveis.

#### `GIT_TRACE_SHALLOW`

Ativa o rastreo das mensagens que podem ajudar na depuração da busca/clonagem dos repositórios rasos. Consulte `GIT_TRACE` para conhecer opções de saída de rastreo disponíveis.

#### `GIT_TRACE_CURL`

Permite um rastreamento curl completo de todos os dados que foram recebidos e enviados, incluindo as informações descritivas, do protocolo de transporte git. É semelhante a fazer `curl --trace-ascii` na linha de comando. Consulte `GIT_TRACE` para conhecer as opções disponíveis geradas pelo rastreo.

#### `GIT_TRACE_CURL_NO_DATA`

Quando um rastreamento curl está ativado (consulte `GIT_TRACE_CURL` acima), não despeje os dados (ou seja, apenas despeje as linhas de informações e os cabeçalhos).

#### `GIT_TRACE2`

Permite mensagens de rastreamento com mais detalhes através da biblioteca "trace2". A saída do `GIT_TRACE2` é um formato simples de texto para facilitar a leitura das pessoas.

Caso esta variável esteja definida como 1, 2 ou true (a comparação não diferencia as maiúsculas das minúsculas), as mensagens de rastreo serão impressas no stderr.

Caso a variável seja configurada com um valor inteiro maior que 2 e menor que 10 (estritamente), o Git interpretará este valor como um descritor de arquivo aberto e tentará gravar as mensagens de monitoramento neste descritor do arquivo.

Alternativamente, caso a variável esteja definida como um caminho absoluto (começando com um caractere `/`), Git interpretará isso como um caminho de arquivo e tentará anexar as mensagens de rastreamento a ela. Caso o caminho já exista e for um diretório, as mensagens de rastreamento serão gravadas em arquivos (uma por processo) nesse diretório, nomeada de acordo com o último componente do SID e um contador opcional (para evitar colisões de nome de arquivo).

Além disso, caso a variável esteja definida como `af_unix:[<socket_type>:]<caminho-absoluto>`, o Git tentará abrir o caminho como um soquete de domínio Unix. O tipo de soquete pode ser `stream` ou `dgram`.

Desativar a variável ou defini-la como vazia *0* ou *false* (não faz distinção entre maiúsculas e minúsculas) desativa as mensagens de monitoramento.

Para mais detalhes, consulte [Trace2 documentation](#).

#### GIT\_TRACE2\_EVENT

Esta configuração registra um formato com base no JSON que é adequado para a interpretação da máquina. Consulte GIT\_TRACE2 para conhecer as opções disponíveis para o monitoramento e o link:technical/api-trace2.html [documentação do Trace2] para obter todos os detalhes.

#### GIT\_TRACE2\_PERF

Além das mensagens texto disponíveis em GIT\_TRACE2, esta configuração escreve o formato da base da coluna para compreender as regiões aninhadas. Consulte GIT\_TRACE2 para conhecer as opções disponíveis para o monitoramento e o link:technical/api-trace2.html [documentação do Trace2] para obter todos os detalhes.

#### GIT\_TRACE\_REDACT

É predefinido que quando o monitoramento seja ativado, o Git redita os valores dos cookies, o cabeçalho "Autorização:" e o cabeçalho "Autorização do proxy:". Defina esta variável como 0 para evitar esta redação.

#### GIT\_LITERAL\_PATHSPECS

Definir essa variável como 1 fará com que o Git trate todos os pathspecs de forma literal, e não como padrões glob. Como, por exemplo, a execução do GIT\_LITERAL\_PATHSPECS=1 git log -- '\*.c' procurará pelos commits que tocam no caminho \*.c e não nos caminhos que coincidem com o agrupamento \*.c. Você pode querer isso caso esteja alimentando caminhos literais para o Git (como, por exemplo, os caminhos informados anteriormente a você pelo git ls-tree, --raw, saída do diff, etc).

#### GIT\_GLOB\_PATHSPECS

Definir essa variável como 1 fará com que o Git trate todos os pathspecs como padrões "glob" (também informados como "glob" mágico).

#### GIT\_NOGLOB\_PATHSPECS

Definir essa variável como 1 fará com que o Git trate todos os pathspecs como literal (também informados como mágica "literal").

#### GIT\_ICASE\_PATHSPECS

Definir essa variável como 1 fará com que o Git trate todos os pathspecs como indiferente para maiúsculas e minúsculas.

