Tutorial Git

Marinêz Virginia de Macêdo Werneck Magalhães
matrícula: 20193015793

CEFET/MG – Campus V

O que é o Git?

Git é um sistema de controle de versão distribuído, que permite uma equipe trabalhar juntos na construção de um projeto. Toda vez que você fizer um commit, ou salvar o estado de seu projeto no Git, ele basicamente tira uma foto de todos os seus arquivos e armazena uma referência para esse conjunto de arquivos. Para ser eficiente, se os arquivos não foram alterados, Git não armazena o arquivo novamente, apenas um link para o arquivo idêntico anterior já armazenado. Git trata seus dados mais como um fluxo do estado dos arquivos.

A maioria das operações no Git só precisa de arquivos e recursos locais para operar - geralmente nenhuma informação é necessária de outro computador da rede. Como você tem toda a história do projeto ali mesmo em seu disco local, a maioria das operações parecem quase instantâneas.Tudo no Git passa por uma soma de verificações (checksum) antes de ser armazenado e é referenciado por esse checksum. Isto significa que é impossivel mudar o conteúdo de qualquer arquivo ou pasta sem que Git saiba

O Git tem três estados principais que seus arquivos podem estar: committed, modificado (modified) e preparado (staged). Committed significa que os dados estão armazenados de forma segura em seu banco de dados local. Modificado significa que você alterou o arquivo, mas ainda não fez o commit no seu banco de dados. Preparado significa que você marcou a versão atual de um arquivo modificado para fazer parte de seu próximo commit.

O diretório Git é onde o Git armazena os metadados e o banco de dados de objetos de seu projeto. Esta é a parte mais importante do Git, e é o que é copiado quando você clona um repositório de outro computador. O diretório de trabalho é uma simples cópia de uma versão do projeto. Esses arquivos são pegos do banco de dados compactado no diretório Git e colocados no disco para você usar ou modificar.

A área de preparo é um arquivo, geralmente contido em seu diretório Git, que armazena informações sobre o que vai entrar em seu próximo commit. É por vezes referido como o "índice", mas também é comum referir-se a ele como área de preparo (staging area).

Como instalar o Git(Linux).

Disponivel em: https://git-scm.com/download/linux

É mais fácil instalar o Git no Linux usando o gerenciador de pacotes preferido da sua distribuição Linux. Abaixo alguns exemplos:

Debian/Ubuntu

```
# apt-get install git

# add-apt-repository ppa:git-core/ppa

# apt update; apt install git
```

Fedora

```
# yum install git (up to Fedora 21)
# dnf install git (Fedora 22 and later)
```

Se você preferir construir a partir do código-fonte, você pode encontrar informações em https://www.kernel.org/pub/software/scm/git/

HELP:

- git help
- git help add
- git help commit

git help <qualquer_comando_git>

As configurações do GIT são armazenadas no arquivo **.gitconfig** localizado dentro do diretório do usuário do Sistema Operacional.

- **Setar usuário**
  git config --global user.name "Mili Werneck"
- **Setar email**
  git config --global user.email mili.werneck@gmail.com
- **Setar editor**
  git config --global core.editor code.exe
- **Setar ferramenta de merge**
  git config --global merge.tool
- **Setar arquivos a serem ignorados**
  git config --global core.excludesfile ~/.gitignore
- **Listar configurações**
  git config -list

Inicializar um novo repositório.

- **Criar novo repositório**
  git init
- **Verificar estado dos arquivos/diretórios**
  git status

Adicionar arquivo/diretório (staged area)

- **Adicionar um arquivo específico**
  git add meu_arquivo.txt
- **Adicionar um diretório específico**
  git add meu_diretorio
- **Clonar um repositório já existente.**
  git clone [url do repositório].
- **Salvando suas alterações para o repositório. Enviando informações para o repositório. Atualizando as informações sobre o repositório.**
  O comando git add diz ao Git que você quer incluir atualizações a um arquivo particular na próxima confirmação. No entanto, git add não afeta realmente o repositório de nenhuma forma significativa—as alterações não são realmente gravadas até você executar git commit. Além de git add e git commit, um terceiro comando git push é utilizado para enviar as alterações confirmadas para repositórios remotos para colaboração.

- **Como funciona o "*branch*".**
  Uma ramificação no *git* é um ponteiro para as alterações feitas nos arquivos do projeto. É útil em situações nas quais você deseja adicionar um novo recurso ou corrigir um erro, gerando uma nova ramificação garantindo que o código instável não seja mesclado nos arquivos do projeto principal. Depois de concluir a atualização dos códigos da ramificação, você pode mesclar a ramificação com a principal, geralmente chamada de *master*.

  Criando um novo branch
  ```
  git branch bug-123
  ```

  Trocando para um branch existente
  ```
  git checkout bug-123
  ```

  Neste caso, o ponteiro principal **HEAD** esta apontando para o branch chamado bug-123.

  Criar um novo branch e trocar
  ```
  git checkout -b bug-456
  ```

  Voltar para o branch principal (master)
  ```
  git checkout master
  ```

- **Fazendo "merge" entre vários "branches".**

  Resolver merge entre os branches
  ```
  git merge bug-123
  ```

  Para realizar o *merge*, é necessário estar no branch que deverá receber as alterações. O *merge* pode automático ou manual.

