BandTec
DIGITAL SCHOOL

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Pesquisa e Inovação I

Introdução ao Git/GitHub

Sumário

O que é Git

Git é um sistema de controle de versão de arquivos (ou sistema de versionamento). Quando um diretório está gerenciado pelo Git, temos o controle sobre os arquivos e diretórios criados, alterados e excluídos nele, podendo saber tudo que foi feito (inclusive por quem e quando), bem como "desfazer" operações. Foi criado pelo "pai" do Linux, **Linus Torvards**, sendo lançado em 2005. É possível usá-lo nos principais sistemas operacionais da atualidade (Windows, Linux e macOS).

Uma das principais inovações do Git frente aos sistemas de versionamento anteriores, é o fato de que cada diretório gerenciado por ele (chamado de **Repositório**) possui o histórico completo das alterações realizadas e registradas via "commit" (falaremos sobre esse termo adiante). Mesmo assim, é possível conectar um repositório **local** a um **remoto**, possibilitando que o versionamento também fique em outro computador (inclusive na internet).

Um resumo de como funcionam os repositórios local e remoto pode ser visto na ilustração da Figura 1.

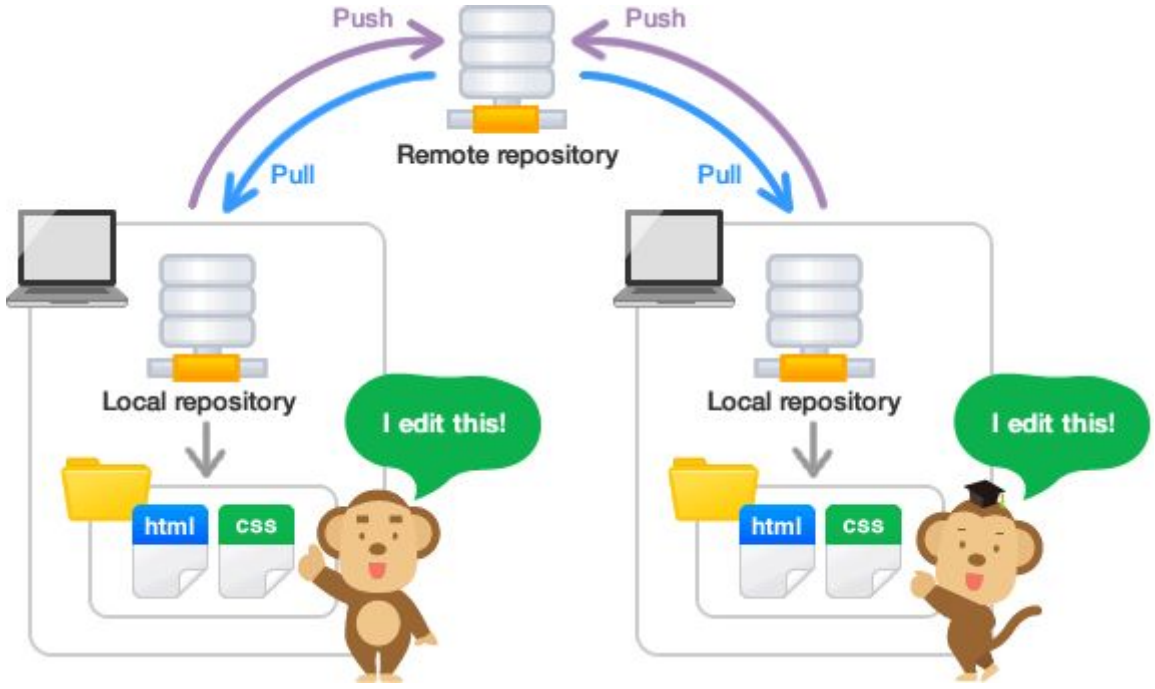

Figura 1: Funcionamento de repositórios local e remoto no Git
Fonte: https://kevintshoemaker.github.io/StatsChats/GIT_tutorial.html

O Git, na prática, é um protocolo, conjunto de comandos executáveis em linha de comando. Porém, existem inúmeras opções de interfaces gráficas como:

- Integrações com o Git em IDEs (Eclipse, IntelliJIdea, NetBeans etc);
- Programas como TortoiseGit, Gitk, SourceTree, GitHub Desktop etc.

Serviços de hospedagem Git na Internet

Como os sistemas de versionamento normalmente são usados para repositórios acessados por um grupo de pessoas, normalmente existe um repositório remoto para centralizar e

unificar seu conjunto de arquivos, diretórios e histórico de operações. Esse repositório remoto pode estar numa rede interna da empresa ou na internet.

Os principais serviços de hospedagem de Git na Internet são: **GitHub** (github.com), **GitHub** (github.com) e **Bitbucket** (bitbucket.org). A diferença entre eles está nas limitações quanto a repositórios privados nas modalidades de uso grátis e nas funcionalidade gráfica de gerenciamento. Todos são 100% compatíveis com os comandos Git ensinados aqui, porém, o foco aqui será o **GitHub**. O GitHub também pode ser configurado como provedor numa rede local.