#### GIT\_REFLOG\_ACTION

Quando uma "ref" é atualizada, são criadas as entradas do reflog para acompanhar a razão da "ref" ter sido atualizada (que geralmente é o nome do comando de alto nível que atualizou a "ref"), além dos valores antigos e novos da "ref". Um comando Porcelana com script pode usar a função auxiliar *set\_reflog\_action* no comando git sh setup para definir o seu nome para essa variável quando é invocado como o comando de alto nível pelo usuário final, que será registrado no corpo do reflog.

#### GIT\_REF\_PARANOIA

Caso seja definido como 1, inclua as referências quebradas ou com nomes incorretos ao se iterar sobre as listas das referências. Em um repositório normal, não corrompido, isso não faz nada. No entanto, habilitá-lo pode ajudar o git a detectar e abortar algumas operações na presença de referências quebradas. O Git define essa variável de forma automática durante a execução das operações



destrutivas como [git-prune\[1\]](#). Você não precisa configurá-lo sozinho, a menos que queira ser paranóico para garantir que uma operação toque em cada "ref" (caso esteja clonando um repositório para fazer uma cópia de segurança por exemplo).

#### GIT\_ALLOW\_PROTOCOL

Caso seja definido como uma lista de protocolos separados por dois pontos, comporte-se como se a opção de configuração `protocol.allow` esteja definida como `never`, e cada um dos protocolos listados possua `protocol.<nome>.allow` definido como `always` (substituindo qualquer configuração já existente). Em outras palavras, qualquer protocolo não mencionado será proibido (ou seja, esta é uma lista de permissões e não uma lista negra). Consulte a descrição do `protocol.allow` no [git-config\[1\]](#) para mais detalhes.

#### GIT\_PROTOCOL\_FROM\_USER

Defina como 0 para evitar que os protocolos utilizados por `fetch/push/clone` sejam configurados por `user`. É útil para restringir a inicialização recursiva do submódulo de um repositório não confiável ou para programas que alimentam as URLs potencialmente não confiáveis aos comandos `git`. Para mais detalhes consulte [git-config\[1\]](#).

#### GIT\_PROTOCOL

Apenas para utilização interna. Used in handshaking the wire protocol. Contains a colon : separated list of keys with optional values `key[=value]`. Presence of unknown keys and values must be ignored.

#### GIT\_OPTIONAL\_LOCKS

Caso seja definido como 0, o Git concluirá qualquer operação solicitada sem executar outra operação opcional onde se exija um bloqueio. Como um efeito colateral isso impedirá que o comando `git status` atualize o índice por exemplo. É útil para os processos que estão em execução no segundo plano que não queiram causar contenção do bloqueio com as outras operações no repositório. A predefinição retorna para 1.

#### GIT\_REDIRECT\_STDIN

#### GIT\_REDIRECT\_STDOUT

#### GIT\_REDIRECT\_STDERR

Apenas no Windows: permite redirecionar os identificadores predefinidos de *input/output/error* para os caminhos definidos através das variáveis do ambiente. Em particular isso é útil nos aplicativos "multi-threaded" onde a maneira canônica de encaminhar os identificadores predefinidos através do `CreateProcess()` não seja uma opção pois exigiria que os identificadores fossem marcados como herdáveis (e consequentemente **todo** processo gerado os herdaria, possivelmente fazendo o bloqueio das operações do Git). A intenção primária de utilização é utilizar os pipes informados para comunicação (`\\.\pipe\my-git-stdin-123` por exemplo).

Dois valores especiais são compatíveis: `off` simplesmente fechará o identificador predefinido correspondente e caso `GIT_REDIRECT_STDERR` seja 2 > & 1, a predefinição do erro será redirecionado para o mesmo identificador na saída padrão.

#### GIT\_PRINT\_SHA1\_ELLIPSIS (descontinuado)

Caso seja definido como `yes`, imprima uma elipse seguido de um valor (abreviado) SHA-1. Isso afeta as indicações dos HEADs desanexados ([git-checkout\[1\]](#)) e a saída diff bruta ([git-diff\[1\]](#)). A impressão de uma elipse nos casos mencionados não é mais considerada adequada e é provável que a compatibilidade seja removida em um futuro próximo (junto com a variável).

## Discussão

Mais detalhes estão disponíveis no [capítulo dos conceitos do Git no manual do usuário](#) e [gitcore-tutorial\[7\]](#).



Um projeto Git normalmente consiste em um diretório de trabalho com um subdiretório ".git" no ponto mais alto. O diretório .git contém, entre outras coisas, um banco de dados dos objetos compactados representando o histórico completo do projeto, um arquivo para o "índice" que vincula este histórico ao conteúdo atual da árvore de trabalho e informa os ponteiros para este histórico, como as tags e os cabeçalhos do ramo.

