Git

Bom dia, boa tarde e boa noite a quem estiver lendo essa sinjela documentação. Este repositório está sendo criado e posteriormente melhorado com o intuito principal de estudos e de referencias futuras para minha pessoa e claro e de quem mais se interessar.

Segue a estrutura do documento:

Configurando o git

Depois de baixado, precisamos instalar e configurar o git. Abaixo foram listados comandos que podem ser usados para configurar a primeiro momento o Git.

O git trabalha as configurações em três diferentes escopos de atuação:

1. **--system**: válido para todos os usuários no sistema e todos os seus repositórios;
2. **--global**: válido somente para seu usuário e os repositórios desse usuário;
3. **config**: válido somente para o repositório onde você se encontra no momento;

A seguir eu tentei listar de forma simples três comandos que podemos usar para configurar username, email e o editor principal que podemos usar no git, lembrando que este ultimo pode ser configurado no momento da instalação.

É uma boa ideia se apresentar ao git com seu nome de usuário e seu email antes de fazer qualquer alteração. A maneira mais fácil de fazer isso é com os próximos dois comandos.

Configurando usuário/username git

Troque "<username>" pelo nome de seu nome de usuário. Lembrando que, ao utilizar o parametro --global estamos configurando este usuário para todos respositórios do usuário do computador.

```
git config --global user.name "<username>"
```

Configurando email

Troque a string "<exemple@email.com>" pelo seu email.

```
git config --global user.email "<exemple@email.com>"
```

Configurar editor principal do git

Esse passo é importante caso você queira usar um editor de texto diferente do padrão do git. Para fazer isso basta trocar <editor> pelo nome do seu editor favorito.

```
git config --global core.editor <editor>
```

Exibindo nossas configurações do Git

Exibe usuário configurado no Git

No comando abaixo exibimos o nome do nosso usuário

```
git config user.name
```

Exibe email configurado no Git

Já no comando abaixo, exibimos o email do usuário que foi configurado.

```
git config user.email
```

Exibe todas as configurações do git

Com o comando abaixo podemos exibir todas as configurações do nosso usuário Git.

```
git config --list
```

Inicializando um repositorio

Passando da etapa inicial de instalar e configurar o Git iremos bricar agora com repositórios. Como podemos criar um repositório na nossa máquina? Ao criá-lo já podemos fazer um commit? Temos que configurar algo a mais? Essas são uma das duvidas iniciais que eu tive quando comecei a estudar git e talvez sejam duvidas comuns entre jovens desenvolvedores que estão dando seus primeiros passos. Enfim, abaixo listei alguns comandos que são como "rituais" que devemos seguir para criar e configurar um repositório usando o Git. Vamos lá!

Criar uma pasta

Primeiro iremos criar um diretório que servirá como base para nosso repositório. Lembrando que o comando abaixo não é necessariamente um comando git, mas sim um comando tipico do um terminal linux.

```
mkdir <nome da pasta>
```

Inicializando repositorio

Digamos que no passo acima criamos um diretorio chamado rep-test com o comando `mkdir rep-test`, ainda com a janela do terminal aberta termos que entrar nesse diretório usando o comando `cd rep-test`. Logo após fazermos isso estaremos dentro de nada mais nada menos do que uma pasta vazia. Para transformar esta página em um repositório usamos o comando abaixo.

```
git init
```

Logo após executar esse comando, voce irá receber uma mensagem parecida com essa:

```
Initialized empty Git repository in C:/Users/Leonardo/Desktop/rep-test/.git/
```

O que acabamos de fazer foi inicializar um repositório vazio na nossa máquina e deixá-lo pronto para iniciar nossos trabalhos de controle de versão da nossa aplicação.

Manipulando um repositorio

As coisas estão fáceis até agora e adivinha, irão continuar fáceis. O que vamos fazer nesses próximos passos é lidar com um fluxo simples de controle de versão do git: verificando o que foi alterado, adicionandos nossos arquivos modificados na zona de staged e, finalmente fazendo nossos tão famigerados commits.

