O que é Git? O que é Versionamento?

O Git é um software de controle de versão (VCS). O versionamento utilizado no Git, basicamente, é de código, mas podemos fazer versionamento de outras coisas também.

Um exemplo de versionamento pode ser de um site criado, por exemplo, no qual os arquivos HTML ficariam dentro de uma pasta dedicada ao mesmo, e dentro dela podemos ter outras pastas para CSS, Javascript, arquivos de imagem, mídia, etc.

Para mostrar uma prévia pra um cliente, por exemplo, podemos compactar essa mesma pasta num ZIP, por exemplo, mas à medida que vamos fazendo atualizações nesse site, podemos acumular muitos ZIPs, que pode ser difícil pra procurar uma versão anterior especificada. Podemos fazer o back-up em uma nuvem, como o Google Drive ou o Dropbox, mas isso torna complicado para encontrar determinadas coisas em versões anteriores.

Isso complica ainda mais quando temos muitas pessoas numa equipe trabalhando num mesmo projeto (como, por exemplo, um pra design, um pra front-end e um para back-end). É aí que o Git entra.

O Git é como uma “máquina do tempo”, ele permite voltar a um ponto específico de um arquivo, e ele só armazena as alterações que são feitas num arquivo, e não várias versões do arquivo inteiro (por exemplo, ao alterar um HTML adicionando duas linhas, ele só faz a atualização dessa alteração, e não uma outra cópia do mesmo arquivo atualizado).

Num versionamento centralizado ou linear, vários usuários fazem a alteração de um mesmo projeto que fica num mesmo servidor (que vamos chamar de “commitar”). Já o Git utiliza o sistema distribuído, que cada equipe commita seu projeto no seu computador, no caso, isso é um versionamento local, mas ele pode ser ligado a um repositório remoto único no qual os usuários farão o “push” de seus repositórios locais para ele (“push” seria o mesmo que enviar).

As principais vantagens do versionamento são essas:

- Controle de histórico.
- Trabalho em equipe.
- Ramificação do projeto.
- Segurança.
- Organização.

O que é GitHub? Pra que Ele Serve?

O Git, exemplificado anteriormente, é o repositório local onde “guardamos” as versões do arquivo, nas quais ele só salva as alterações com os “commits”, ao invés de fazer cópias de arquivos com as alterações.

O GitHub é o repositório remoto no qual nós salvaremos nossos projetos versionados no Git. O GitHub é uma plataforma social para programadores, tal como uma rede social, onde, além do versionamento, nós podemos participar de projetos de outros usuários e seguir perfis tais como uma rede social mesmo.

No GitHub, podemos ter:

- Repositórios ilimitados.
- Hospedagem de código-fonte.
- Características de rede social.
- GitHub Pages integrado (para hospedagem de sites com HTML, CSS e JS).
- Colaboração.
- Forks (continuar um projeto de outra pessoa).

Além do GitHub, temos também o GitLab, BitBucket, Phabricator, Gogs, Kallithea, entre outros.

Criando o Primeiro Repositório

Primeiramente, faça o login no GitHub Desktop indo em Files, Options e em Account, clique em Sign into github.com. Ele abrirá o navegador padrão do sistema para que o login possa ser feito.

Só relembrando, lembra do caso onde cada desenvolvedor de um projeto realiza seus commits em seus computadores locais com o Git. Depois todos eles fazem o push para o repositório remoto no GitHub.

No GitHub Desktop, vamos criar um repositório de exemplo (como Ola-Mundo, evite acentuações e espaços). Clique em Create a New Repository on Your Local Drive e ele criará um repositório Git no seu computador. Marque a opção Initialize this repository with a README. Não se preocupe no momento com o Git Ignore e o License (mas nessa podemos escolher uma licença como a GPL ou a Apache).

No caso, ele criará um repositório local onde colocaremos nossos códigos. Nessa mesma pasta criada, tem um diretório oculto com o nome “.git” onde nós temos os dados do repositório.

PS: Pode ser que tenhamos que configurar o usuário para realizar os commits, nesse caso abrimos o Open Git Settings (ou indo em File e Options) e em Git colocamos o nome e o e-mail do usuário, além do branch principal padrão para novos repositórios.

Ao criar o repositório, ele já fará um commit automaticamente, mas podemos desfazer ele clicando em Undo. Podemos ver em History para ver as alterações feitas no repositório. Para fazer um commit no repositório local, coloque o nome dele e clique em Commit logo abaixo.

Para publicar o repositório remotamente no GitHub, vá na parte de cima do GitHub Desktop e clique em Publish Repository. Ele pedirá o nome e a descrição, e se o repositório será privado (se quiser ele público, basta desmarcar essa opção). Um repositório privado pode ter até 3 colaboradores, já um público a quantidade de colaboradores é ilimitada.

PS: Podemos abrir o repositório local no VS Code ou Sublime Text para editar os códigos dos programas. Outras IDEs também costumam ter integração com o Git.

