**MANUAL E REGRAS DO GIT.**

Neste arquivo será apresentado o manual ensinando os principais comandos git usados no dia a dia de trabalho e formas e regas de uso adotados para seguir dentro da Elgscreen.

SUMÁRIO

[O QUE É GIT E GITLAB? 3](#_Toc122022157)

[CLONANDO REPOSITÓRIOS DO GITLAB. 3](#_Toc122022158)

[SALVANDO SUAS ALTERAÇÕES E OBTENDO MAIS INFORMAÇÕES. 4](#_Toc122022159)

[QUANDO E COMO FAZER UM COMMIT / MERGE? 8](#_Toc122022160)

[ENVIANDO E BUSCANDO ALTERAÇÕES DO REPOSITORIO REMOTO. 9](#_Toc122022161)

[CRIANDO UMA BRANCH E MESCLANDO. 11](#_Toc122022162)

[GITIGNORE. 13](#_Toc122022163)

[CRIANDO UM RESPOSITORIO NO GITLAB. 13](#_Toc122022164)

[README 16](#_Toc122022165)

# O QUE É GIT E GITLAB?

Git é um sistema de controle de versão distribuído, que faz o controle do que cada colaborador está atualizando, mantendo as alterações salvas, impedindo que uma mudança passe por cima da outra permitindo compartilhar e colaborando em equipe de forma eficaz.

O GitLab é uma plataforma para hospedagem de códigos e arquivos, permitindo que o trabalho em equipe em projetos públicos e privados.

# CLONANDO REPOSITÓRIOS DO GITLAB.

Para começar a trabalhar e colaborar com o desenvolvimento de um projeto primeiramente clone um repositório para que você possa trabalhar nele localmente, use git clone <nome da branch> <endereço SSH>, caso o argumento branch não seja especificado o git irá clonar todas as branchs.

Pegue o endereço na página do projeto do GitLab. Observe um exemplo nas imagens abaixo:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Importante! Use sempre SSH para realizar a clonagem dos repositórios, isso garante mais segurança.

Com o comando git branch -a você consegue ver todas as branchs que o repositório possui e em qual delas você está, para se movimentar entre elas use git checkout <nome da branch>. Como ilustra a figura a seguir:

**Texto

Descrição gerada automaticamente**

# SALVANDO SUAS ALTERAÇÕES E OBTENDO MAIS INFORMAÇÕES.

**git add <arquivo>** adiciona o arquivo individualmente ou **git add .** para adicionar todos os arquivos. Com isso você adiciona os arquivos que alterou ou os novos que criou ao futuro commit que será feito.

Caso queira desfazer use **git restore <arquivo>**

Após adiciona os arquivos modificados no commit o commit em si é necessário, para isso use **git commit**,veja mais na imagem abaixo:

**Texto

Descrição gerada automaticamente**

Na primeira linha coloque o título, em seguida deixei uma em branco e na terceira linha coloque a descrição, descreva bem e de forma objetiva e clara o que está alterado/adicionando. No commit são salvas as mudanças dos arquivos que você adicionou ao seu repositório local.

Importante! Se preciso reverte um commit é possível, porém faça isso somente se realmente for necessário, por exemplo, caso tenha feito um commit com erro no projeto e antes de reverte garanta que a sua árvore de trabalho está limpa, faça o git add e o git commit. Veja se está tudo Ok como mostra a imagem a seguir:

Texto

Descrição gerada automaticamente

Para reverte um commit use git revert <hash do commit> (hash é o identificador de cada commit, um comando para verificar o hash será mostrado mais a frente), git revert HEAD~1, sendo 1 a quantidade de commits que deseja reverter ou para o último commit use git revert HEAD. O que o git está fazendo nesse comando é um novo commit com os dados do commit anterior ao que você deseja reverte. Observe na figura abaixo:

Texto

Descrição gerada automaticamente

Quando feito o comando para reverte o git irá abrir o editor vim para você editar sua mensagem do commit de reversão, em amarelo coloque a mensagem que deseja, por padrão esta é a mensagem que vem. Para salvar e prosseguir pressione “**Esc”, dois pontos (:)** e em seguida **digite** “**wq”**, **use estes passos sempre que estiver dentro do editor de texto para sair.**

Após, o commit revert será feito e o seguinte resultado irá aparecer em seu terminal, como ilustra a imagem abaixo:

Texto

Descrição gerada automaticamente

Com isso o último commit feito será revertido.