Reportar status do repositorio

Olha, logo após adicionarmos ou modificarmos um ou vários arquivos do nosso projeto podemos listar todos os arquivos que, de fato, sofreram alguma modificação usando o seguinte comando:

```
git status
```

Então, digamos que queremos adicionar um arquivo chamado teste.txt no nosso repositório e fazê-lo ter uma mensagem "testando o comando git status" basta digitarmos no terminal:

```
echo "Testando o comando git status" >> teste.txt
```

Logo depois de executarmos o comando acima veremos que ele já aparece na nossa arvore do respositório como um arquivo adicionado, para ver isso basta executar o comando `git status`. Uma mensagem parecida com a seguinte será exibida:

```
On branch master

No commits yet

Untracked files:
(use "git add <file>..." to include in what will be committed)
        teste.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Viram? Nosso arquivo apareceu logo abaixo da mensagem `(use "git add <file>..." to include in what will be committed)` e essa mensagem merece ser lida e significada, porque agora, se seguirmos o que essa mensagem nos diz veremos que ela nos pede para prepararmos nosso arquivo para ser commitado. Então vamos lá!

Adicionar um arquivo ou arquivos em um grupo de versionamento

Lembra do nosso arquivo teste.txt? O que devemos fazer com ele para que possamos monitorá-lo com nosso versionador de código? É isso ai, temos que executar o comando abaixo. Bora lá, executar o comando pra ver o que acontece.

```
git add <nome do arquivo>
```

Ué, mas não aconteceu nada. Será que deu alguma coisa errada? Que tal executar o comando `git status` novamente? Provavelmente apareceu no seu terminal algo parecido com:

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   teste.txt
```

Parece que algo de diferente foi exibido. Vamos ver o que foi? Então, logo abaixo da mensagem `Changes to be commited:` vemos o nosso arquivo novamente. E ai, o que aconteceu? O que aconteceu foi que seu arquivo teste.txt digievoluiu de untracked file para ready to commit file (arquivo pronto para ser commitado). E ai, o que eu faço agora? Simples, commita ele ... o que pode dar de errado? (rs)

Commitar um arquivo ou diretorio

Olha só, chegamos no tão famigerado commit, e ai, o que ele faz? Bem, quando commitamos um arquivo ou um conjunto de arquivos estamos criando um screenshot daquele conjunto de arquivos, mostrando pro git o que está diferente entre ele e uma versão mais antiga dele. Lembrando que todo commit é guardado em um histórico, que pode ser lido e, se algum commit foi feito de maneira errada ou gerou algum bug, tudo pode ser voltado atrás, quando tudo estava funcionando lindamente bem (ou quase ...)

O comando para commitar é o seguinte:

```
git commit -m <filename>
```

Vamos praticar usando o nosso exemplo de antes, o tal do teste.txt? Pois bem, execute ai no seu terminal o comando `git commit -m "Esse é meu primeiro commit no arquivo teste.txt"`. Olha só, e ai, o que acontece agora?

Provavelmente, ao teclar ENTER, será mostrada uma mensagem parecida com a seguinte no seu terminal:

```
[master (root-commit) fa1a61d] Este é meu primeiro commit no arquivo teste.txt
1 file changed, 1 insertion(+)
create mode 100644 teste.txt
```

Pronto, commit feito! Agora já pode partir e implementar novas funcionalidades, criar novos módulos ou novas interfaces. Se seguir esse ritualzinho básico de `git status`, `git add` e `git commit` voce já faz muita coisa.

Conselho, dê uma olhada na documentação do comando git commit e estuda o que é esse parametro `-m` e vê se ele é o unico parametro que pode ser usado nesse comando.

Mostra todos os commits já feitos

Imaginemos aquele cenário lindo, em que você implementou várias funcionalidades em vários arquivos diferentes e, do nada, vem na sua cabeça que você não lembra quando, onde ou se, fez determinada alteração no código que pode te gerar alguns problemas no futuro. Qual tal listar todos os commits que você já fez?

O comando `git log` lista todos os commits que voce já fez naquele repositório (ou branch ...). Executa ai no seu terminal o comando `git log` e vê o que aparece.

```
git log
```

Lembra do exemplo em que estavamos trabalhando? O retorno do comando `git log` para aquele repositório é o seguinte.

```
Author: "Seu user.name" <"seu user.email">
Date:   Wed Feb 1 16:52:39 2023 -0300

    Este é meu primeiro commit no arquivo teste.txt
```

Olha só, apareceu alguma coisa. O comando me retornou um commit, feito pelo autor que tem o user.name e que tem o user.email, e foi feito na data mostrada. Legal né?

