

# Git e GitHub

EXPLORANDO OS ELEMENTOS FUNDAMENTAIS DO  
CONTROLADOR DE VERSÕES

Elaborador: Fagner de Oliveira Bernardo  
Contato: [bernardo.fagner@hotmail.com](mailto:bernardo.fagner@hotmail.com)  
[oliveira.fagnerbernardo@gmail.com](mailto:oliveira.fagnerbernardo@gmail.com)

# Conteúdo Programático – Parte 1

---

- Definições;
- Princípio de funcionamento do Git;
- Instalação configuração do Git;
- Ciclo de vida do Git;
- Instruções básicas;



# Conteúdo Programático – Parte 2

---

- Logs;
- Desfazendo ações;
- Branches.;
- Merge;
- Rebase.

# Conteúdo Programático – Parte 3

---

- Princípio de funcionamento do GitHub;
- Repositórios remotos (criação manipulação);
- Vincular repositório remoto a repositório local;
- Clonagem de projetos;
- Fork (duplicata) de repositórios remotos;
- Enviar arquivos do repositório local para o repositório remoto e;
- Enviar arquivos do repositório remoto para o repositório local;

O GIT

---



**git**



# O GIT

---

- GIT é um Sistema de Controle de Versão e tem a finalidade de gerenciar diversas versões do seu projeto ou documento.
- Todos os comandos `<git ...>` apresentados neste curso estarão entre os símbolos menor '`<`' e maior '`>`'.

# Funcionamento do controle de versão

---

- O Git guarda os ESTADOS dos arquivos de cada versão do projeto através de fotos (snapshots), desta forma, pode-se ter diferentes configurações do projeto, facilitando o retorno a pontos específicos da implementação. Outros sistemas de controle versão atualizam o documento, tornando difícil retornar a um ESTADO específico do seu projeto.

# Instalação do Git

---

- No Linux o Git já vem instalado, por padrão, mas caso sua distribuição não o contenha, acesse <https://git-scm.com/download/linux> para realizar a instalação.
- No Windows é necessário utilizar um instalador que pode ser encontrado em <https://git-scm.com/> acessando a opção 'Download' logo na página.



# Configuração do Git

---

- O git guarda suas informações em 3 lugares:
  - no git config do sistema como um todo;
  - no git config do usuário e;
  - no git config do projeto em seu diretório.
- Para passar informações para todos os repositórios, usaremos o config do usuário, que é o global. Para realizar estas configurações abra o terminal em qualquer diretório e execute os seguintes passos:

# Configuração do Git

---

- Configurar o nome de usuário global:
  - `<git config --global user.name "nome de usuário">` {Pode passar o nome usando aspas duplas e espaço entre as aspas duplas}.

“Este nome é o que será usado em alguns momentos da interação com o git, é altamente recomendado manter este nome igual ao nome usado no GitHub.”
- Configurar o e-mail vinculado:
  - `<git config --global user.email "email valido">`

# Configuração do Git

---

- Consultar o dado registrado:
  - `<git config user.name>` “Visualiza o nome registrado”
  - `<git config user.email>` “Verifica o email cadastrado”
  - `<git config core.editor>` “Verifica o editor de texto padrão”
  - `<git config --list>` “Lista diversas opções de configuração”