Abrindo um arquivo, como o ReadMe, podemos editar ele, e o GitHub Desktop já perceberá as alterações e sugerirá fazer um commit, então fazemos o commit colocando o nome dele, e ele já estará atualizado localmente. Para publicar no repositório remoto vinculado, clique em Push Origin.

Podemos alterar os arquivos diretamente no site do GitHub, podemos por exemplo editar o ReadMe e clicar em Commit Changes. Coloque o motivo do Commit e escolha a opção Commit Directy to the master Branch. Futuramente podemos criar outras branchs para o mesmo projeto.

Para atualizar nosso repositório local, que teve alterações, clique em Fetch Origin, e depois que ele terminar, em Pull Origin para ele puxar o que está no repositório remoto para o nosso repositório local.

Para atualizar o repositório local com as alterações do repositório remoto no GitHub, clique em Fetch para ele buscar os dados, e se tiver alguma alteração, ele mostrará a opção Pull (puxar) ou Push (enviar).

PS: As opções de Fetch, Pull, Push, entre outras, estão no menu, em Repository. Também não é recomendado fazer os commits no branch principal (no caso, o master).

Clonando um Repositório

Para clonar um repositório (que pode ser de outro usuário), nós abrimos a página do GitHub onde está o repositório.

Podemos baixar diretamente como ZIP ou usando as linhas de comando, mas como estamos utilizando o GitHub Desktop, podemos copiar o link do Git e colar no programa, em Clone Repository, mas podemos clicar diretamente na opção Open with GitHub Desktop que ele abrirá o repositório para ser clonado pro seu computador. Ele copiará absolutamente tudo que está no repositório clonado.

Lembrando que nós não podemos alterar diretamente um repositório de outras pessoas, para isso devemos usar Issues, que aprenderemos mais pra frente.

PS: Para saber se uma pasta estiver versionada, basta ver se ela tem a pasta “.git” oculta na raiz dessa pasta.

Para remover um repositório local, basta ir em Repository e em Remove.

PS: Todas as branches são copiadas nos repositórios clonados, mas apenas a master é a que permanece ativa. Veremos isso mais pra frente.

Versionando seus Projetos Antigos

Para pegar um projeto antigo e versionar ele com Git, abra o GitHub Desktop, clique em File e em New Repository, da mesma forma, coloque o nome do repositório como antes. Caso você tenha uma organização, você pode escolher ela (isso é só pra quem tem empresas cadastradas no GitHub). Daí é só arrastar os códigos para a pasta do repositório e o GitHub Desktop já vai identificar as mudanças.

PS: Caso tenha algum arquivo inútil na pasta, podemos excluir ele no GitHub Desktop, no caso ele vai criar um arquivo oculto com o nome .gitignore onde ele vai ignorar os arquivos especificados. Daí, é só fazer o Commit e o Push pro repositório remoto.

Daí, num outro computador, podemos clonar o repositório como já aprendemos, clicando pra abrir no GitHub Desktop e ele abrirá o programa para clonar. Só lembrando que quando você clona um repositório, ele só é clonado pro repositório local, quando você copia outro projeto pro seu perfil, seria o fork.

Para escrever num comentário do GitHub, podemos usar o @ pra fazer referência ao usuário, e colocar links com a sintaxe *[Texto do Link](URL)* (como por exemplo, *[Abrir o Google] (https://www.google.com.br/)*).

Você sabe Usar Issues?

A Issue é um problema, uma questão, um levantamento de alguma coisa. Vamos supor que a gente descobre um problema num programa de código aberto, como por exemplo, o VS Code. Daí a gente pega os erros do programa e manda uma mensagem no projeto dele no GitHub, para ser corrigido pelos desenvolvedores. Quando a gente é programador experiente, podemos clonar o repositório deles pra nós mesmos resolver. Daí, nesse caso, depois de resolvermos, nós fazemos um fork desse projeto, crio uma branch só pra resolver esse problema, nós corrigimos da nossa forma, e abrimos um pull request (no caso de nós resolvermos o problema, e mostramos pro desenvolvedor do projeto). Caso ele ache necessário, ele pode tornar sua ramificação parte do projeto, e a gente fica como colaborador do projeto.

Vamos, por exemplo, ir no repositório oficial da Microsoft e procurar o repositório principal do VS Code e ver suas issues.

As issues, no caso, são problemas que nós identificamos mas que não sabemos a resposta. Elas podem estar abertas ou fechadas (no caso de finalizada ou descartada, por exemplo). Podemos pesquisar sobre erros específicos, inclusive pelo número. Podemos ter inclusive prints dos bugs dos programas. É recomendado fazer uma busca antes de enviar uma issue para não duplicar as mesmas.

Abra o repositório do curso.

Vamos abrir uma issue de exemplo nesse repositório. Vá em issues e em New Issue, colocamos um título e o conteúdo da mesma, com todos os detalhes e clique em Create. Qualquer pessoa pode submeter issues em qualquer repositório e também podemos comentar nas issues. Podemos também fechar e reabrir nossas próprias issues, além de trancar e destrancar conversas, destacar (em pin issue) e excluir as issues (no final da tela, no canto direito). Podemos formatar, colocar imagens e outras coisas, pra prever as mudanças vá em Preview.