O banco de dados do objeto contém os objetos dos tipos da árvore principal: bolhas, que contêm os dados do arquivo; árvores, que apontam para as bolhas e as outras árvores para criar as hierarquias do diretório; e os commits, cada qual faz referência a uma única árvore e algum número do commit do pai.

O commit, equivalente ao que os outros sistemas chamam do "conjunto de alterações" ou "versão", representa uma etapa no histórico do projeto e cada pai representa uma etapa anterior. Os commits com mais de um pai representam as mesclagens das linhas independentes do desenvolvimento.

Todos os objetos são nomeados pelo hash SHA-1 do seu conteúdo, normalmente gravados como uma sequência com 40 dígitos hexadecimais. Tais nomes são globalmente únicos. Todo o histórico que antecede a um commit pode ser comprovado assinando apenas este commit. Um quarto tipo de objeto, a tag, é fornecida para esta finalidade.

Quando criados pela primeira vez, os objetos são armazenados em arquivos individuais, porém, visando uma maior eficiência, podem ser compactados posteriormente em "pacotes de arquivos".

Os Ponteiros informados chamados refs marcam os pontos interessantes na história. Uma "ref" pode conter o nome SHA-1 de um objeto ou o nome de outra referência. As referências com nomes que comecem com ref/head/ contêm o nome SHA-1 do commit (ou "head") mais recente de um ramo em desenvolvimento. Os nomes SHA-1 das tags de interesse são armazenados em ref/tags/. Uma referência especial chamada HEAD contém o nome da ramificação com a averiguação do momento.

O arquivo do índice é inicializado com uma lista de todos os caminhos e para cada caminho, um objeto bolha e um conjunto de atributos. O objeto bolha representa o conteúdo do arquivo no cabeçalho do ramo atual. Os atributos (hora da última alteração, tamanho, etc.) são obtidos do arquivo correspondente na árvore de trabalho. As alterações subsequentes na árvore de trabalho podem ser encontradas comparando estes atributos. O índice pode ser atualizado com um novo conteúdo e os novos commits podem ser criadas a partir do conteúdo armazenado no índice.

O índice também é capaz de armazenar as várias entradas (chamadas de "estágios") para um determinado nome do caminho. Esses estágios são utilizados para manter as várias versões não mescladas de um arquivo quando uma mesclagem está em andamento.

## DOCUMENTAÇÃO ADICIONAL

Consulte as referências na seção "descrição" para começar a utilizar o Git. O seguinte tem provavelmente bem mais detalhes do que o necessário para um usuário iniciante.

O [capítulo de conceitos do Git do manual do usuário](#) e o [gitcore-tutorial\[7\]](#) fornecem introduções à arquitetura subjacente do Git.

Para obter uma visão geral das recomendações do fluxo de trabalho, consulte [gitworkflows\[7\]](#).

Para mais alguns exemplos úteis, consulte também o documento [howto](#).

As entranhas estão documentadas no [Documentação da API do Git](#).

Os usuários que estiverem migrando do CVS também podem querer ler [gitcvs-migration\[7\]](#).

## Autores

O Git foi iniciado por Linus Torvalds e atualmente é mantido por Junio C Hamano. Várias contribuições vieram da lista de discussão do Git <[git@vger.kernel.org](mailto:git@vger.kernel.org)>.

O <http://www.openhub.net/p/git/contributors/summary> fornece uma lista mais completa de todos os colaboradores.

Caso tenha um clone do *git.git*, a saída do [git-shortlog\[1\]](#) e do [git-blame\[1\]](#) pode exibir os autores para as partes específicas do projeto.

## Reportando um Erro

Relate os erros na lista de discussão do Git <[git@vger.kernel.org](mailto:git@vger.kernel.org)> onde o desenvolvimento e as principais manutenções são realizadas. Você não precisa se inscrever na lista para enviar uma mensagem para lá. Para os relatórios dos erros anteriores e outras discussões, consulte o arquivo da lista de discussão em <https://lore.kernel.org/git>.

Os problemas relevantes para a segurança devem ser divulgadas em particular na mailing list do Git Security <[git-security@googlegroups.com](mailto:git-security@googlegroups.com)>.

## VEJA TAMBÉM

[gittutorial\[7\]](#), [gittutorial-2\[7\]](#), [giteveryday\[7\]](#), [gitcvs-migration\[7\]](#), [gitglossary\[7\]](#), [gitcore-tutorial\[7\]](#), [gitcli\[7\]](#), [O Manual do Usuário do Git](#), [gitworkflows\[7\]](#)

## GIT

Parte do conjunto [git\[1\]](#)