Criando um repositório no GitHub

Vamos criar nosso primeiro repositório no GitHub (github.com). É necessário criar uma conta de forma gratuita e confirmá-la em seu email. Feito isso, basta clicar no ícone **[+]** e escolher a opção ***New Repository***, como mostra a Figura 2.

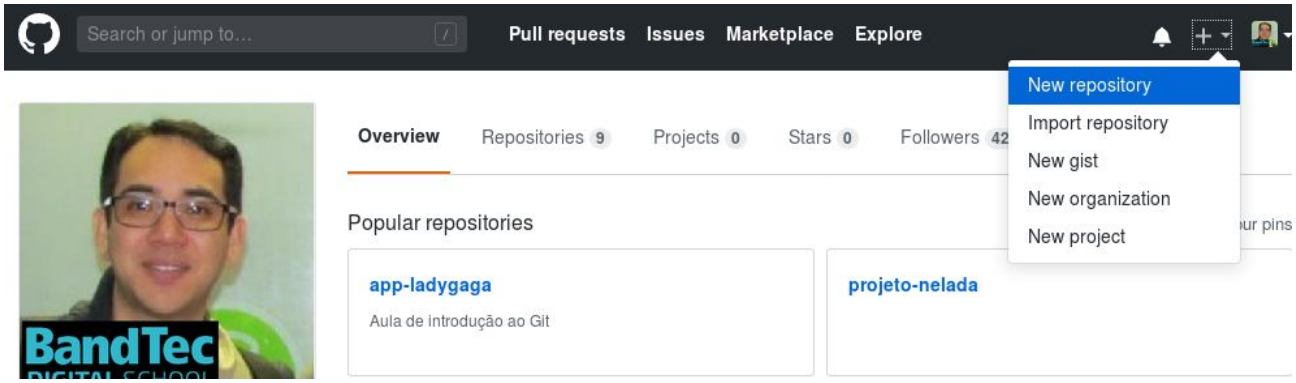

Figura 2: Solicitando a criação de um repositório Git no GitHub

Você será encaminhado ao formulário de criação de repositório. É possível ver um exemplo desse formulário na Figura 3.