PS: Uma issue pode ser fechada e reaberta pelo dono do repositório. Ele pode também trancar e destrancar uma conversa, destacar e excluir a issue.

Guia da Linguagem Markdown

Vamos testar criando uma issue no repositório acima, para formatar, basicamente fazemos assim, usando essas marcações:

 # Título 1
Título 2
Título 3
Título 4
Título 5
Título 6
Negrito
*_Itálico_*
> Citação

`Código`
[Link](https://nomedosite.com.br/)
![Texto Alternativo](https://linkdaimagem.com/imagem.png)
@CitarUser

Além de usar as marcações, podemso também usar as formatações usando as opções da caixa de texto. Lá temos outras opções, como para criar listas ordenadas e não ordenadas. Podemos também fazer o upload de imagens, que serão linkadas para o comentário.

Para fazer uma tabela, podemos formatar assim:

*Num	Nome	Nota*
*1	Gustavo	8,5*
*2	José	10,0*
*3	Maria	9,0*

Para um comando de mais de uma linha, podemos fazer assim:

```
num = int(input("Digite um valor: "))

if num % 2 == 0:
   printf(f"O valor {num} é par!")
else:
   printf(f"O valor {num} é ímpar!")
```

E para listas:

* Item 1
* Item 2
   * SubItem 1
* Item 3

Podemos também colocar emojis nos comentários, inclusive podemos usar código do Emojipedia, mas eles não podem ser usados nos títulos, nesse caso cole os emojis diretamente.

Git Branches de Forma Fácil

Branch significa ramificação, é como uma “árvore” com vários ramos, e o tronco principal seria o ramo mestre, essa é a ideia de branches no Git, onde temos a branch principal (denominada como main ou master) e temos outras branches (ramos) que partem dessa branch principal e que podem ser fundidas com a branch principal. O ramo principal (branch master) é obrigatório, os outros não são obrigatórios.

Sempre que fazemos uma alteração num projeto no Git, nós fazemos o commit (criar uma versão nesse ponto) e assim ele cria uma nova subversão. Esse conteúdo da branch master pode ser feito o push (enviado) para a origin (origem) (ou seja, tirar do repositório local para enviar pro repositório remoto, no caso, do GitHub).

PS: Evite commitar tudo na branch master, para o caso de não fazer besteiras no programa principal.

Vamos supor que nós façamos um site com o repositório Git, a partir do primeiro commit na master, podemos criar uma branch baseada no mesmo (por exemplo, pra design) e também podemos commitar nessa nova branch criada, que não afetará a branch master. No final, podemos fazer o merge (fusão) da branch nova com a master. Podemos criar quantas branches precisar, e cada uma tem seus commits independentes, e podemos excluir também essas branches.

Vamos criar um repositório local de teste para nós exemplificarmos na prática. Vamos publicar ele no GitHub. Crie um código qualquer nele e commite ele no master, e depois publique no GitHub fazendo o push.

Para criar uma nova branch localmente, vá em Branch e em New Branch e coloque o nome dela (por exemplo, conteúdo) e ele vai dar a opção para deixar as mudanças em master ou trazer minhas mudanças para conteúdo. Faça as alterações na branch e faça o commit nela e depois publique com o push. Seus commits não interferirão na branch principal.

Vamos da mesma forma fazer uma branch pra design, indo no mesmo local, só que dessa vez escolhemos de qual branch ele bifurcará, no caso, será também bifurcação da master.

Na parte de cima, podemos escolher qual a branch atual (current), pra publicar no repositório remoto, basta ir em Publish Branch.

Num projeto profissional, cada setor de um projeto (como design, front-end e back-end) pode ter sua própria branch pra mexer. Todas as branches podem ser publicadas no repositório remoto, assim como acontece localmente.

PS: Ao mudar a branch no GitHub Desktop, ele automaticamente atualizará o código no Sublime Text ou no VS Code. Outras IDEs também atualizarão seus códigos automaticamente.

Vamos mexer na branch de design e fazer outras edições e faça seu commit e sua publicação, elas e seus commits não interferirão na branch principal nem na de conteúdo.

Vamos supor que o cara do design mexeu, por exemplo, no título, e o do conteúdo também mexeu. Isso pode gerar conflito na hora de fundir (merge).

Para fazer um merge, coloque o master como Current Branch, e em Branch, clique em Merge into Current Branch e escolha a branch a ser fundida (no caso, a conteúdo) com a master e clique em Create a Merge Commit. Esse merge é local, então devemos fazer um push pro repositório remoto.

Só que da mesma forma, o design também será fundido com o master, isso dará um conflito, e na hora de fundir o GitHub Desktop avisará isso. Clique normalmente em Create a Merge Commit e ele abrirá pra resolver o conflito, e você pode abrir no editor, que deixará os códigos destacados, e você poderá corrigir no editor. Daí é só voltar no GitHub Desktop e continuar o merge. Depois de tudo, podemos fazer o push pro repositório remoto.

