[Udemy] Git e contribuições para projetos Open Source

***Objetivo:*** *aprender Git e sua utilidade na prática!*

**Seção: 1 Introdução ao controle de versão com Git**

2. Gerência de Configuração de Software

* O que é Git-SCM
* Gerência de Configuração de Software

3. Controle de Versão

* Ferramentas de Controle de Versão

4. GitHub e alternativas

* O que é GitHub
* Servidores

5. Ferramentas Git

* Instalar
* Criar conta no Github

6. Observações

* Instalar o java
* Terminal Bash
* Terminal Fish (Friendly Interative Shell)

**Seção: 2 Básico em Git**

6. Chave SSH

7. Primeiros comandos Git

* Ajuda (<https://git-scm.com/doc>)
* Criar Repositório
* Criar Arquivo
* Configurar
* Status do repositório
* Últimos commits no repositório

8. Criar repositório remoto

* Criar um repositório no Github
* Configurar o repositório remoto
* Fazer o upload das alterações

9. Editar o README.md

* Criar o arquivo README.md

10. Revisão dos comandos

* Git Workflow
* Git Add
* Git Commit
* Git Push

11. Estado dos Arquivos no Git

* Diff

**Seção: 3 Histórico e Conflitos**

13. Comandos Clone e Pull

* Git Clone
* Git Pull

14. Colaboração em repositórios

* Configurar colaborador no GitHub

15. Navegar no histórico com git checkout

16. Desfazer alterações

* Git Checkout
* Git Revert
* Git Reset

18. Conflitos

20. Resolvendo conflitos com Merge

21. Visualizando o histórico em uma Interface Gráfica

* GitHub
* GitEye

**Seção: 4 Branching, Merge e Rebase**

22. Branching

* Branch

24. Git Merge

25. Git Rebase

27. Git Fetch

28. Tags

**Seção: 5 Colaboração com Open Source**

30. Interfaces Gráficas

31. Dando estrelas e seguindo no GitHub

32. Fork e Issues

* Fork no GitHub
* Issues no GitHub

33. Pull Request e Workflow do GitHub

* Pull Request
* Fluxo do GitHub

35. Verificar e aceitar Pull Request

* Checkout em Pull Request

36. Caso de Exemplo em Open Source

**Seção: 6 Além do Básico**

37. Git Ignore

38. Git Commit Amend

39. Git Stash

40. Git Cherry-pick e Git Blame

* Git Cherry-pick
* Git Blame

41. Git Bisect

42. GitHub Pages

43. GitHub Milestones

44. WebHooks

* Hooks e Serviços

45. GitKraken GUI

**Seção: Exercícios**

* Exercício de commit
* Exercício de colaboração em repositórios
* Exercício de conflitos
* Exercício de branchs
* Exercício de merge
* Exercício de rebase
* Exercício de fetch
* Exercício de fork e pull request
* Git Game

**Seção: 1 Introdução ao controle de versão com Git**

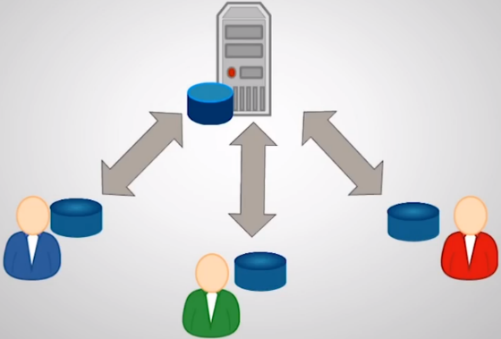
**2. Gerência de Configuração de Software**