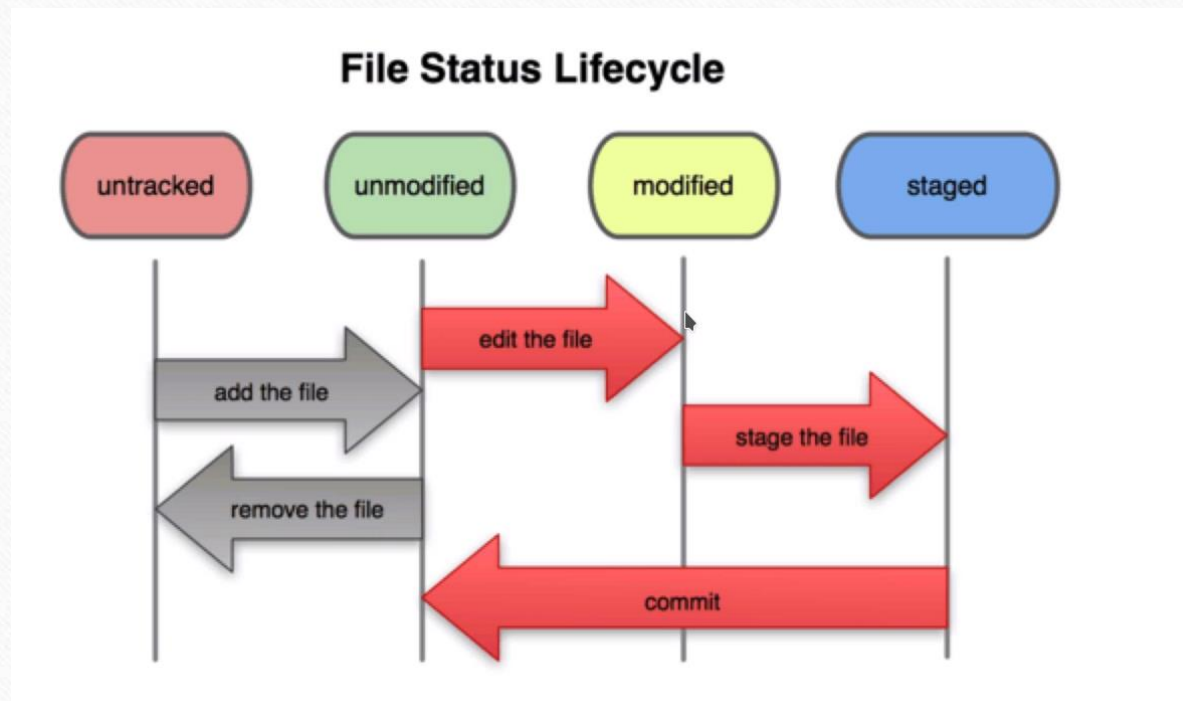


# Iniciando os trabalhos com o Git

---

- Inicializando o diretório vinculado ao Git:
  - “Primeiramente, escolha um diretório e crie uma pasta com o nome do projeto que você deseja controlar”.
  - Entre no diretório e dê o comando para inicializa-lo, para que o Git fique monitorando-o constantemente, faça: **<git init>**. Use o comando **<ls -la>** para verificar a criação de alguns arquivos e diretrizes.
  - Daí você pode usar **<cd .git/>** para acessar o diretório de configuração onde você poderá ver informações sobre o branch atual, configurações, descrições e mais informações.

# Ciclo de vida dos estados dos arquivos



# Ciclo de vida dos estados dos arquivos

---

- 1 - **untracked** (não rastreado/monitorado): Significa que o arquivo acabou de ser adicionado no repositório mas ainda não foi visto pelo git, ou seja, o git não o reconhece como um arquivo a ser controlado.
- 2 - **unmodified**: Assim que o arquivo foi adicionado ao git (add) ele passa a este estágio.



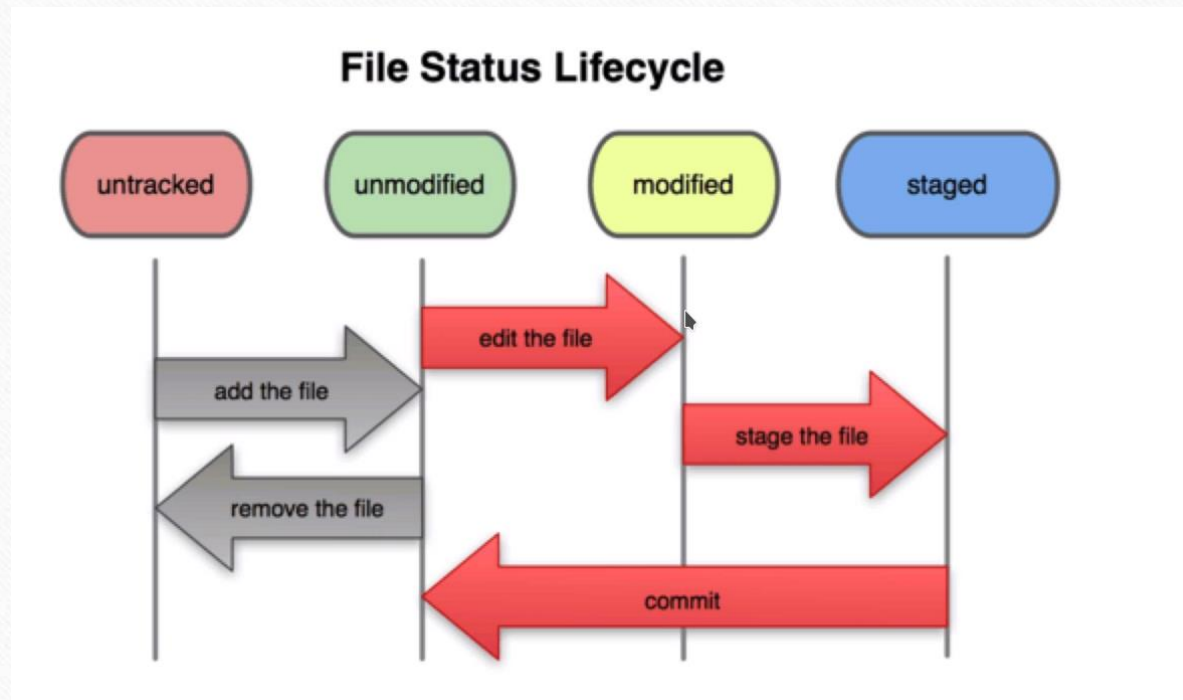
# Ciclo de vida dos estados dos arquivos