Para verificar o histórico dos seus commits use git log, com isso poderá ver mais informações como data do commit, autor, hash, título e descrição. Observe na imagem a seguir:

Texto

Descrição gerada automaticamente

É possível personalizar esse comando para exibir as informações da forma que deseja com git log –pretty, um exemplo é git log –pretty ="format:%an | %as | %s" que exibi somente o autor, data e título do commit, como mostra a imagem:

Texto

Descrição gerada automaticamente

Veja mais em <https://devhints.io/git-log-format>

Por último para obter mais informações sobre o status do repositório, utilize git status. Veja um exemplo dos resultados possíveis nas imagens a seguir.

Quando o arquivo e modificado ou criado. Observe que o git mostra o que precisa ser feita, indicando que use o comando git add:

Texto

Descrição gerada automaticamente

Após o git add, informar que existem novas mudanças a serem “comitadas”:

Texto

Descrição gerada automaticamente

Logo depois de ser feito o commit, neste caso não existem mais mudanças a serem salvas:

Texto

Descrição gerada automaticamente

Em resumo:

* Adicionar arquivos para serem salvos: git add
* Salvar as mudanças em si: git commit
* Verificar o status do projeto: git status
* Visualizar o histórico de commits: git log

# QUANDO E COMO FAZER UM COMMIT / MERGE?

Em geral nunca commit um código que não funciona, quando fazer fica a critério de cada um, recomendações são “commitar” a cada alteração e correção significativas.

Atenção aos commits! Eles são uma forma de documentação do projeto registrando quais alterações foram feitas, quem as fez e mostra como o projeto foi evoluindo ao longo do tempo, por isso é importante manter um bom registro fazendo bons commits seguindo um padrão e boas práticas.

Um commit deve ser objetivo e sempre com um título e corpo. No título coloque uma frase que descreva o que foi feito, não há um limite para o título, mas a boa prática é usar até 50 caracteres e para a mensagem do corpo coloque em mais detalhes o que foi feito e onde foi feito, buscando ser objetivo, caso queira use listas também. O corpo é importante para diferenciamos commits que tenha os títulos iguais ou parecidos e para manter um histórico melhor, com informações importantes sobre as alterações feitas.

Um commit deve ser feito a cada alteração significativa no projeto, sempre separe em commit diferentes suas alterações, por exemplo, em um projeto você alterou as configurações e os módulos usados, faça um commit para as configurações e um para os módulos assim caso ocorra um erro será mais fácil voltar a versão antiga onde é necessário corrigir somente as configurações.

Não faça um git add . para adicionar tudo ao commit se os arquivos não forem sobre a mesma modificação, separe em cada commit o que foi feito.

Se você fizer um commit com algo errado no título ou no corpo corrija, como dito antes tudo isso serve como documentação e é importante manter o bom registro, para isso utilize o comando git commit –amend, com isso o editor vim será aberto para você editar o título e a descrição do último commit.

Usar commit para registrar quem está mexendo em qual parte do código?

# ENVIANDO E BUSCANDO ALTERAÇÕES DO REPOSITORIO REMOTO.

Agora que você já editou, alterou, modificou e criou vamos enviar tudo para o repositório remoto. Observe a imagem:

Texto

Descrição gerada automaticamente

Em um repositório com conexão a outro repositório remoto após fazer o commit o git status mostra que ainda precisamos fazer o git push, com isso iremos enviar as mudanças para o repositório remoto, use o comando git push <nome do repositório> <branch>, para verificar o nome do repositório use git remote, por padrão o git nomeia um repositório clonado como origin. Esses parâmetros não são obrigatórios, caso não seja informado será considerado o repositório origin e a branch em que está localizado. Veja um exemplo do resultado de um git push:

Texto

Descrição gerada automaticamente

Pronto, suas alterações foram salvas no repositório remoto. Veja na imagem as mudanças salvas:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

No GitLab vemos que tudo já foi salvo mostrando o último commit feito e o arquivo criado nele já está no projeto.