Mostrar historico de commits detalhado

Na mesma vibe (Ooops ... quero dizer, seguindo a linha de raciocinio) do comando anterior, podemos mostrar no nosso terminal um histórico detalhado dos nossos commits. Bora lá? Dessa vez eu vou deixar você executar o comando no seu terminal.

```
git log --decorate
```

Mas, e ai? Mostrou a mesma coisa do que foi mostrado com o `git log` convencional? Calma, quando você estiver trabalhando em um projeto um pouco mais completo, testa o comando, pra ver o que te retorna de diferente do `git log` tradicional.

Listar todos os commmits por autor

Olha só que legal, se passarmos o parametro `--author="Leonardo"` teremos todos os commits feitos pelo author Leonardo. Legal né? É um comando legal quando voce está gerenciando um projeto ou trabalhando com outras pessoas e quer ver suas proprias alterações.

```
git log --author="<author name>"
```

Lista o historico resumido por autor

Imagine que você quer um resumo do seu histórico onde seja mostrado na tela uma lista de autores, quantos commits cada um fez e quais foram esses commits, com `shotlog` você consegue esse histórico, basta executar o comando abaixo no seu terminal.

```
git shortlog
```

Mostra historico gráfico de commits

O Git também pode te retornar o histórico de maneira gráfica, mostrando os pontos em que seu projeto se ramificou em mais de um branch e onde cada um se encontrou ou se encontrou com o branch principal. Se você tiver somente um branch no seu projeto esse não é um comando muito interessante, porém, se você gosta de ramificar seu projeto para testar novas funcionalidades ou está trabalhando com várias pessoas ao mesmo tempo e quer saber de forma visual onde cada ramo do projeto se criou, esse é um comando muito util.

```
git log --graph
```

Verificar mudanças em um arquivo antes de ser commitado

Algumas vezes você está brincando com aquele projeto, com várias funcionalidades diferentes, em vários arquivos diferentes e acaba deixando tudo pra commitar depois, quando terminar tudo. Enfim, é bem provavel que você acabe sem lembrar o que você mudou em cada arquivo e onde mudou (se você for que nem eu). Pensando nisso (provavelmente) o Git implementa tambem uma função chamada `diff`, essa função te retorna um resumo do arquivo destacando em que pontos houve mudança e em que pontos tudo se manteve inalterado.

```
git diff <filename>
```

Lembrando que filaname é opcional. Caso você não especifique esse parametro, o git irá retornar para você um resumo de todas as alterações que houve em todos os arquivos alterados na arvore do seu projeto.

Mostrar somente o nome do arquivo que foi modificado

Taí um comando que eu acho desnecessário, mas claro, é só minha opinião. Lembrando que essa opinião também é formada com base na pouca experiência que tive com essa ferramenta e com projetos no geral. Então, bora lá? Esse comando aqui te mostra a lista de todos os arquivos que foram motificados, somente os nomes.

```
git diff --name-only
```

Verificar mudanças em um arquivo já commitado

Commitei meu arquivo, mas não lembro se as alterações realmente batem com a minha descrição, e ai, como faço pra conferir as mudanças em um arquivo já commitado? Fácil, usa um git show no teu arquivo, ou, no caso, na hash do teu commit.

```
git show <hash>
```

Lembrando que a hash do commit também é opcional. Mas lembre que é mais fácil de ler as alterações feitas em um só arquivo por vez.

Commit em arquivo que já existiu

É como o meme "Copia mas não faz igual". Eu nem sei bem qual a diferença entre esse comando e o comando `git commit -m "mensagem"`, mas enfim, existe, funciona e faz praticamente a mesma coisa.

```
git commit -am "<mesage>"
```

Dizem que esse comando é serve pra commitar arquivos recem adicionados no seu projeto (reza a lenda ...).