---

- 3 - **modified**: Um arquivo que já foi adicionado ou modificado (editado) passa para este estágio.
- 4 - **staged**: Significa que o arquivo está em uma área onde será criada a versão deste documento (pré-commit). Assim que um commit (submissão) for realizado, o arquivo volta ao estado 2.

“O **commit** é o comando utilizado para que o git crie uma imagem do arquivo, ou seja, uma versão do mesmo. Dessa forma, se o arquivo for modificado posteriormente, será possível retornar a esta versão caso necessário.”

# Ciclo de vida dos estados dos arquivos



# Comandos básicos do Git

---

```
Fagner@DESKTOP-9KHFC3G MINGW64 ~/Documents/Git/teste (master)
$ git status
On branch master
nothing to commit, working tree clean

Fagner@DESKTOP-9KHFC3G MINGW64 ~/Documents/Git/teste (master)
$ git log
commit fd342dd59d2cd5ae82481022e816bc6dbda6e275 (HEAD -> master)
Author: bernardofagner <bernardo.fagner@hotmail.com>
Date:   Tue Dec 12 08:59:48 2017 -0200

    Primeiro commit

Fagner@DESKTOP-9KHFC3G MINGW64 ~/Documents/Git/teste (master)
$ _
```



# Status do Repositório e Adição de Arquivos

---

- Para listar o estado atual do repositório use o comando `<git status>;`
- Para mudar o estado de um arquivo de **untracked** para **unmodified** use o comando `<git add “nome-do-arquivo.extensão”>` (sem aspas). Você pode usar `<git add -A>` para incluir todos os arquivos do diretório de uma vez só!

# Criação de Uma Versão

---

- Para gerar um commit use `<git commit -m "mensagem">` (-m é usado para passar uma msg junto com o commit).
- O resultado da ação apresenta a seguinte mensagem: **[master (root-commit) 8413d32] mensagem.**
  - "master" indica o branch atual;
  - "8413d32]" é o código hash para a versão e;
  - "mensagem" é o comentário associado ao commit.

# Visualização de Alterações

---

- O comando `<git diff>` permite que possamos ver as modificações no documento que ainda não estão sendo monitoradas (unmodified), ou seja, antes de se usar o `<git add ">".`
- Caso você tenha usado o `<git add "arquivo">` e deseja desfazer esta ação (retornar para o estado unmodified) use o comando `<git reset HEAD "nome-arquivp.extensao">`.
- O comando `<git diff --name-only>` retorna somente os nomes dos arquivos que foram modificados. Facilitando a identificação deles antes de alterar seus estados.