Por fim temos o comando git pull <repositório> <branch>, ele vai atualizar o seu repositório local clonado, baixando novos arquivos e deixando igual ao repositório remoto. Assim você ficará atualizado sobre as ultimas atualizações feitas no projeto pelos seus colegas de equipe. Observe na imagem abaixo:



Neste exemplo o git informa que o repositório já está atualizado, não tendo novas mudanças.

# CRIANDO UMA BRANCH E MESCLANDO.

Para criar uma nova branch a parti da branch em que você está use git branch <nome da branch>, para ver a lista de branchs que existem no projeto utilize git branch e para trocar de branch git switch <nome da branch>. Olhe como seria o resultado dos comandos na ilustração a seguir:

Texto

Descrição gerada automaticamente

Dessa forma você criou uma branch cópia da branch main.

Agora para que essa nova branch esteja no repositório remoto use git push --set-upstream <repositório> <branch>, veja um exemplo abaixo:

Texto

Descrição gerada automaticamente

Acessando o site do GitLab no projeto vemos que a nova branch já está lá, observe na imagem:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Usando o comando o git merge <branch> você poderá mesclar duas branch, com isso o git irá juntar as modificações de ambas as branchs e unir em uma. Com isso a sua branch local main foi atualizada com as modificações da branch novo-ramo:

Texto

Descrição gerada automaticamente

Em seguida faça um git push para que a branch seja atualizada no repositório remoto também:

Texto

Descrição gerada automaticamente

# GITIGNORE.

“.gitignore” (ponto gitignore) é um arquivo git de texto oculto, no git bash arquivos que começam com ponto são ocultos, criado para adicionar nele uma lista com os arquivos e pastas que deseja que o git ignore o monitoramento e impeça que seja colocado em seu projeto, como por exemplo arquivos de IDE que variam de acordo com cada um e cada programador pode usar um diferente do outro.

* **Ignorar arquivo:** nome do arquivo
* **Ignorar pasta:** nome da pasta/

Use “#” para colocar comentários, sempre coloque um comentário explicando o motivo e a origem do arquivo que deseja ignorar. Veja na ilustração a seguir um exemplo:

Polígono

Descrição gerada automaticamente com confiança média

Atenção! Para adicionar o. gitignore ao commit é preciso usar o comando individual, git add .gitignore, usar git add . para adicionar tudo não irá localizar o. gitignore, já que ele é um arquivo oculto. Coloque-o no diretório raiz do projeto.

# CRIANDO UM RESPOSITORIO NO GITLAB.

Na página inicial, procure pelo botão “New project” e aperte, como ilustra a imagem:

****

Em seguida na nova página que será aberta selecione uma das opções que desejar, irei prosseguir clicando em “Criar projeto em branco”. Observe na ilustração como seria:

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

Depois escolha um nome para o projeto e selecione o nível de visibilidade como **privado**, para os projetos da empresa utilize somente essa opção para que terceiro de fora não se conectem, clique em “criar projeto”. Veja na imagem como ficaria:

**Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente**

E pronto, temos o projeto. Veja na imagem o exemplo:

**Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente**

README**.**

README.md é um arquivo é um arquivo de extensão markdown, nesse arquivo adicionados as informações sobre o projeto, introduzindo um novo colaborador a ele. O README faz parte da documentação do projeto, por isso é importante preenchê-lo ele será como um cartão de visitas do projeto, podendo ser usados imagem e vídeos. Siga o modelo ilustrado abaixo:

Interface gráfica do usuário, Texto, Aplicativo, Email, Teams

Descrição gerada automaticamente

Mais informações podem ser adicionadas caso ache necessário, em “Funções em uso” coloque os arquivos no qual você está trabalhando e veja em quais seu time está atuando para evitar conflitos de versionamento e trabalhar melhor em equipe.

**FLUXO DE NOVAS ATUALIZAÇÕES:**

Git add

Git commit

Git push repositório staging

No servidor com a base de staging testar as novas mudanças

Correto?

Sim, merge em produção

Não, trabalhar na correção

Sim, fim

Caso correto siga com os testes também em produção, correto?

Nota: será mesmo preciso testar de novo em produção?