PS: Uma branch pode também ser deletada, para isso vá em Branch e em Delete Branch, caso deseja que ela seja apagada do repositório remoto, marque essa opção, as fusões com a master não serão alteradas.

Hospedagem Grátis no GitHub Pages

Podemos hospedar sites estáticos simples (HTML, CSS e Javascript) no GitHub Pages para que ele possa ser acessado online por outras pessoas.

Para isso, vá no repositório no GitHub, vá em Settings, Pages e em Source, escolha Deploy from a Branch, e em Branch escolha a branch desejada para ser exibida (geralmente a master mesmo), e em Save. Espere um minuto e seu site estará num link como *https://usuariodogithub.github.io/NomeDoRepositorio*.

Fazendo um Fork de um Projeto

Podemos fazer um fork de um projeto já existente de outro usuário no GitHub. Para isso, acesse o repositório desejado e clique no botão Fork, geralmente localizado no canto superior direito da página. Isso criará uma cópia desse repositório no seu próprio perfil do GitHub. Você pode escolher se deseja copiar apenas a branch principal (geralmente chamada de main ou master) ou todas as branches do projeto.

Tudo o que estiver no repositório original será copiado para o seu repositório forkado. Qualquer alteração que você fizer no seu fork não afetará o repositório original.

Se quiser sugerir que suas alterações sejam incorporadas ao repositório original, você pode criar um Pull Request. O dono do repositório original poderá revisar suas mudanças e decidir se deseja integrá-las ao projeto.

Para fazer um Pull Request, acesse o seu repositório forkado, faça as alterações desejadas, realize os commits e envie (push) as alterações para o seu repositório remoto no GitHub. No GitHub, clique no botão Contribute e selecione a opção Open Pull Request. Escreva um título, adicione uma descrição explicando suas mudanças e clique em Create Pull Request.

O que é Git?

Git é um software para rastrear alterações em qualquer conjunto de arquivos, geralmente usado para coordenar o trabalho entre programadores que desenvolvem código-fonte de forma colaborativa durante o desenvolvimento de software. Seus objetivos incluem velocidade, integridade de dados e suporte para fluxos de trabalhos não-lineares distribuídos.

Vamos supor que temos um projeto, como um site, e nosso cliente peça alterações no visual do mesmo, por exemplo. Nós alteramos e modificamos o projeto. Vamos supor que de repente nosso cliente acaba voltando a querer a primeira versão do projeto, se nós fizemos muitas modificações, fica difícil encontrarmos versões mais antigas. Podemos fazer o versionamento criando pastas ou ZIPs com várias versões, inclusive com um arquivo de texto citando as alterações, mas isso acumularia muitos arquivos e consumiria muito espaço em disco. É para resolver todos esses problemas que o Git existe.

Com o Git, podemos, de forma mais inteligente:

- Salvar múltiplas revisões do mesmo projeto em um único diretório. Sem ter v1, v2, v2_final, etc.
- Trocar de versões em um simples comando.
- Trabalhar simultaneamente em 2 ou mais “features” diferentes sem bagunçar o código original do projeto.

- Colaborar com outros programadores no mesmo time.

Os nossos projetos serão hospedados remotamente num servidor, no caso, no GitHub.

GitHub Inc. é um provedor de hospedagem na internet para desenvolvimento de software e controle de versão usando Git. Ele oferece o controle de versão distribuída e a funcionalidade de gerenciamento de código-fonte do Git, além de seus próprios recursos.

No GitHub, nós podemos:

- “Hostear” o seu código em um local seguro.
- Compartilhar seu projeto facilmente com outros devs.
- Outros devs podem colaborar com o seu projeto.
- Entre outras coisas.

Conceitos Básicos do Git

Essencialmente, Git repositório é um diretório chamado .git (oculto) dentro do seu projeto.

Este repositório rastreia todas as mudanças feitas nos arquivos do seu projeto, construindo um histórico ao longo do tempo.

O commit é como se fosse uma marcação histórica no qual seu projeto já está com alguma parte importante definida (como, por exemplo, adicionando forma de pagamento, ou adicionando integração com Whatsapp). É de forma parecida com as fases salvas de um jogo de videogame. Para um commit ser realizado, alguma coisa no projeto deverá ser modificada para isso, os estados, são esses:

- **Modified:** Arquivos alterados.
- **Staging:** Arquivos prontos para serem enviados.
- **Commited:** Arquivos salvos (“commitados”).

PS: Os commits são feitos no repositório local, antes de serem enviados para o repositório remoto (push).

As branches são uma linha do tempo no projeto principal, cuja branch principal geralmente é a master ou main, e desta deriva outras linhas do tempo paralelas, que não interferem uma na outra, e que podem ser fundidas da branch principal.