# Logs

---

- Logs são usados para verificar os estados do projeto. O comando **<git log>** apresenta diversas informações sobre as atividades no diretório monitorado.
- As informações são apresentadas em ordem cronológica com as modificações mais recentes no início da lista, por exemplo:

# Visualização de logs

---

commit ee6b28011c7163ed959903695c07ed814f7011c3

Author: Fagner de Oliveira Bernardo <bernardo.fagner@hotmail.com>

Date: Mon Nov 27 13:45:46 2017 -0200

Segundo Commit

commit 8413d32bc078f5b9ecef5ac1b22b5bd5b8e1243

Author: Fagner de Oliveira Bernardo <bernardo.fagner@hotmail.com>

Date: Mon Nov 27 13:42:32 2017 -0200

Primeiro Commit

# Logs Específicos

---

- O comando **<git log>** aceita parâmetros. Por exemplo, é possível listar todos os commits de um autor específico, passando o nome completo ou uma sub string que contenha um padrão que possa ser encontrado no nome dele. Como por exemplo: **<git log --author "Fag">**.
- Caso o parâmetro passado não contenha nenhum registro associado, o git não retorna nada.



# Logs Resumidos

---

- O comando `<git shortlog>` exibe nome dos autores, quantidade de commits feitos e as mensagens associadas a cada commit, de modo que a visualização fique mais organizada e generalizada. Ideal para mensurar a taxa de contribuição dos contribuintes.

# Log Linha do tempo

---

- O comando `<git log --graph>` exibe os commits de uma forma que se pode ter uma ideia mais clara de como eles foram realizados ao longo do desenvolvimento do projeto, mostrando os pontos em que os arquivos sofreram modificações e onde **branches** foram criados e unificados.

# Exibindo Detalhes da Versão

---

- O comando `<git show "hash">` (sem aspas duplas) é utilizado para mostrar as modificações realizadas no arquivo. Deve ser passado como parâmetro a hash associada ao commit (use `<git log>` para encontra-la), por exemplo:

`<git show ee6b28011c7163ed959903695c07ed814f7011c3>`.



# Desfazendo Ações Nível 1

---

- O comando `<git checkout "nome-do-arquivo.extensao">` (sem aspas) permite desfazer alguma alteração em um arquivo que não foi validado pelo `<git add >`. O arquivo tem todas as modificações desfeitas até o último commit realizado. Ideal para retornar ao último estado válido do sistema caso você se perca em algum momento anterior a um commit. É necessário que seu projeto possua pelo menos 1 commit já realizado.
- O comando `<git reset HEAD "nome do arquivo.extensão">` (sem aspas) pode ser utilizado para retornar o arquivo do estado modified (antes do commit) para o estágio unmodified.

# Desfazendo Ações Nível 1

---

- Para remover um arquivo do Git, remova-o dos arquivos que estão sendo monitorados (mais precisamente, removê-lo da sua área de seleção) e então fazer o commit.
- O comando **<git rm>** faz isso e também remove o arquivo do seu diretório para você não ver ele como arquivo não monitorado (untracked file) na próxima vez.

# Desfazendo Ações Nível 1

---

- Se você apenas remover o arquivo do seu diretório, ele aparecerá em “Changes not staged for commit” (isto é, fora da sua área de seleção ou unstaged).
- Caso o arquivo já esteja na área staged (pré commit) mas um commit ainda não foi realizado, utilize o parâmetro '-f' para forçar a retirada do arquivo do controle do git, faça: **<git rm "nome.extensao" -f>**.



# Desfazendo Ações Nível 2



- O comando **<git reset --X Y>** é usado para retornar à um estado específico no seu projeto, onde **X** pode assumir três valores:  $X = \{\text{soft} \mid \text{mixed} \mid \text{hard}\}$ . **Y** é o código hash do ponto ao qual se quer voltar (use git log para encontrá-lo).
  - **soft** faz retornar para o ponto staged, em que o arquivo está pronto para receber um commit;
  - **mixed** vai retornar o arquivo para o ponto modified e;
  - **hard** vai destruir todos os outros commits até o ponto do hash que você escolheu (mas não deleta o hash que você escolheu).

# Branch

---

- O branch é um ponteiro móvel que leva um commit, ou seja, um branch aponta para um commit específico (aquele em que você estava ao criar o branch).
- É possível ter diversos branches apontando para estados diferentes dentro de um mesmo projeto. Com o branch, pode-se alterar os arquivos sem alterar o branch master, ou seja, sua contribuição é paralela à linha de desenvolvimento original.
- O branch evita conflitos, pois cada contribuidor trabalha em uma linha de desenvolvimento diferente.
- Os branches possuem duas definições:
  - locais – São aqueles que existem no repositório local (seu computador);
  - remotos – São aqueles que existem no repositório remoto (Github, Bitbucket);



# Branches locais

---

- Para criar um branch local, abra o terminal no diretório do projeto e use o comando `<git checkout -b "nome_do_branch">` (sem espaço caso o nome seja composto), onde o parâmetro 'b' indica a criação de um novo branch para aquele projeto.
- É possível listar todos os branches locais atualmente existentes usando o comando `<git branch>`. Ele também mostra, por meio de uma marcação em asterisco, o branch em que você está atualmente.