* **O que é Git-SCM**
* Sistema de Controle de Versão Distribuído
* SCM -> Source Control Management
* Criado por Linus Torvalds (2005)
* Auxiliar no Desenvolvimento do Linux
* **Gerência de Configuração de Software**
* Durante o desenvolvimento do software, deve-se saber:
* O que mudou e quando?
* Por que mudou?
* Quem fez a mudança?
* Pode reproduzir esta mudança?
* Quatro (4) Tópicos

|  |  |
| --- | --- |
| 1. Identificação | Identificar os componentes do software |
| 2. Documentação | Documentar internamente e documentar o desenvolvimento |
| 3. Controle | Verificar e testar o que está sendo desenvolvido |
| 4. Auditoria | Permitir auditar o que está sendo desenvolvido, verificar se está aliado as expectativas do projeto |

* Artefatos (passível de controle de versão)
* Código fonte
* Documentação do Software
* Manual de Usuário

**3. Controle de Versão**

* Controle de versão faz o 'merge'/unificação (mantendo a coerência) entre as alterações
* **Ferramentas de Controle de Versão**
* Subversion (SVN)
* O SVN não é distribuído, portanto é necessário a conexão com o servidor para realizar o controle de versão
* Mercurial (HG)
* CVS (Concurrent Versioning System)
* Bazaar
* Git (é distribuído, mais rápido e eficiente)
* Cada usuário tem uma cópia do repositório do servidor remoto, desta forma, não é necessário a conexão com o servidor para realizar o controle de versão. Isto é feito localmente.

**4. GitHub e alternativas**

* **O que é GitHub**
* Servidor de repositórios Git
* Surgiu em 2008
* +10 milhões de repositórios
* +10 milhões de usuários
* **Servidores**

|  |  |  |  |
| --- | --- | --- | --- |
| Servidores/Suporte | Git | Mercurial | SVN |
| GitHub | X |  |  |
| GitLab | X |  |  |
| Bitbucket | X | X |  |
| SourceForge | X | X | X |
| ~~Google Code~~ | X | X | X |

* GitLab (repositórios privados gratuitos sem limites)
* Bitbucket (repositórios privados com limite de 5 usuários e sem chave SSH - necessário se autenticar a cada uso)
* Google Code (encerrado)

**5. Ferramentas Git**

* **Instalar**
* Instalar o Git
* Linux/Debian

**$ sudo apt-get install git**

* Windows: <https://git-scm.com/book/pt-br/v1/Primeiros-passos-Instalando-Git>, <https://woliveiras.com.br/posts/instalando-o-git-windows/> e <https://gitforwindows.org/>
* Escolha sua interface gráfica (<https://git-scm.com/downloads/guis>)
* Recomendado: GitEye (<https://www.collab.net/downloads/giteye>)
* **Criar conta no Github**
* Acessar <https://github.com/> e criar sua conta

**6. Observações**

* **Instalar o java**

**$ sudo apt install openjdk-8-jre-headless**

**$ java -version**

* **Terminal Bash**
* Mostrar a branch no terminal bash
* <https://gist.github.com/ankurk91/2efe14650d54d7d09528cea3ed432f6d>
* Adicionar as linhas abaixo no arquivo ~/.bashrc

*# Show git branch name*

**force\_color\_prompt=yes**

**color\_prompt=yes**

**parse\_git\_branch() {**

**git branch 2> /dev/null | sed -e '/^[^\*]/d' -e 's/\* \(.\*\)/(\1)/'**

**}**

**if [ "$color\_prompt" = yes ]; then**

**PS1='${debian\_chroot:+($debian\_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[01;31m\]$(parse\_git\_branch)\[\033[00m\]\$ '**

**else**

**PS1='${debian\_chroot:+($debian\_chroot)}\u@\h:\w$(parse\_git\_branch)\$ '**

**fi**

**unset color\_prompt force\_color\_prompt**

* **Terminal Fish (Friendly Interative Shell)**
* <https://www.ostechnix.com/install-fish-friendly-interactive-shell-linux/>

**$ sudo apt-get update**

**$ sudo apt-get install fish**

* Mudar para o fish do seu shell padrão (bash)

**$ fish**

*$ Welcome to fish, the friendly interactive shell*

* Retornar ao shell padrão (bash)