Clonando Repositórios e Configuração de Usuário

Para clonar um repositório, basta copiar o link do diretório do GitHub e escrever no terminal *git clone https://github.com/nomedousuario/NomeDoRepositorio.git*. Podemos clonar nossos próprios repositórios e também repositórios de outros usuários.

PS: Ao clonar um repositório, ele clonará todas as branches do mesmo, mas apenas a master estará verificada (checkout) e ativa. Não se preocupe com isso agora.

Para entrar com seu e-mail e usuário do GitHub no seu repositório Git na máquina local, digite esses comandos:

git config –global user.email “nomedeusuario@servidor.com”

git config –global user.name “Nome do Usuário”

Para deslogar depois, basta digitar:

git config –global –unset user.name

git config –global –unset user.email

Esses dados serão armazenados num arquivo .gitconfig que ficará no diretório do usuário do seu sistema (como C:\Users\NomeDoUsuario no Windows e /home/NomeDoUsuario no Linux).

Para verificar os usuários inseridos, faça assim:

git config –global user.name

git config –global user.email

Para definir a branch padrão, digite esse comando:

git config –global init.defaultbranch master

E para ver a mesma:

git config –global init.defaultbranch

Podemos ver esses e outros dados na configuração do Git, usando o comando *git config –list*.

Criando Repositórios

Para ver a versão do Git instalada, digite *git –version*.

Para criar um repositório, primeiramente crie duas pastas, pode ser pelo terminal mesmo, com os nomes Projeto-1 e Projeto-2 (evite acentuação e espaços).

Dentro do diretório Projeto-1, digite *git init* para iniciar um novo repositório nela, ele criará uma pasta oculta com o nome “.git” dentro dessa pasta, onde estará as configurações do repositório. Por enquanto não mexeremos no Projeto-2. A branch padrão será a main ou a master (depois criaremos outras).

No repositório Projeto-1, vamos criar um arquivo, por exemplo, index.html, com algum código básico. Esse arquivo e quaisquer outros inseridos nesse repositório, serão monitorados pelo Git, que identificará quaisquer alterações neles, por mínimas que sejam.

Colocando Arquivos no Stage

Para ver o estado atual do repositório local, digite *git status*. Ele mostrará se o repositório tem arquivos sendo monitorados e commits realizados. No caso ele mostrará que tem arquivos que ainda não estão sendo rastreados pelo Git (no caso, nosso index.html) e nenhum commit.

Para adicionar o arquivo pra ser rastreado pelo Git, digite *git add index.html*.

Para remover um arquivo do rastreamento do Git, digite *git rm –cached index.html*. Pode ser necessário especificar *-rf* para excluir diretórios. Para restaurar o arquivo à área de staging após removê-lo do rastreamento digite *git restore --staged index.html*.

Podemos também mover e renomear arquivos dentro do repositório, usando algo como *git mv index.html main.html* ou *git mv *.css css/*.

No Projeto-2, inicie o Git com *git init* e crie mais arquivos, como o index.html e um de estilo.css, que pode estar numa pasta. Nesse caso, para adicionar todos os arquivos do diretório apenas com *git add .* (sem esquecer do ponto), para remover, podemos usar *git rm –cached **.

Fazendo Commits

Remova o Projeto-2 e deixe apenas o Projeto-1, nesse projeto, vamos adicionar os arquivos com *git add .* (não esqueça do ponto, que significa “tudo”), caso não seja adicionado e vamos colocar um ponto no tempo desse projeto, que vamos chamar de “commitar”.

Para fazer o commit, digite *git commit -m “Motivo do Commit”*. Veja o status de noivo com *git status* para ver se o commit foi executado com êxito.

Para ver o histórico de commits, basta digitar no terminal *git log*. Ele mostrará o autor do commit, a data do mesmo e seu hash. Para sair da tela basta digitar “q”. Para ver resumido, digite *git log –oneline*.

Coloque um conteúdo básico no HTML e no CSS, para eles serem alterados. Novamente, digite esses comandos na ordem:

git status

git add index.html

git commit -m “Adicionando Conteúdo HTML”

git status

git log

Desfazendo Commit

Como sabemos, cada commit é um ponto no histórico do seu repositório. Há necessidade de voltar a um commit anterior ao último. Existem três formas de desfazer um commit:

- *checkout*: Um comando seguro, que não altera o histórico do Git. Permite navegart até um commit antigo ou mudar de branch. Pode ser usado para visualizar um estado anterior do projeto, mas não desfaz commits, apenas muda o ponto de trabalho temporariamente.
- *revert*: Também seguro, pois não apaga nem reescreve o histórico. Cria um novo commit que desfaz as alterações de um commit anterior. É recomendado quando é necessário

desfazer mudanças mantendo o histórico intacto, especialmente em projetos já compartilhados.
- *reset*: O comando mais arriscado, pois reescreve o histórico do Git. Pode remover completamente commits posteriores ao commit para o qual se está retornando. Se usado com *–hard*, também apaga alterações nos arquivos e no staging. Deve ser usado com cuidado, especialmente em repositórios compartilhados.