Desfazendo alterações

Voltar ao status de um arquivo para antes da edição

Fiquei empolgado no meio do projeto e fiz algumas alterações em alguns arquivos no meu projeto e tudo quebrou mas eu não lemnbro bem o que eu de fato alterei. Que tal voltar seu arquivo para o que era antes das alterações? O papai git deixa ...

```
git checkout <filename>
```

Remover arquivos da zona de staged

E quando eu adiciono algum arquivo na área de staged (pronto pra commitar) mas não era minha intenção? Eu quero commitar os outros arquivos, mas não aquele, o que eu faço? Roda o comando abaixo que é sal ...

```
git reset HEAD <filename>
```

Lembra de colocar o nome direitinho do arquivo que você quer remover da zona de staged

Remover arquivos ja commitados

Commitei um tal arquivo mas não era bem isso que queria fazer, e agora? Os três comandinhos abaixo podem te ajudar, mas sempre lembre "com grandes poderes vem grandes responsabilidades".

```
ex1: git reset --soft <hash>
ex2: git reset --mixed <hash>
ex3: git reset --hard <hash>
```

- --soft: retorna o arquivo para zona de staged (pronto para commitar)
- --mixed: retorna o arquivo para zona de modified (pronto para addicionar ao stagied e commitar - ainda com as alterações do arquivo)
- --hard: mata o commit e todas as alterações feitas no arquivo

Obs: e esse hash ai, qual eu escolho? Se eu quero desfazer uma alteração commitada, eu vou pegar o hash anterior ao hash do commit que eu quero "matar"

Reverter um commit

Parece que tive um déja-vu. Tá, eu usaria qualquer um dos comandos acima, mas existe, e tai.

```
git revert <hash_do_commit>
```

Tá, eu fiquei curioso e fui procurar a documentação do comando, e olha, é um comando legalzin se tu quer manter teu histórico intácto. Vide a descrição inicial abaixo:

O git revertcomando pode ser considerado um comando do tipo 'desfazer', porém não é uma operação de desfazer tradicional. Em vez de remover o commit do histórico do projeto, ele descobre como inverter as alterações introduzidas pelo commit e anexa um novo commit com o conteúdo inverso resultante. Isso evita que o Git perca o histórico, o que é importante para a integridade do seu histórico de revisões e para uma colaboração confiável. fonte (https://www.atlassian.com/git/tutorials/undoing-changes/git-revert#:~:text=The%20git%20revert%20command%20is%20a%20forward%2Dmoving%20undo%20operation,in%20regards%20to%20los

Repositorio Remoto

Adicionando um repositório remoto no seu projeto

Eu conheço dois jeitos de usar um repositório remoto no seu projeto, o primeiro é criando um repositório local e depois linkando com seu repositório remoto e o segundo é criando um repositório remoto e fazendo um clone na sua máquina.

No primeiro caso você deve que ter um repositório local, devidamente inicializado, e um repositório remoto criado na sua plataforma de hospedagem preferida (GitHub, BitBucket e por ai vai). Não sabe inicializar um repositório?

Fácil, primeiro crie um diretório na sua máquina em qualquer lugar, abra o terminal, o bash do git ou faça tudo pela sua IDE. Eu acho mais fácil usando o bash do git (porquê é colorido rs) e é assim que vou ensinar aqui, enfim ... Seguindo em frente dentro do seu terminal acesse o diretório que você acabou de criar e que será seu repositório local. Estando dentro do diretório, execute o comando `git init` e vualá, seu repositório local está criado. Em seguida podemos linkar nosso repositório remoto com seu repositório local.

No github, dentro seu repositório remoto, você pode visualizar um botão verde, escrito `<> Code`, clique nele e copie o link https. Com isso feito, basta executar o comando abaixo no seu terminal (ou no meu caso, no git bash)

```
git remote add origin master <cole aqui o link do seu repositório remoto>
```

Vamos só revisar as partes do comando acima:

- *git remote add*: Você está dizendo ao git que quer adicionar um repositório remoto.
- *origin*: Muito utilizado por convenção *origin* é somente o nome do seu repositório remoto e claro, pode ser qualquer outro nome que você desejar, está ai simplesmente para fins de identificação.
- *master*: A identificação do seu branch principal. Por convenção também é usado o master mas acredito que em algum projeto envolvendo muitas pessoas o nome desse branch possa mudar.
- *url do repositório remoto*: Por ultimo, é para onde você quer que o git mande suas alterações.

Uma pergunta que me vinha na cabeça quando eu estava estudando sobre git era se eu poderia ter mais de um repositório remoto para o mesmo repositório local. A resposta é sim, você pode, eu posso e todos nós podemos. Mas cuidado, é fácil se confundir para onde você está mandando as alterações no seu projeto, então fique atento.