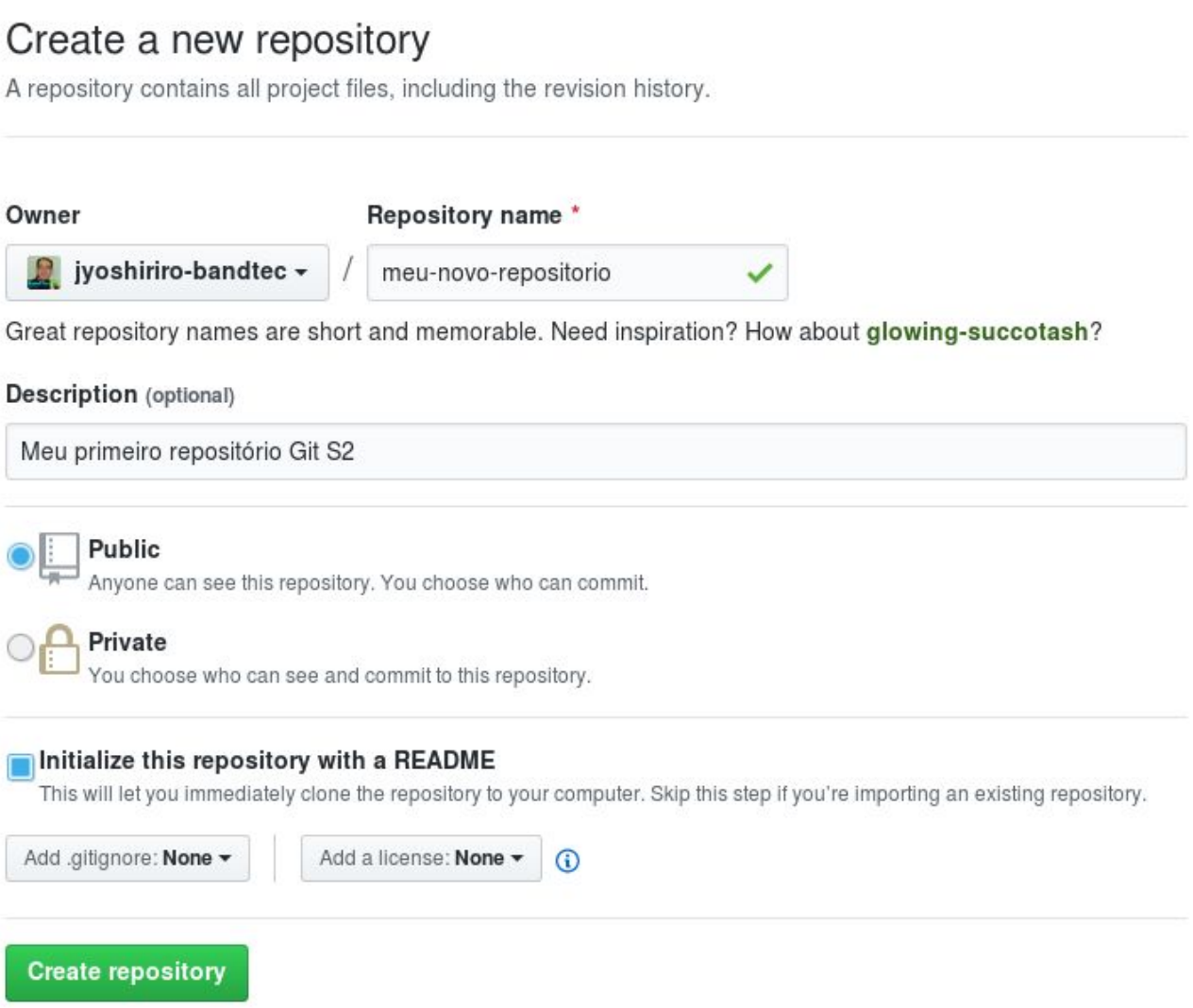

Figura 3: Formulário de criação de um repositório Git no GitHub

Nele, informe o nome do repositório (***Repository name***), cujo padrão é "**separado-por-hifen**". Opcionalmente, informe sua descrição (***Description***).

Defina ainda a visibilidade do projeto, ou seja, se é **Public** ou **Private** .

- ***Public*** - Qualquer pessoa poderá ver e até 'baixar" o projeto, mas só quem for membro poderá alterar seus arquivos.
- ***Private*** - Somente pessoas autorizadas poderão ver ou "baixar" o projeto. Desde janeiro de 2019, o GitHub permitiu a criação de projetos privados sem custo contanto que só tenham até 3 (três) colaboradores. Caso seja necessária uma quantidade maior de colaboradores, é necessário aderir ao plano pago do GitHub[1].

Caso você esteja criando um repositório que não será associado a um repositório git local, **marque** a opção ***Initialize this Repository with a README*** , caso contrário, terá que usar uma série de comandos para poder começar a usar o repositório. Na dúvida, deixe esse item **marcado**.

[1] **Preços dos planos pagos do GitHub** - https://github.com/pricing

Na opção **add .gitignore** você indica a tecnologia principal que vai usar no repositório (ex: Java, Arduino, Node etc).

A função do arquivo **.gitignore** é indicar que arquivos e/ou diretórios (pastas) serão ignorados do controle de versão. Isso porque quase toda plataforma de programação tem um conjunto de arquivos que não precisam ser gerenciados porque são criados, alterados e excluídos automaticamente enquanto estamos programando. Esse arquivo no **git** garante que só arquivos os quais nós manipulamos diretamente serão gerenciados.

Um arquivo **.gitignore** no diretório raiz do repositório indica o que será ignorado em todos os diretórios (pastas). Porém, é possível criar um .gitignore em cada diretório (pasta) que achar necessário. Se fizer isso, as configurações nele irão valer para aquele diretório e seus sub-diretórios.

Se não indicar o arquivo **.gitignore** no momento da criação, sem problemas. É possível adicionar esse arquivo manualmente facilmente depois.

Na opção **add a licence** você indica a licença[2] sobre a qual quer que seu código fonte fique. Indicar ou não algo aqui não influencia em nada no funcionamento do seu repositório.

Após clicar em ***Create repository***, já é possível manipular e até ver o repositório no GitHub, como mostra a Figura 4.