Altere mais uns arquivos e crie mais uns três commits, para exemplificarmos melhor. Depois dê um *git log –oneline* para ver os commits.

Vamos voltar pro terceiro commit, pegar o código dele mostrado no log e digite dessa forma: *git checkout d874777*. Para desfazer essa alteração, digite *git checkout master* que ele fica novamente como antes. Para comprovar, digite *git log –oneline* de novo.

Agora vamos voltar pro segundo commit, pegar o código dele também e digitar assim: *git revert c67f99d*. Ele vai adicionar um comentário nos arquivos de configuração, daí pra confirmar digite *:q*.

Para resetar tudo e que nada fique no histórico, podemos pegar um código do commit e digitar assim: *git reset e0323fe*. Mas no entanto, caso tenhamos alterado o arquivo nos editores, devemos fazer um novo add e um novo commit para essas alterações. Para resetar totalmente, inclusive commits posteriores, digitamos *git reset 0551f68 –hard* (esse comando deve ser usado com cautela).

Ignorando Arquivos

Podemos configurar o Git para que ele não monitore determinados arquivos do repositório, isso é feito num arquivo chamado .gitignore dentro dele. Vamos, por exemplo, criar um robots.txt e colocar ele pra ser ignorado.

Para isso, crie o arquivo .gitignore e adicione essas linhas:

robots.txt
**.gif*
dist

No caso, o primeiro é um arquivo robots.txt, o segundo pra todos os arquivos GIF no diretório, e o dist no caso seria outro diretório dentro do repositório.

Criando Branches

Quando criamos um repositório, automaticamente ele cria uma branch principal, geralmente chamada main ou master. Então podemos fazer um branch, que no caso, é uma ramificação da branch principal, paralela à mesma, onde cada branch tem seus commits independentes. Cada desenvolvedor pode ter sua própria branch no mesmo projeto, e elas posteriormente podem ser fundidas à branch principal.

Digite *git status* para ver o status do repositório e digite *git branch* para ver as branches no projeto, que no caso só teremos a master (principal). Para criar uma nova branch digitamos, por exemplo, *git branch teste* para criar uma branch de nome teste. Ao dar *git branch* de novo, ele mostrará as branches master e teste (a branch na qual estamos atualmente será destacada por um asterisco e de cor verde).

Para mudar pra branch teste, digite *git checkout teste*. Para ver se ele mudou, digite *git branch* de novo. O checkout, além de trocar commits, também troca branches.

Tudo que fizermos na branch teste não afetará em nada a branch master. Vamos por exemplo, criar um arquivo Javascript, adicionar e dar commit nele. Digite *git log –oneline* para vermos os commits realizados no repositório.

Ao digitar *git checkout master* pra voltar pra branch principal, veremos que nada foi alterado (no caso, não terá arquivos Javascript). Ao dar *git log –oneline* de novo ele não terá o commit do Javascript.

Para deletar uma branch, digite *git branch -d teste*, mas caso ela não seja fundida à branch master, ele não deixará excluir. Para forçar exclusão coloque o "D" maiúsculo, como *git branch -D teste*.

Crie uma nova branch, por exemplo, *git branch mercado-pago*, e mais uma com *git branch whatsapp*, simulando dois funcionários trabalhando em diferentes coisas no mesmo programa. Para trabalharmos com a branch do Mercado Pago digitamos *git checkout mercado-pago*, e podemos, por exemplo, colocar um Javascript simulando o uso do mesmo, fazendo a adição e o commit. Mude para a branch do Whatsapp com *git checkout whatsapp* e veremos que nada colocado do Mercado Pago estará lá, adicione também um Javascript, adicione e commite também. Volte depois pro Mercado Pago com *git checkout mercado-pago*, e pra master com *git checkout master.*

Fundindo Branches

Continuando nosso assunto de branches, vamos fundir as duas branches criadas anteriormente. Digite *git branch* para ver as branches e a branch atual. Digite *git checkout whatsapp* para ver as alterações.

Para fundir as branches, devemos ir na branch de destino (a que receberá a outra branch), no caso, fundiremos a branch do Mercado Pago com a principal (master). Digite *git checkout master* para isso, e para fundir o Mercado Pago com a master, digite *git merge mercado-pago*.

Para fundir a branch do Whatsapp, é o mesmo processo, na branch master, digite *git merge whatsapp*. Numa segunda fusão, ele poderá abrir o editor de configurações e nesse caso, basta digitar *:q*.

Ao dar *git log* novamente, ele mostrará incorporado à branch master, os commits das outras branches que foram fundidas com ela.

Resolvendo Conflitos

Podemos continuar trabalhando numa branch mesmo depois dela ser fundida com outra. Vamos supor agora que vamos alterar um estilo na branch do Whatsapp. Primeiro vá pra essa branch com *git checkout whatsapp* e altere o CSS, como por exemplo, a cor de fundo do site, adicione e commite. Voltando na branch principal com *git checkout master*, vamos fazer outra alteração de cores no CSS, adicionamos e commitamos.