O segundo caminho para linkar um repositório local a um repositório remoto (e eu não sei se isso é uma boa prática, provavelmente não) é criando primeiro um repositório remoto e depois usando o comando abaixo

```
git clone <url do repositório remoto>
```

Escolha um diretório em que você deseja que fique seu repositório local, acesse esse diretório com o seu terminal e execute o comando acima. Pronto, depois que o git terminar de fazer o clone do repositório remoto você terá seu repositório local e você nem precisa se preocupar em executar o comando `git remote add`, porque já está tudo implicito no `git clone`.

Remover repositórios remotos

Você deve estar se perguntando: "E se eu configurar errado meu repositório remoto e não conseguir fazer um pull ou push?". Se por acaso você, assim como eu, conseguir causar algum conflito inicial na configuração dos seus repositórios (local -> remote) e já fez alguns commits no seu repositório local e só está querendo subir suas alterações sem problema mas o git não está deixando, você pode resolver seus conflitos criando um novo repositório remoto sem nada dentro (assim não vai ter o que fazer com o pull) e remover a referencia do antigo repositório com comando abaixo:

```
git remote remove <nome do repositório>
```

Lembra do passo Adicionando um repositório remoto no seu projeto? Lembra que você configurou um nome para o seu repositório remoto, que no exemplo foi nomeado como `origin`, pois bem, apague-o e adicione o seu novo repositório remoto, com o mesmo nome se desejar.

Mostra os respositorios remotos que existem

Eu to mandando minhas alterações pra onde? Sei lá, vou dar um `git remote` aqui pra saber

```
git remote
```

Mostra os respositorios remotos que existem detalhado

Eu to mandando minhas alterações pra onde? Sei lá, vou dar um `git remote` aqui pra saber (detalhado rs)

```
git remote -v
```

Enviar para o repositorio remoto

"Tá, terminei de commitar, to cansado e quero ir tomar um cafezin pra terminar o dia". Tudo bem, mas não esqueça de fazer o que todo programador deve fazer (quebrar o sistema que ele tá trabalhando ... brinks) subir as alterações pro repositório remoto (Vai que tua máquina resolve se matar de hoje pra amanhã, pelo menos teu trabalho não se perdeu).

```
git push [origin] [master]
```

- origin: nome do repositorio remoto
- master: branch que estou no momento

Obs.: Tanto origin quanto master são opcionais, um `git pull` também funciona.

Resgatar alterações do meu repositorio remoto

Olha bem, esse comando serve pra resgatar as alterações que outros membros da equipe fizeram antes de você ou se voce quebrou tanto o código que decidiu só apagar tudo e quer resgatar uma versão do projeto que ainda funcione.

```
git pull [origin] [master]
```

Obs.: Tanto origin quanto master são opcionais, um `git pull` também funciona.

Clonar todo um repositorio meu ou de terceiros

"Olha, um projeto legal. Quero trabalhar em cima dele", faz um clone ai. Também serve pro caso de "Onde diabos eu fiz o backup do meu repositorio local?", faz um clone, ele resolve.

```
git clone <url do repositorio>
```

Branchs

Branchs são ponteiros moveis que lavam a um commit (é, são ponteiros 😑)

Criando um branch

"Tá, pensei numa coisa aqui, mas não queria quebrar muito meu projeto". Você pode resolver esse problema, meu jovem! Faz um branch, ou ramo, e codifica o que você quiser por lá. (Mas se fizer besteira demais, não faz um merge, por favor)

```
git checkout -b <nome do branch>
```

Mostrar os branchs que eu tenho no respositorio

Tá, você é uma pessoa hiperativa, quer fazer várias coisas de uma vez, criou vários branchs e não sabe mais quantos ou quais são. O comando abaixo ajuda você.

```
git branch
```

Mudar para outro branch no repositorio

Você cansou de fazer aquelas modificações em paralelo ao seu projeto principal e agora quer voltar para o seu branch principal? Dá também, é só usar o comando abaixo:

```
git checkout <nome do branch>
```

Deletar branch que não é mais necessário

Agora que você fez o que tinha que fazer, mas acha que as alterações que você fez naquele branch não vão encaixar muito bem no seu projeto principal. Nesse caso, voce pode simplesmente apagar aquele branch com o `-D` (Delete).

```
git branch -D <nome do branch>
```

Unir branchs

Existem basicamente duas maneiras de unir dois branchs, o 'merge' e o 'rebase'. Eles fazem basicamente a mesma coisa, mas só que de jeitos totalmente diferentes. É bom saber a diferença.