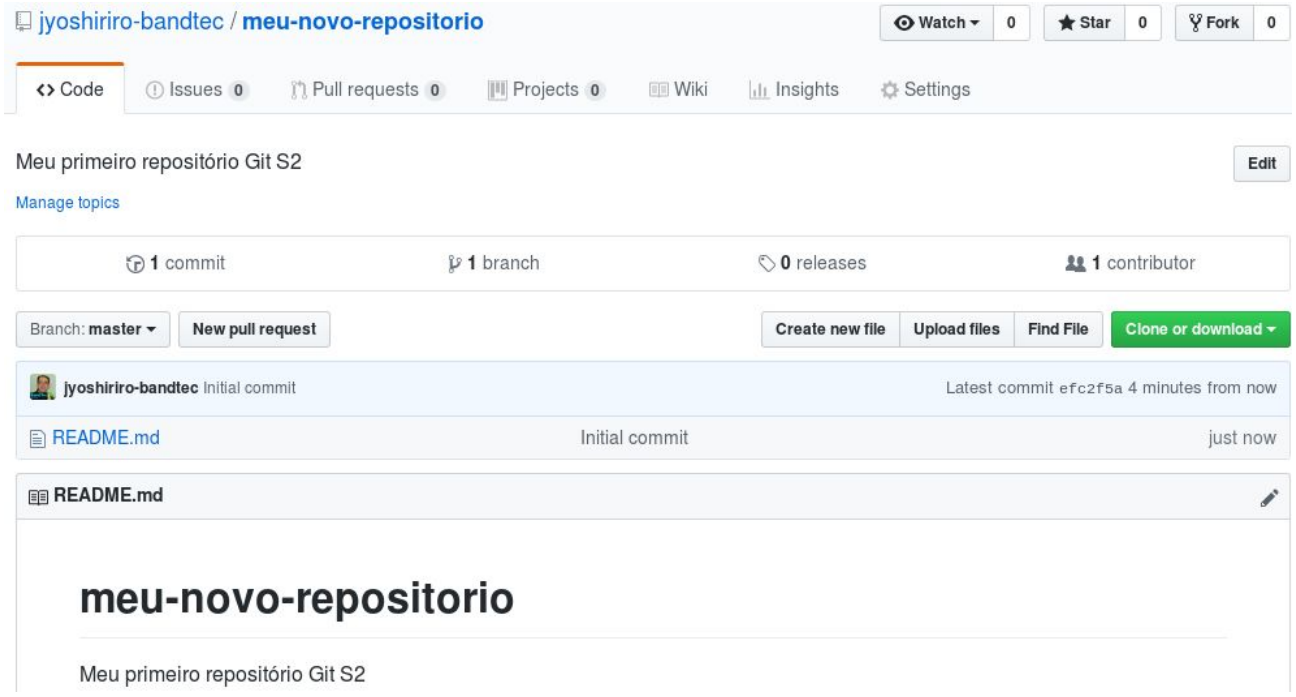

Figura 4: Repositório Git recém criado no GitHub

[2] **Licenças open-souce** https://imasters.com.br/desenvolvimento/como-funcionam-as-licencas-open-source

Adicionando membros a um repositório

Uma das principais motivações para a adoção de um **repositório remoto** é a centralização e unificação de arquivos e operações de diferentes pessoas em um projeto. Assim, para definir quem e como pode interagir num repositório no GitHub, é necessário gerenciar seus **membros**. Para adicionar um membro, basta ir na aba ***Settings*** e depois, no menu lateral esquerdo, escolher ***Collaborators*** (Vide Figura 5).

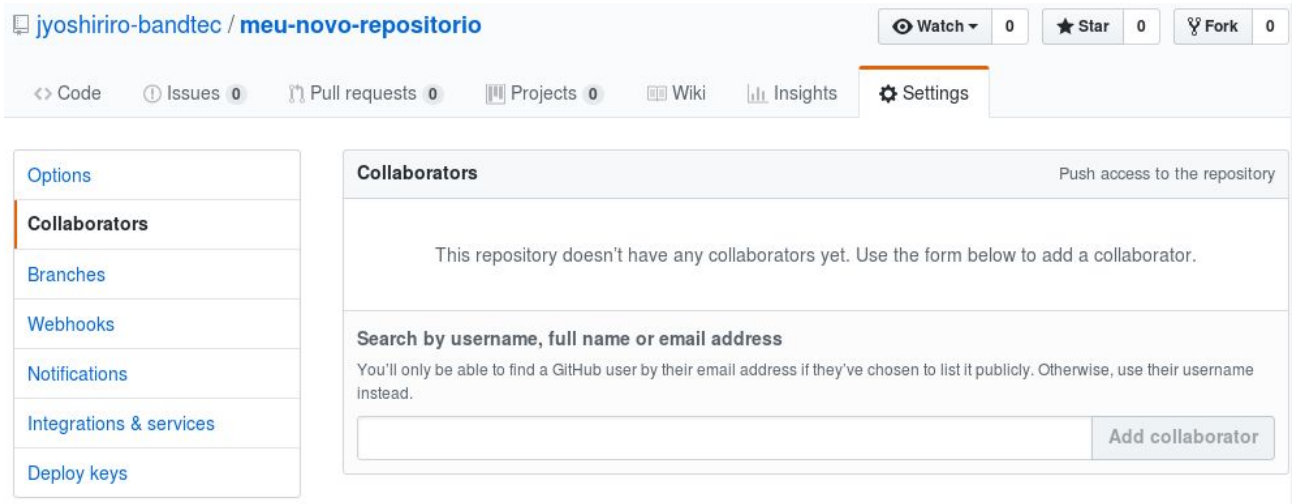

Figura 5: Tela de gerenciamento de membros num repositório no GitHub

Na página de gerenciamento de membros, do lado do botão **Add collaborator**, você digita o nome ou o login do usuário que quer adicionar em seu repositório que o GitHub vai sugerindo. Quando for a pessoa que você queria, clique no botão e verá que a foto e nome do usuário vai para cima, indicando que foi convidado para seu repositório (Vide Figura 6).