Ao fazer a fusão no master de novo com *git merge whatsapp*, ele dará um conflito de código, e não fundirá as branches. No caso você abrirá o editor, que marcará o código com os conflitos, para ser corrigido manualmente. Após a edição, veja o status, adicione e commite ele, e também podemos

omitir a mensagem, pois o Git saberá que se trata de um conflito, então digite apenas *git commit*, ele abrirá os comentários de configuração, nesse caso digite *:q* e ele concluirá a fusão.

Vamos pra branch do Mercado Pago digitando *git checkout mercado-pago* e vamos editar o Javascript dele, adicione e commite ele, volte pro master com *git checkout master*, edite o mesmo Javascript do Mercado Pago, adicione e commite também. Vamos fundir novamente o branch do Mercado Pago com o master, digitando *git merge mercado-pago* e ele mostrará o conflito novamente. Resolva da mesma forma anterior.

PS: Várias IDEs podem oferecer comparações sobre os conflitos dos códigos dos branches para serem corrigidos pela gente.

Iniciando com GitHub

O GitHub é um serviço de host para repositórios Git. Até o momento, nós criamos nosso repositório Git localmente, ele está apenas no seu computador, para isso nós teremos que colocar ele num repositório remoto, que no caso, usaremos o GitHub. Existem outros semelhantes como o GitLab, o BitBucket, o Phabricator, o Gogs, etc.

Basicamente, para enviar dados do nosso computador pro GitHub usamos o “push”, pra buscar os dados do GitHub usamos o “fetch”, e para puxar dados do GitHub pro nosso repositório no computador usamos o “pull”.

Entre no seu perfil do GitHub pelo navegador, vá em Repositories e em New para criar um novo repositório remoto. Escolha as configurações como o nome do repositório, se ele será público ou privado (no caso, escolheremos público). Veja se é necessário adicionar outras coisas como adicionar um README, o .gitignore, a licença, etc. Depois disso clique em Create Repository.

PS: Caso você queira configurar o SSH para acesso ao GitHub, vá no seu perfil, em Settings e em SSH and GPG Keys. Podemos usar o Git remoto por SSH e por HTTPS.

- **HTTPS:** *https://github.com/nomedousuario/NomeDoProjeto.git*
- **SSH:** *git@github.com:nomedousuario/NomeDoProjeto.git*

Para fazer o upload do seu repositório local pro GitHub, entre na pasta desse repositório e veja com *git status* o atual estado do repositório. Para fazer o “push” (enviar pro GitHub), digite *git push https://github.com/nomedousuario/NomeDoProjeto.git master* (ou a branch que quer enviar). Só com isso, todos os nossos arquivos serão enviados pro repositório remoto, incluindo os commits e tudo mais o que fizemos localmente.

Agora, no nosso repositório, faça uma alteração no HTML, veja o status, adicione e commit o repositório. Até então, as alterações serão locais. Para ele sincronizar com o repositório remoto digite *git push https://github.com/nomedousuario/NomeDoProjeto.git master.*

Para não ficar copiando e colando o link do repositório toda hora, podemos criar um alias para esse comando, utilizando *git remote add origin https://github.com/nomedousuario/Projeto-1.git*. Para ver se foi adicionado, digite *git remote -v*, ele mostrará o repositório para fetch (buscar) e push (enviar). Daí é só usar *git push origin master.*

PS: Lembrando que onde está origin pode ser qualquer nome, mas geralmente origin é o mais usado.

De uma outra forma, criaremos um segundo repositório remoto no GitHub, da mesma forma do primeiro. Nesse caso criaremos o .gitignore e o README (que é usado como uma documentação do projeto).

Vamos clonar esse repositório para a nossa máquina com *git clone https://github.com/nomedousuario/Projeto-1.git*. Caso queira outro nome na pasta clonada, podemos colocar *git clone https://github.com/nomedousuario/Projeto-1.git NovoNomeDoProjeto*.

Podemos só buscar as informações sem alterar o repositório com *git fetch https://github.com/nomedousuario/Projeto-1.git master*, depois podemos dar um *git status* para ver se é necessário fazer o pull do repositório remoto (puxar). Para puxar as alterações do projeto para o nosso repositório local, digite *git pull https://github.com/nomedousuario/Projeto-1.git master.* É recomendado primeiro usar o fetch, depois o pull.

PS: Ao clonar um repositório, ele automaticamente colocará o repositório remoto clonado com o alias "origin". Para ter certeza, digite novamente *git remote -v.* E os comandos push, fetch e pull podem ter a branch omitida, caso você queira fazer as tarefas na branch que você está atualmente, algo como *git push https:/github.com/nomedousuario/NomeDoProjeto.git* ou *git push origin* (se você tiver na master ele fará o envio pra master, em uma branch criada ele enviará pra esta criada, etc.