Merge

Ao fazer um merge se cria um um outro commit no ramo principal, onde esse commit apontará para as duas ramificações dos dois outros branchs.

- Pros: Operação não destrutiva;
- Contra: É necessário um commit extra; Histórico fica poluído

```
git merge <branch-name>
```

Rebase

Recoloca tudo que estava no branch separado e coloca no começo da fila 'matando' os commits que estavam no branch onde foi feito o rebase

- Pros: Evita commits extras; Histórico Linear;
- Contra: Perca de ordem cronológica; Histórico fica poluído

A depender da técnica que você use, e organização que seja feita para fazer merge na branch principal (e.g.: master), esses contras não existem, aliás é o contrário, a história de commits fica mais fácil de entender em relação ao fluxo de desenvolvimento.

Segundo Manoel Vivela Machado (https://www.linkedin.com/in/lerax/) (um baita de um desenvolvedor):

" nunca fazer rebase em branchs públicas e compartilhadas (master/develop etc), mas apenas em feature-branchs, assim não rola conflito com rebase e se você ainda fizer squash por feature-branch fica mais fácil de fazer revert, gerar changelog automático por release etc."

```
git rebase <branch-name>
```

Salvando modificações temporariamente

Salvando alterações em stash

Esse próximo comando é uma mão na roda quando você tem alterações no repositório local mas não quer fazer commit ainda. Dá pra pegar essas alterações, salvá-las num arquivo temporário e depois, só voltar com elas, ou excluí-las caso queira.

Esse comando salva as modificações de um arquivo temporariamente dentro de um stash, para que não seja necessário fazer um commit, caso se queira mudar de um branch para outro.

```
git stash
```

Aplicar as mudanças guardadas anteriormente em stash

Lembra que `git stash` salva suas alterações em um arquivo temporário? O parametro `aplly` aplica as modificações que você salvou anteriormente no stash.

```
git stash apply
```

Listar todos os stashs que se está fazendo

O parametro `list` é usado pra listar todos os arquivos temporários que você tem salvo em stash.

```
git stash list
```

Limpar tudo que está no stash

Já o parametro `clear` serve pra você limpar os arquivos temporários que você tem salvos em stash.

```
git stash clear
```

Configurar alias/abreviação em um comando git

Os comandos seguintes servem para o contexto de produtividade.

Configurando alias do comando status com s

O Git te permite configurar abreviações para comandos, por exemplo, no comando abaixo, configurei o alias/abreviação `s` para o parametro `status`.

```
git config --global alias.s status
```

Depois dessa configuração acima podemos chamar o comando status como

```
git s
```

Criando tags

Passar uma tag com uma anotação

Da mesma forma que a maioria dos VCSs, o Git tem a habilidade de marcar pontos específicos na história como sendo importantes. Normalmente as pessoas usam essa funcionalidade para marcar pontos onde foram feitas releases (v1.0 e assim por diante). Nessa sessão, você irá aprender como listar as tags existentes, como criar novas tags e quais são os diferentes tipos de tags. [fonte (https://git-scm.com/book/pt-br/v2/Fundamentos-de-Git-Criando-Tags)](https://git-scm.com/book/pt-br/v2/Fundamentos-de-Git-Criando-Tags)

```
git tag -a [<numero-da-versão>] -m "<anotação>"
```

Subir uma tag

Semelhante ao commit, posso subir uma tag para o repositório remoto.

```
git push origin master --tags
```

Referências

- [Documentação Git (https://git-scm.com/docs/git/pt_BR)](https://git-scm.com/docs/git/pt_BR)
- [Git para iniciantes (Vídeo) (https://www.youtube.com/watch?v=IBCIN6VpJDw&list=PLIAbYrWSYTiPA2iEiQ2PF_A9j__C4hi0A&ab_channel=WillianJusten)](https://www.youtube.com/watch?v=IBCIN6VpJDw&list=PLIAbYrWSYTiPA2iEiQ2PF_A9j__C4hi0A&ab_channel=WillianJusten)
- [Git tutorial - Atlassian (https://www.atlassian.com/git/tutorials)](https://www.atlassian.com/git/tutorials)