# Branches remotos

---

- Para criar um branch remoto, abra o terminal no diretório do projeto e use o comando **<git checkout -b "nome\_do\_branch">** (sem espaço caso o nome seja composto), onde o parâmetro 'b' indica a criação de um novo branch local para aquele projeto. Para fazer com que este branch exista no repositório remoto também, é preciso enviá-lo para o repositório remoto usando o comando: **<git push origin "nome\_do\_branch">**.
- É possível listar todos os branches atualmente existentes usando o comando **<git branch -r>**. Ele também mostra, por meio de uma marcação em asterísco, o branch em que você está atualmente.

# Navegando Entre Branchs

---

- Para mover-se para um outro branch dentro do mesmo projeto basta usar o comando `<git checkout "nome do branch">` (sem aspas).
- Use `<git status>` ou `<git branch>` para verificar o branch em que se está atualmente.

# Apagando Branches locais

---

- Caso queira apagar um branch local que não quer mais usar pode-se usar o comando `<git branch -d "nome-do-branch">`. Ao passar o parâmetro **'-D'** (com D maiúsculo) em vez de utilizar **'-d'**, você deleta o branch ignorando quaisquer modificações que não foram atualizadas (sem fazer o **merge**) no branch master.
- Pode-se passar o nome do branch com ou sem aspas duplas.



# Apagando Branches remotos

---

- Caso queira apagar um branch remoto que não quer mais usar pode-se usar o comando `<git push origin --delete "nome-do-branch">`.
- Pode-se passar o nome do branch com ou sem aspas duplas.

# Merge

---

- **Merge** é o procedimento para unir linhas de desenvolvimento paralelas ao branch master no branch master.
- A vantagem do merge é que ele não destrói nenhum histórico dos commits anteriores.
- Quando o merge é realizado, ele cria mais um ponto de commit, que terá todas as implementações paralelas e as implementações do branch master.

# Merge

---

- Desta forma, tem-se também o histórico de todos os commits realizados no branch principal e nos branches paralelos gerando mais de um ponto de referência (hash). A criação de diversos branches pode deixar um pouco confusa a visualização da linearidade do desenvolvimento, mas pode ajudar muito na hora de retornar a um certo ponto do projeto.



# Merge

---

- Para realizar o merge, mova-se para o branch no qual você deseja adicionar uma contribuição (realizada em outro branch) usando `<git checkout "nome-do-branch-de-destino">` (com ou sem aspas) e na sequência `<git merge "nome-do-branch-que-você-deseja-unir">`. Use `<git log>` para visualizar as mudanças.

# Rebase (A Título de Conhecimento)

---

- **Rebase** é o procedimento de unir branches de forma similar ao merge, mas neste caso, não se tem a geração de diversos pontos de referência pois o rebase move o ponteiro, do branch a ser unido ao branch master, para um ponteiro a frente do branch master e deleta o ponteiro paralelo, desta forma alguns dados históricos são perdidos e pode ficar difícil retornar a algum ponto anterior do projeto que não esteja no branch master.
- Prefira o Merge em vez do Rebase!