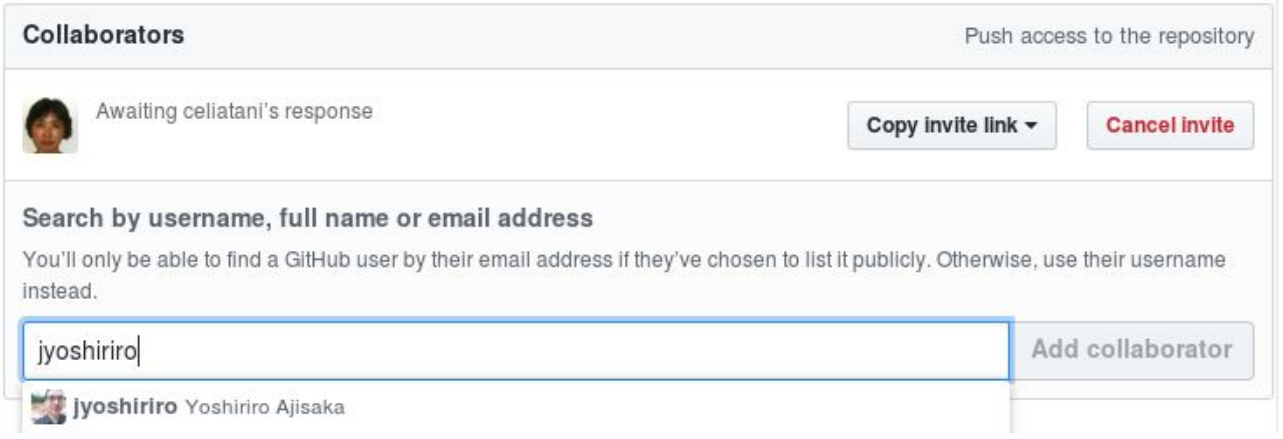

Figura 6: Formulário de inscrição de membros num repositório GitHub

Cada membro adicionado **precisa confirmar sua inscrição** ou em seu email ou em seu perfil no GitHub para que possa enviar alterações ao repositório.

Se for o caso, é possível remover um membro nessa mesma tela, pois depois que ele aceita, um botão para removê-lo fica ao lado de seu nome.

Instalando o Git

Para poder usar o Git, pode ser necessário instalá-lo em seu computador, pois não é um programa nativo em alguns sistemas operacionais.

Para saber se o Git já está instalado em seu SO, abra um terminal e tente executar o comando…

git --version

Se você ver uma descrição de versão, o Git já está instalado. Caso contrário, receberá a mensagem de comando não encontrado.

Linux: Na maioria das distribuições o git está disponível como pacote sob o nome **git**. Assim, basta fazer um **apt install git** ou **dnf install git** ou **yum install git**, dependendo da distribuição usada.

macOS: O git costuma vir instalado nesse SO. Porém, caso não esteja, baixe o instalador em **https://git-scm.com/download/mac** e siga a instruções.

Windows: Baixe o instalador em **https://git-scm.com/download/win** e siga a instruções. Após instalado, use o programa **Git Bash** para executar os comandos git. Trata-se de um terminal com cores customizadas para o git e que aceita comandos do Linux ao invés de comandos do prompt do Windows. Por exemplo, pode executar **ls**, **cat**, **clear** etc. Outra característica desse terminal é que guarda o histórico de comandos (recuperável usando-se as setas do teclado) mesmo que seja fechado.

Configurando seu usuário local Git

Após a instalação, é necessário definir os **nome** e **email** do usuário local Git. Para isso, basta executar os comandos…

git config --global user.name "Seu Nome"
e
git config --global user.email "seu_email@seu_provedor.com"

Principais comandos git

A seguir, vamos interagir com o repositório remoto usando os principais comandos do Git.

git clone - Obtendo um repositório remoto

Quando você deseja começar a trabalhar num repositório que já existe, precisa obtê-lo remotamente. Para isso, use o comando...

git clone

No diretório que deseja que fique o diretório gerenciado pelo git, **não crie o diretório manualmente**. Apenas execute o seguinte comando:

git clone https://github.com/seu-usuario-github/seu-repositorio-git-lab/

Dica: Esse endereço após o *git clone* é mesmo que seu repositório tem ao acessá-lo pelo navegador.

Serão solicitados seu login e senha do GitHub. **Um diretório com o mesmo nome do repositório será criado** pelo Git. Assim, basta fazer um **cd seu-repositorio-git-lab** para entrar nele.

git pull - Obtendo as atualizações do repositório remoto

Uma vez que seu repositório já está clonado, sempre que quiser obter as alterações do repositório remoto em seu repositório local, use o comando...

git pull

Com ele, todas as alterações que eventualmente não estejam registradas em seu repositório local passarão a existir. Esse comando também sincroniza uma série de configurações do repositório remoto com o local.

git status - Verificando se há alterações locais

Para saber se houve alterações em arquivos no repositório local, use o comando…

git status

Se não houver nenhuma alteração, você deverá ver uma saída como esta:

```
On branch master
Your branch is up to date with 'origin/master'.
nothing to commit, working tree clean
```

Alterações em repositórios Git

Quanto ao status de gerenciamento de arquivos, o Git possui 3, que são:

Unstaged: Arquivos ainda sem monitoramento, ou seja, externos ao git (arquivos recém criados, alterados ou excluídos sobre os quais não fizemos o **git add**, comando sobre o qual já falaremos)

Staged: Arquivos prontos para o envio (**commit**). São todos sobre os quais fizemos o comando **git add**.

Commited: Arquivos cujas alterações foram enviadas (que sofreram **commit**).

A Figura 7 ilustra a sequência cronológica dos status.