Pode também ser necessário forçar o fetch e o pull em alguns casos. Para isso, podemos usar comandos como:

git fetch –all
git reset –hard origin/master

Já para o push, podemos usar:

git push origin master –force

Ou, de forma mais segura:

git push origin master –force-with-lease

Simulando Múltiplos Devs

Como já percebemos, num projeto do Git podemos ter vários desenvolvedores trabalhando, no caso, vamos simular múltiplos devs.

Vamos abrir três terminais, onde um será você, o outro será Fulano e o outro será Sicrano.

Vamos clonar o repositório Projeto-1 do GitHub digitando *git clone https://github.com/nomedoususario/Projeto-1 Projeto-1-Fulano* e depois fazer o mesmo com Sicrano.

No caso nós vamos salvar com nomes diferentes por exemplificar no mesmo computador, no entanto, num projeto real, cada um trabalharia num computador diferente e por isso, as pastas poderiam ter o nome original mesmo.

Vamos supor que Fulano começa a alterar o código, como adicionar um estilo CSS. Depois ele verá o status e adicionará, e o commit no caso ele fará com o seu e-mail, assim: *git commit -m "Alterando CSS" –author="Fulano <fulano@gmail.com>"*. Daí para enviar pro GitHub basta digitar *git push origin master* (lembrando que ao clonar ele automaticamente coloca o repositório no alias origin).

Agora vamos pra Sicrano, supondo que ele iria começar a trabalhar no mesmo projeto. Para ele ver se o projeto teve alterações, digite *git fetch origin master* e *git pull origin master,* e ele vai puxar as últimas atualizações feitas no repositório remoto. Daí ele aproveita e faz outras alterações, pode ser no CSS mesmo, e daí ele vê o status, adicionará e commitará com *git commit -m "Nova Alteração no CSS" –author="Sicrano <sicrano@gmail.com>"*. Daí ele também fará o push com *git push origin master.*

PS: Isso estamos simulando numa mesma branch, mas é recomendado que cada desenvolvedor tenha sua própria branch para trabalhar.

Aí você, como coordenador do projeto, você dará o *git fetch origin master* e o *git pull origin master* para ver as alterações que os outros devs fizeram.

Num projeto real, vários desenvolvedores podem estar no mesmo projeto, e aí é toda hora fetch, pull e push, além de ser necessário resolver conflitos, como exemplificado anteriormente.

Fazendo Pull Request

Como já exemplificamos, não é recomendado fazer tudo na branch principal, e por isso é recomendado criar branches para cada coisa que será manipulada no repositório, e depois fundir elas com a branch principal.

Vamos continuar com dois devs simulados, você e o Sicrano. Vamos supor que o Sicrano vai criar uma nova branch, nós podemos usar o checkout com a opção -b para ele, além de criar, automaticamente ele muda pra mesma branch recém-criada, ficando assim: *git checkout -b svg*.

Nessa nova branch, vamos colocar algumas imagens. Adicione, commite com o nome do usuário *git commit -m "Adicionando Imagens" –author="Sicrano <sicrano@gmail.com>"*, e depois dê o push, que não será no master, mas nessa branch criada, com o *git push origin svg*.

Aí, no site do GitHub, ele identificará o Pull Request do SVG, aí você e sua equipe vai ver no repositório remoto e na notificação, clique em Compare & Pull Request, aí você poderá colocar uma mensagem e clicará em Create Pull Request, daí ele verificará se há conflitos e, caso não haja, aparecerá a opção Merge Pull Request para fundir a branch do outro usuário com a principal. Você pode ir em Commits e ver as alterações no Commit desse outro usuário, além de mandar mensagem pra ele, caso algo esteja errado ou incompleto.

Sicrano poderá continuar adicionando e commitando nessa branch que ele criou. Vamos fazer mais uma alteração em algo, adicionar e commitar como Sicrano, e fazer o push.

Para fundir, basta clicar em Merge Pull Request, e é recomendado fechar essa branch criada.

Daí, no seu computador, basta dar um fetch e um pull no repositório master pra sua máquina.

Fazendo Fork de um Repositório

Para contribuir com projetos de software livre, podemos fazer um fork de um repositório de outro usuário, que nada mais é do que copiar um repositório pro seu perfil do GitHub.

Para isso, vá no repositório desejado e clique em Fork. Lá você poderá alterar o nome, decidir se só copia a branch principal ou copia todas, etc. Depois clique em Create Fork.

Daí, podemos clonar nosso repositório forkado com *git clone https://github.com/nossousuario/NomeDoProjeto.git*, daí fazemos as alterações, daí nós adicionamos, commitamos e damos o push pro seu repositório. Para mandar nossas modificações pro usuário criador original do repositório, clique em Contribute e depois em Pull Request, e siga as mesmas instruções ensinadas anteriormente.

Issues

Nós podemos também relatar bugs, sugerir melhorias e outros tipos de comentários em um repositório do GitHub.

Para isso, vá no repositório desejado, em Issues e clique em New Issue, escreva o título e a descrição, e clique em Create.