# Visualizando Fluxo de Merges

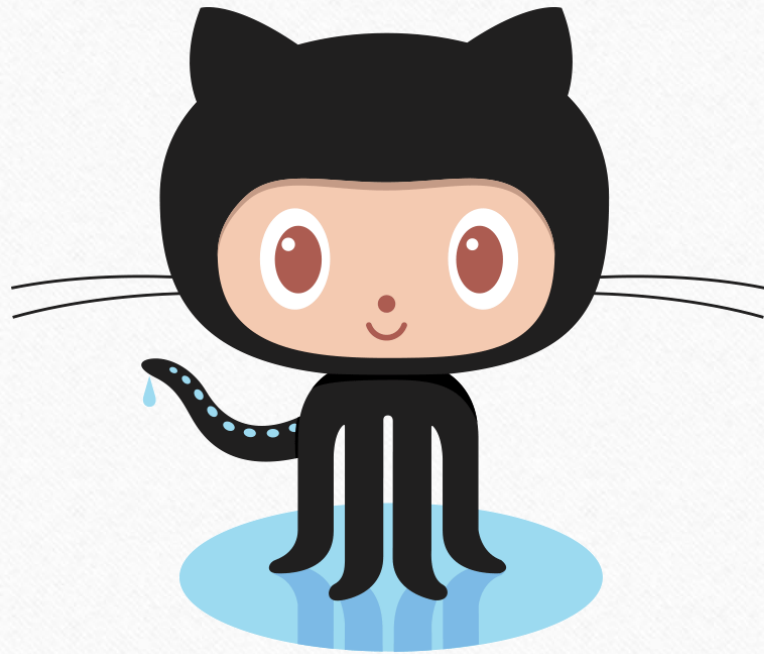
---

- O comando `<git log --graph>` pode ser usado para auxiliar na visualização dos merges que foram realizados ao longo do período de desenvolvimento.



# Repositórios Remotos

---



# GitHub

# O GitHub

---

- O GitHub é um serviço de web compartilhado para projetos que utiliza o Git para versionamento. O GitHub é um site onde você coloca seus arquivos que serão controlados pelo Git.
- O GitHub é open source, portanto seus arquivos lá armazenados são de livre acesso a todos os usuários do site. Mas você pode possuir locais privados onde seu material não é visível a todos os usuários do sistema.