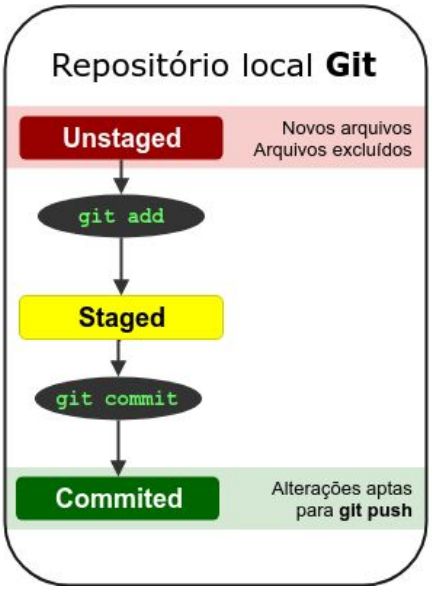

Figura 7: Status de arquivos num repositório Git.

A seguir, um exemplo de saída do **git status** com os 3 tipos de status:

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        new file:   listagem.txt
        modified:   receita.txt
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)
        deleted:    poema.txt
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        declaracao.txt
```

git add - Adicionando inclusão e/ou exclusão de arquivos

No Git, criação e a exclusão de arquivos são operações que precisam ser registradas explicitamente antes de serem enviadas ao controle de versão. Usa-se, então, o comando **git add**. Exemplos:

Você acaba de criar o arquivo **poema.txt**, no diretório raiz do repositório:
git add poema.txt

Você acaba de criar o arquivo **japao.txt**, no diretório **paises** do repositório:
git add paises/japao.txt

Você acaba de excluir o arquivo **receita.txt**, no diretório raiz do repositório:
git add receita.txt

Você acaba de criar e/ou excluir vários arquivos do repositório, inclusive em vários subdiretórios, e quer adicionar tudo de uma só vez:
git add .

Atenção: O **git add** ainda **não** registra as alterações no controle de versão do repositório! Uma vez feito isso, a(s) alteração(ões) registradas estão prontas para o envio (**commit**). Esse sim, faz com que o controle de versão contenha as alterações.

git diff - Exibindo as alterações em arquivos

As alterações nos arquivos ainda não enviadas via **commit** podem ser vistas usando-se o comando

git diff

Após a execução desse comando, a saída é como a seguir:

```
diff --git a/listagem.txt b/listagem.txt
index cf2f7b0..9430a63 100644
--- a/listagem.txt
+++ b/listagem.txt
@@ -1 +1,2 @@
-frutas
+legumes
+massas

diff --git a/receita.txt b/receita.txt
deleted file mode 100644
index f702bcd..0000000
--- a/receita.txt
@@ -1,3 +0,0 @@
-ingredientes
```

```
-instruções
-cuidados
```

Atenção: Arquivos recém criados e sobre os quais não foi executado o comando **git add** não aparecem na saída do comando **git diff**.

Verificando se há alterações no repositório remoto

Caso você deseje saber se existem alterações no repositório remoto que você ainda não recuperou via **git pull**, basta executar estes 2 comandos…

git fetch

depois

git diff origin/master

A saída é no mesmo estilo que a que vimos para o **git diff**.

git commit - Enviando alterações para o controle de versão

O Git não registra alterações por arquivo e sim por "pacotes" de envio de alterações (**commits**). Num commit podem estar registradas criação, alteração ou exclusão de 1 ou mais arquivos. Todo **commit** deve possuir uma mensagem informada pelo usuário Git.

Por exemplo, vamos supor que você já fez todos os **git add** que precisava. Basta executar o comando...

git commit -m "Motivo das alterações que estou enviando"

Com isso, se você, por exemplo, criou 1 arquivo, alterou 4 e excluiu 2, seu **commit** contemplará 7 arquivos afetados.

A mensagem que resume o motivo daquele commit é obrigatória e com limite de **72 caracteres**.

Atenção: o conceito de **commits** não impede que possamos reverter alterações em um único arquivo, mesmo que suas alterações tenham ido junto às de outros.

Dúvida comum: É possível enviar a alteração de apenas 1 arquivo num **commit**? Sim. Basta fazer o **git add** somente sobre o arquivo desejado antes do **git commit**.

git log - Obtendo a listagem de commits no controle de versão

Para ver quais commits estão registrados no controle de versão, use o comando

git log

Ele vai gerar uma saída como o exemplo a seguir:

```
commit 65ebf9015f9c8f03735fd5be014b835368521755 (HEAD -> master)