  Apagando um branch
  ```
  git branch -d bug-123
  ```

  Listar branches
  ```
  git branch
  ```

  Listar branches com informações dos últimos commits
  ```
  git branch -v
  ```

  Listar branches que já foram fundidos (merged) com o master
  ```
  git branch --merged
  ```

  Listar branches que não foram fundidos (merged) com o master
  ```
  git branch -no-merged
  ```

- **Resolvendo conflitos.**

  Bisect

  O bisect (pesquisa binária) é útil para encontrar um commit que esta gerando um bug ou uma inconsistência entre uma sequência de commits.

  Iniciar pesquisa binária

  git bisect start

  Marcar o commit atual como ruim

  git bisect bad

  Marcar o commit como bom

  git bisect good

  Finalizar a pesquisa binária

  git bisect reset

- **Remover arquivo**

  git rm meu_arquivo.txt

- **Remover diretório**

  git rm -r diretorio

- **Exibir histórico**

  git log

- **Exibir histórico de um arquivo especifico**

  git log -- <caminho_do_arquivo>

- **Exibir histórico de um arquivo especifico que contêm uma determinada palavra**

  git log --summary -S<palavra> [<caminho_do_arquivo>]

- **Exibir histórico modificação de um arquivo**

  git log --diff-filter=M -- <caminho_do_arquivo>

  Pode ser substituído por: Adicionado (A), Copiado (C), Apagado (D), Modificado (M), Renomeado (R), entre outros.

  Exibir histórico de um determinado autor

  git log --author=usuario

- **Desfazendo alteração local (working directory)**

  Este comando deve ser utilizando enquanto o arquivo não foi adicionado na **staged area**.

  git checkout -- meu_arquivo.txt

- **Desfazendo alteração local (staging area)**

  Este comando deve ser utilizando quando o arquivo já foi adicionado na **staged area**.

  ```
  git reset HEAD meu_arquivo.txt
  ```

- **Exibir os repositórios remotos**

  ```
  git remote
  git remote -v
  ```

- **Vincular repositório local com um repositório remoto**

  ```
  git remote add origin git@github.com:MiliWerneck/TutorialGit
  ```

- **Exibir informações dos repositórios remotos**

  ```
  git remote show origin
  ```

- **Renomear um repositório remoto**

  ```
  git remote rename origin TutorialGit
  ```

- **Desvincular um repositório remoto**

  ```
  git remote rm TutorialGit
  ```

- **Enviar arquivos/diretórios para o repositório remoto**

  O primeiro **push** de um repositório deve conter o nome do repositório remoto e o branch.

  ```
  git push -u origin master
  ```

  Os demais **pushes** não precisam dessa informação

  ```
  git push
  ```

- **Atualizar repositório local de acordo com o repositório remoto**

- **Atualizar os arquivos no branch atual**

  ```
  git pull
  ```

- **Buscar as alterações, mas não aplicá-las no branch atual**

  ```
  git fetch
  ```

- **Clonar um repositório remoto já existente**

  ```
  git clone git@github.com:MiliWerneck/TutorialGit
  ```

- **Criando uma tag leve**

  ```
  git tag vs-1.1
  ```

- **Criando uma tag anotada**

  git tag -a vs-1.1 -m "Mili Versão 1.1"

- **Criando tags no repositório remoto**

  git push origin vs-1.2

- **Criando todas as tags locais no repositório remoto**

  git push origin -tags

- **Comitar um arquivo**

  git commit meu_arquivo.txt

- **Comitar vários arquivos**

  git commit meu_arquivo.txt meu_outro_arquivo.txt

- **Comitar informando mensagem**

  git commit meuarquivo.txt -m "minha mensagem de commit"

- **Alterando mensagens de commit**

  git commit --amend -m "Minha nova mensagem"

- **Alterar últimos commits**
  - Alterando os três últimos commits

    git rebase -i HEAD~3

  - O editor de texto será aberto com as linhas representando os três últimos commits.

    pick f7f3f6d changed my name a bit
    pick 310154e updated README formatting and added blame
    pick a5f4a0d added catfile

  - Altere para edit os commits que deseja realizar alterações.

    edit f7f3f6d changed my name a bit
    pick 310154e updated README formatting and added blame
    pick a5f4a0d added catfile

  - Feche o editor de texto.
  - Digite o comando para alterar a mensagem do commit que foi marcado como *edit*.
  - git commit -amend -m “Nova mensagem”
  - Aplique a alteração
  - git rebase --continue

- Juntando vários commits

  Seguir os mesmos passos acima, porém marcar os commtis que devem ser juntados com *squash*

- O que é oGitHub?

  O hub de GitHub é o que torna uma linha de comando, como o Git, a maior rede social para desenvolvedores do mundo. Além de contribuir em projetos específicos, o GitHub permite a socialização com pessoas que possuem os mesmos interesses que você. Você pode seguir pessoas e acompanhar o que fazem ou com quem se comunicam.

- Como criar um repositório no GitHub.