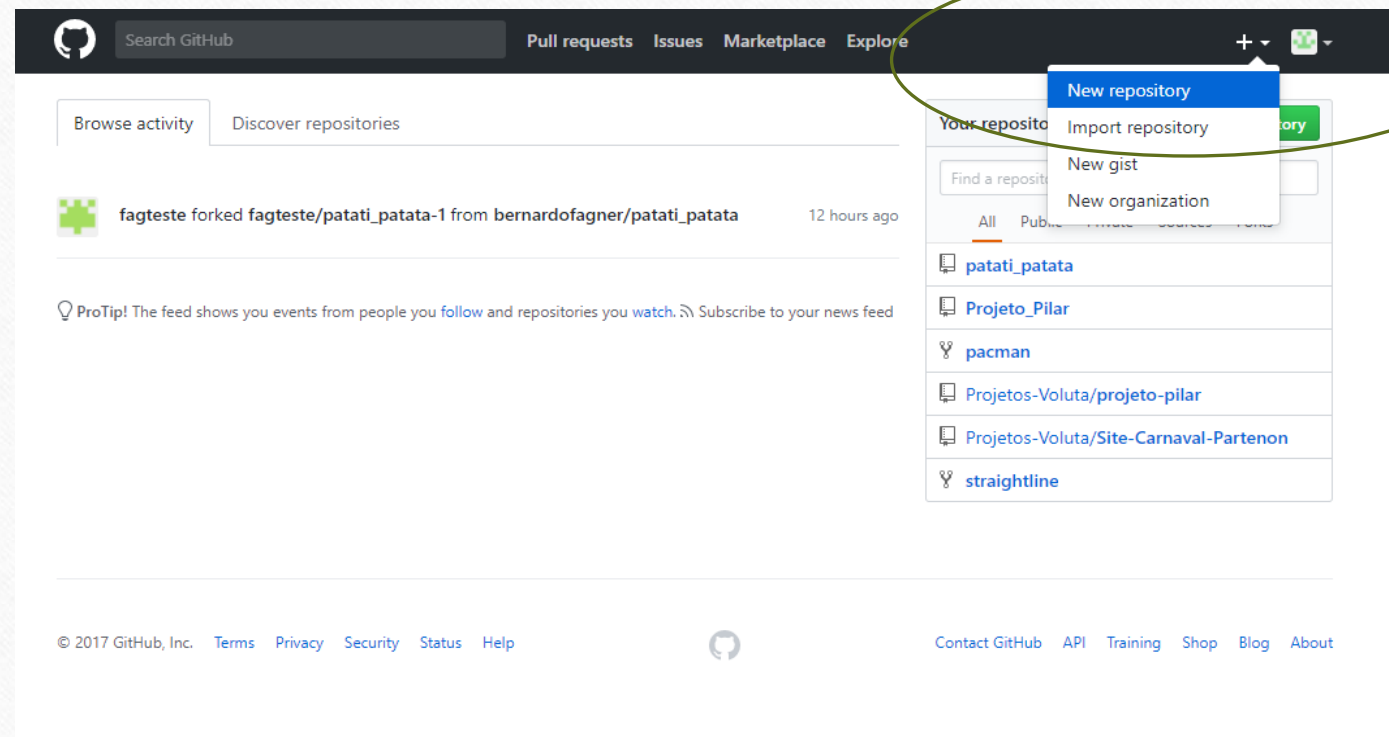
# Repositórios Remotos

---

- Repositório remoto é aquele que fica em outro servidor (servidor remoto), o mais popular é o GitHub, que é conhecido como a maior rede social de código open source do mundo e pode ser acessado pelo link: <https://github.com>.
- Neste momento deve-se criar uma conta no GitHub. O 'username' fornecido na criação da conta no GitHub será o nome utilizado na hora de enviar os arquivos do repositório local (no computador) para o repositório remoto (no github), anote-o e não o perca!



# Criando um Repositório Remoto 1




# Criando um Repositório Remoto 2

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 bernardofagner ▾


Repository name

patati\_patata ✓


Great repository names are short and memorable. Need inspiration? How about scaling-giggle.

Description (optional)

Porque palhaço é aquele que não se diverte ^^

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾ ⓘ

Create repository

# Ligando um repositório Local ao Remoto

---

- Pode-se vincular um repositório remoto já existente a um repositório local controlado pelo git (na sua máquina) de forma simples.
- O comando a seguir vincula o repositório remoto ao seu repositório local:
- `<git remote add origin https://github.com/"nome de usuario github"/"nome do repositorio remoto".git>`
- Este tipo de vínculo é realizado via protocolo **https**.



# Informações Sobre o Repositório Remoto

---

- O comando **<git remote>** retorna o nome do repositório remoto. Por padrão, o nome é 'origin', mas ele pode ser alterado no momento que o repositório remoto for ligado ao repositório local.
- O comando **<git remote -v>** traz maiores informações sobre o diretório remoto.

# Baixando um Projeto do Repositório Remoto

---

- Após realizar a ligação do repositório remoto a um repositório local, deve-se fazer o “download” dos arquivos do diretório remoto para o seu repositório local, use o comando a seguir para realizar esta ação.
- **<git pull origin master>**
- Se o git solicitar uma autenticação, forneça as informações e continue a tarefa.
- Este comando também pode ser usado para atualizar o seu projeto local caso haja alguma nova informação adicionada no repositório remoto.



# Enviando a Versão Local Para o GitHub

---

- O comando **<git push -u origin master>** envia todos os arquivos do repositório local para o repositório remoto lá no GitHub.
- O parâmetro 'origin' indica para onde vão os arquivos do diretório local, o parâmetro 'master' indica o branch em que os arquivos que chegarão no repositório remoto estão (branch principal) , o parâmetro '-u' serve para que o github aprenda sobre este repositório local e para que da próxima vez seja possível usar apenas o comando **<git push>**.
- Sempre que um push é realizado o repositório remoto é atualizado com a versão do usuário que está realizando o push.



# Clonando um Projeto do Repositório Remoto

---

- O comando `<git clone "URL">` lhe permite clonar um repositório para que você possa fazer contribuições em projetos do repositório que você está clonando. Para obter a URL do projeto que se quer clonar, acesse o GitHub, encontre o nome do usuário alvo (dono do projeto), encontre o nome do projeto (repositório remoto) e acesse-o.
- Ao acessar o repositório remoto, haverá uma opção chamada “Clone ou Download”. Clique nesta opção e copie o link gerado.

# Clonando um Projeto do Repositório Remoto 1

The screenshot shows the GitHub interface for the repository 'bernardofagner / patati\_patata'. At the top, there are buttons for 'Watch', 'Star', and 'Fork', each with a count of 0. Below these are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The repository description is 'Porque palhaço é aquele que não se diverte ^^' with an 'Edit' button. Below the description, there are statistics: '2 commits', '1 branch', '0 releases', and '1 contributor'. A row of buttons includes 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button which is circled in green. At the bottom, a commit history table is visible.