Author: José Ruela <jruela@gmail.com>

Date:   Mon Feb 15 08:20:12 2019 -0300

    correções ortográficas

commit c28447fc0f9aa7dae89f5738afe0d5081270f024

Author: José Ruela <jruela@gmail.com>

Date:   Mon Feb 15 07:43:15 2019 -0300

    novas alterações gerais
```

Notou um código alfanumérico após cada palavra “commit”? Esse é o **código identificador** de cada **commit**. Sabe aquele “código de rastreamento” de encomendas físicas? É como se fosse isso. O Git associa cada identificador desse a um conjunto de alterações.

git checkout - Revertendo alterações

Você já teve aquele pensamento *“se eu pudesse voltar no tempo faria aquilo diferente…”*? Bom, no mundo real não podemos voltar no tempo, mas o Git nos permite, como que, fazer isso em relação a nossos arquivos. Podemos pedir para que um arquivo, ou até o repositório inteiro, “volte no tempo” e fique como era até determinado commit.

Revertendo alterações num arquivo

Vamos supor que deseja que um o arquivo **ingredientes.txt** volte a ficar como era até um determinado **commit**. Você faz o **git log** e localiza o **identificador** do **commit** desejado. Então, basta fazer:

git checkout 63fe1916beddc6ad3a986e4bcf74a9e2ee9f501d ingredientes.txt

Caso você queira que o mesmo arquivo fique exatamente como está atualmente no repositório remoto, faça 2 comandos:

git fetch

depois

git checkout origin/master ingredientes.txt

Revertendo alterações em todo o repositório

Vamos supor que deseja que todo o repositório local fique como estava até determinado commit. Encontre o identificador do commit e execute…

git checkout 63fe1916beddc6ad3a986e4bcf74a9e2ee9f501d

Esse comando faz o repositório inteiro “voltar no tempo” e ficar exatamente como ficou após o commit informado. Porém, fica bem complicado realizar alterações para serem enviadas para o repositório remoto a partir daqui.

Se quiser que o repositório local fique como o remoto, faça 2 comandos:

git fetch

depois

git reset --hard origin/master

git push - Enviando os commits para o repositório remoto

Mesmo que você tenha criado seu repositório git a partir de um clone de um remoto, registrar os commits não os envia para ele. É necessário informar, explicitamente, que você deseja enviar todos os seus commits locais para o repositório remoto. Para isso, use o comando:

git push

Esse comando irá solicitar seus login e senha do GitHub. Ao entrar com eles corretamente e se o push funcionar sem problemas, você deve ver uma saída como a seguir.

```
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (5/5), 449 bytes | 224.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://github.com/jruela/java-queiro
c28447f..65ebf90  master -> master
```

Pode ocorrer que, ao tentar o **push**, você seja informado que ele foi rejeitado (**rejected**). Isso ocorre quando alguém fez um push para o mesmo repositório remoto após seu último **pull**. Nesse caso, você terá uma saída como esta:

```
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/jruela/java-queiro'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Para resolver isso, basta fazer um **git pull**. Se aparecer uma tela meio estranha (editor de textos nano), apenas tecle **CTRL+X**. É apenas um texto informativo. Feito isso, execute o **git push** novamente.

Resolvendo conflitos em arquivos

Pode ocorrer que, ao tentar um **git pull**, você se depare com um problema de conflito (**conflict**). Isso ocorre quando outra pessoa mexeu em um mesmo arquivo que você e o enviou ao repositório remoto após seu último **pull**. Nesse caso a saída será como esta:

```
Unpacking objects: 100% (3/3), done.
From https://github.com/jruela/java-queiro
   65ebf90..29a5fca  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Caso isso ocorra, você tem 2 saídas: Abrir o(s) arquivo(s) com conflito e ajustar manualmente seu conteúdo ou fazer o(s) arquivo(s) com conflito “voltar” a uma versão anterior.

Ajustar conflitos manualmente

Caso um ou mais arquivo seus tenham sofrido com conflito, ao abrir um deles, terá coisas um tanto estranhas em seu conteúdo. Vejamos um exemplo de como fica um arquivo desses a seguir.