Commit	Author	Message	Time
d9bbe41	bernardofagner	Update README.md	41 minutes ago

# Fork 1

---

- O Fork cria uma duplicata de um projeto que está no Github e cria uma linha de desenvolvimento paralela para você, útil para copiar um repositório em que você deseja contribuir com seu conhecimento.
- Primeiro deve-se fazer um fork do projeto e assim que todas as suas contribuições forem realizadas, você envia um **pull request** ao dono do projeto, que avaliará suas contribuições e as adicionará ao projeto original caso ele ache válido.
- Encontre o nome do usuário do dono do projeto no GitHub e encontre o projeto que você deseja contribuir. Selecione-o e faça um 'fork' e depois um 'clone' dele para o repositório local escolhido.



# Fork 2

The screenshot shows the GitHub interface for the repository 'bernardofagner / patati\_patata'. The repository name is circled in green. In the top right, the 'Fork' button and its count (7) are also circled in green. Below the repository name, there are tabs for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. The main content area displays the text 'Porque palhaço é aquele que não se diverte ^^' with an 'Edit' button. Below this, statistics show 22 commits, 1 branch, 0 releases, and 6 contributors. At the bottom, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. A recent activity bar shows a merge pull request #6 from JuliaAzt/master by bernardofagner, with the latest commit 6b159cb made 2 days ago.

GitHub navigation bar: This repository Search Pull requests Issues Marketplace Explore

Repository: **bernardofagner / patati\_patata** Watch 0 Star 0 Fork 7

Repository tabs: Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Repository description: Porque palhaço é aquele que não se diverte ^^ Edit

Repository statistics: 22 commits 1 branch 0 releases 6 contributors

Repository actions: Branch: master New pull request Create new file Upload files Find file Clone or download

Repository activity: bernardofagner Merge pull request #6 from JuliaAzt/master Latest commit 6b159cb 2 days ago

# Atualizando o Fork 1

---

- Uma vez que você tiver um fork do projeto de outra pessoa, pode acontecer da versão original do projeto ter recebido atualizações. É possível obter as atualizações sem ter que realizar um novo fork no projeto.
- Os comandos a seguir podem ser utilizados para atualizar o seu repositório local com as novas informações contidas no projeto original, URL indica o link para o projeto original.
- **<git remote add upstream URL >**
- **<git pull upstream master>**

# Atualizando o Fork 2

---

- Após realizar a atualização, é ideal adicionar as atualizações baixadas ao seu repositório remoto utilizando o comando **<git push >** garantindo que sua contribuição esteja atualizada tanto localmente, quanto remotamente.



# Enviar Pasta de Projeto Para o Repositório Remoto (Revisão) 1

---

- 1 - Crie um novo repositório no GitHub (idêntico ao nome do projeto de preferencia)
- 2 - Preencha o campo descrição com as informações relevantes, marque a opção "public" logo abaixo caso sua licença seja gratuita. Marque a opção "Initialize this repository with a README file", clique em "CREATE REPOSITORY".
- 3 - Diretório criado, use o comando `<git clone ...>` ou `<git remote add ...>` para cloná-lo/liga-lo no diretório de seu computador, sincronize o repositório remoto ao local usando `<git pull ...>`.

# Enviar Pasta de Projeto Para o Repositório Remoto (Revisão) 2

---

4 - Entre na pasta criada no seu computador e cole todos os arquivos deste projeto, fazendo desta pasta, a pasta raiz do seu projeto. Pode excluir o arquivo README.md se quiser usando **<git rm ...>**.

5 - Utilize os comandos vistos anteriormente para adicionar os arquivos ao controlador local do git (para adicionar o conteúdo de uma pasta digite: **<git add pasta/>**, onde 'pasta' é o nome da pasta a ser enviada e '/' indica que é para adicionar todos os arquivos dentro da pasta também). Depois realize os commits necessários para que finalmente seja possível fazer o "push" para o repositório remoto no GitHub.

6 – Você também pode usar **<git add .>** (git add 'ponto final') para enviar todos os arquivos do diretório, similar a **<git add -A>**.



# Material Complementar

---

- Git and GitHub for Poets;
  - <https://www.youtube.com/watch?v=BCQHnlnPusY&t=5s>
- Aprenda Git do Básico ao Avançado:
  - <http://comandosgit.github.io/>



# Dúvidas?

---