```
# java-queiro
<<<<<<< HEAD
se eu não te amasse tanto assim...
=======
se eu num te amasse um tanto assim...
>>>>>>> 4e35b01632c66c0b655eb79a061f8e9724d47058
```

Confusão, não? E isso é um exemplo de um arquivo bem pequeno com uma linha de conflito. Se optou por ajustar manualmente, verifique como o conteúdo deve ficar e certifique-se de tirar TODAS as linhas com <<<<<<< e >>>>>>>, pois são marcações de conflito do Git.

Após fazer os ajustes necessários, execute o comando...

git commit -a

Provavelmente você verá a tela do editor nano novamente (exemplo na Figura 8). Basta usar o **CTRL+X** para sair dela. É apenas uma mensagem informativa.

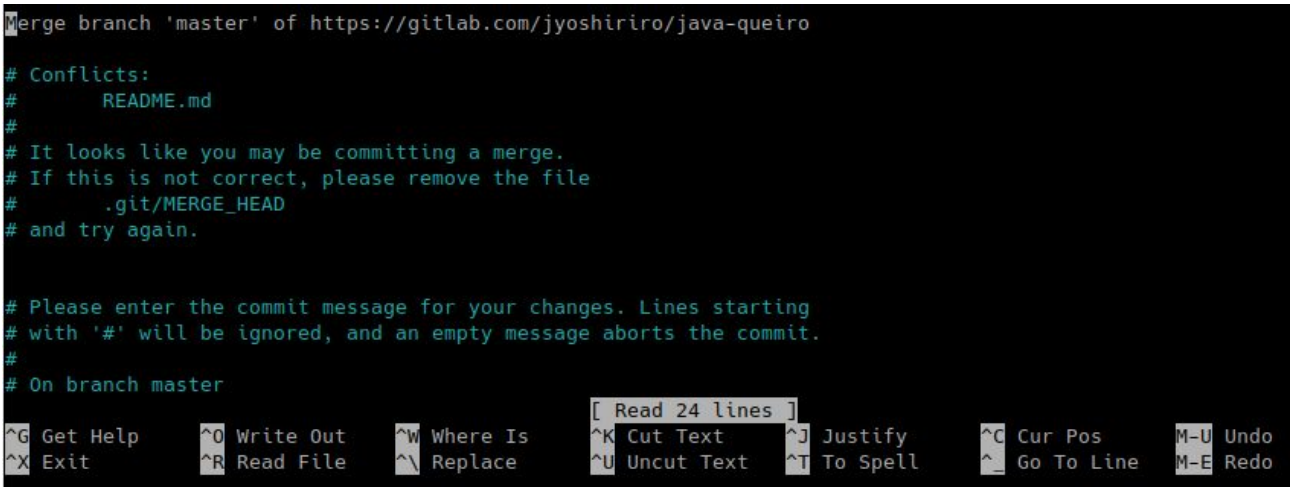

Figura 8: Tela do editor nano após um “git commit -a”

Resolvendo conflitos revertendo a versão

Caso queria simplesmente reverter a versão do arquivo para a de algum commit ou para a versão do repositório remoto, siga as instruções do tópico **“git checkout - Revertendo alterações”** e depois execute o comando...

git commit -a

Provavelmente você verá a tela do editor nano novamente (exemplo na Figura 8). Basta usar o **CTRL+X** para sair dela. É apenas uma mensagem informativa.

Para ajudar no entendimento de todo o ciclo entre repositórios local e remoto, observe o diagrama na Figura 9.

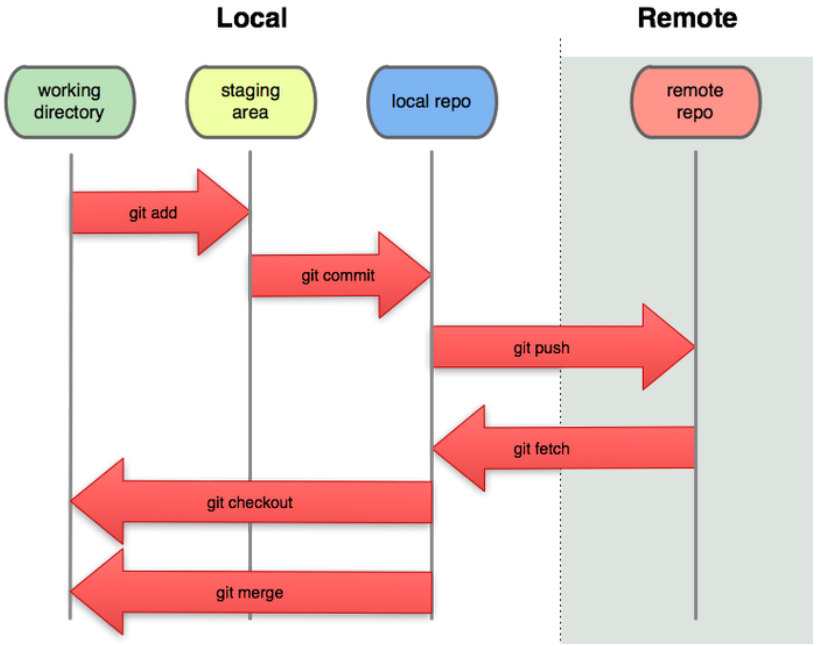

Figura 9: Comandos e status dos repositórios local e remoto Git
Fonte: https://greenido.files.wordpress.com/2013/07/git-local-remote.png?w=696&h=570

Prática Git / GitHub

Para colocar em prática os conceitos aqui estudados, vamos realizar a seguinte prática:

a) Crie um novo repositório remoto no GitHub. Adicione pelo menos 1 membro.
b) Faça o clone desse repositório
c) Adicione pelo menos 1 arquivo. Faça o commit e o push. Peça para pelo menos 1 dos membros fazer o mesmo
d) Todos devem executar os comando necessários para que, ao final, todos tenham a mesma versão do repositório local, com todos os arquivos de todos os membros
e) Combinem entre 2 membros que mexam no mesmo arquivo (façam alterações diferentes). O primeiro, claro, vai conseguir fazer o push, mas o segundo não. Execute os comando necessários para resolver o conflito
f) Altere algum arquivo local e depois o faça ficar como sua cópia no repositório remoto

Bibliografia

AQUILES,Alexandre; FERREIRA, Rodrigo. Controlando versões com Git e GitHub. São Paulo: Casa do Código, 2014.

Git. Git - Reference. Disponível em <https://git-scm.com/book/pt-br/v2/>. Acesso em: 15 de fevereiro de 2018.