**$ bash**

**Seção: 2 Básico em Git**

**6. Chave SSH**

* É necessário uma chave SSH para não redigitar a senha a cada transação com o repositório remoto (GitHub)
* <https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>
* <https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/>
* <https://help.github.com/articles/testing-your-ssh-connection/>

**7. Primeiros comandos Git**

* **Ajuda (**[**https://git-scm.com/doc**](https://git-scm.com/doc)**)**

**$ git help <command>**

* **Criar Repositório**
* Será criado o diretório configurado como um repositório Git
* Todas as configurações do repositório ficam na pasta .git

**$ git init <repositorio>**

* **Criar Arquivo**
* Criar um arquivo, editar e adicionar ao repositório

**$ gedit <arquivo>**

**$ git add <arquivo>**

**$ git commit -m "Criado o primeiro arquivo."**

*# ou git commit e inserir os comentários posteriormente*

* **Configurar**
* Configurar seu nome e e-mail (definir a identidade padrão da sua conta, a mesma do GitHub)

**$ git config --global user.name "<username>"**

**$ git config --global user.email "<email@email.com>"**

Omitir --global para definir a identidade apenas neste repositório.

* **Status do repositório**

**$ git status**

* **Últimos commits no repositório**
* Exibir os commits do mais recente ao mais antigo

**$ git log**

**8. Criar repositório remoto**

* **Criar um repositório no Github**
* Acessar o GitHub
* Clicar em "New repository"
* Fornecer um nome e uma descrição para o repositório, escolher a opção "Public" e deixar desmarcado (ou por default) as demais opções
* Clicar em "Create repository"
* **Configurar o repositório remoto**

**$ git remote add <remote> <url>**

* origin é o nome padrão para o principal remote

**$ git remote add origin git@github.com:wjuniori/StarWarsRepo.git**

* git remote é o comando para listar os remotes de um repositório

**$ git remote -v**

Neste remote, é possível pode fazer o fetch (baixar do repositório remoto) e o push (enviar para o repositório remoto).

* **Fazer o upload das alterações**
* Somente na primeira vez (criar a branch master no remote origin)

**$ git push -u origin master**

* Nas próximas vezes

**$ git push**

**9. Editar o README.md**

* **Criar o arquivo README.md**

O arquivo README.md exibe informações na página inicial do repositório. Possui tags específicas para estilização do conteúdo (sintaxe Markdown), como:

|  |  |
| --- | --- |
| # | títulos (semelhante à tag h1 do HTML) |
| ![TIE Fighter](tiefighter.png) | adicionar imagem |

**$ gedit README.md**

* Após edição do arquivo README.md

**$ git add .** *# adicionar todos os arquivos*

*# que estiver no diretório ao commit*

**$ git commit -m "Adicionado o README.md"**

**$ git push** *# sincronizar com o repositório remoto*

**10. Revisão dos comandos**

* **Git Workflow**
* Basicamente, a maior parte do trabalho com o git consiste nestas tarefas:
* Editar
* Adicionar e Commitar
* Sincronizar com o repositório remoto
* Exercite estes comandos!
* git status
* git log
* git add
* git commit
* git push
* **Git Add**
* Adicionar os arquivos novos e modificados ao próximo commit

**$ git add <lista de arquivos>**

* Adicionar todo os arquivos que estiver no repositório ao commit

**$ git add .**

* **Git Commit**
* Registrar o commit com todos os arquivos adicionados com o "git add"
* Se o parâmetro de mensagem não for passado, abrirá um editor de texto para escrever a mensagem.

**$ git commit [-m "Message"]**

* Para trocar o editor de texto de escrita da mensagem

**$ git config --global core.editor gedit**

* **Git Push**
* Enviar alterações (commits) de uma branch para o repositório remoto
* O envio é rejeitado se o repositório local não estiver sincronizado (ou seja, há commits no repositório remoto não baixados para o repositório local)
* Somente na primeira vez (criar a branch master no remote origin)

**$ git push -u origin master**

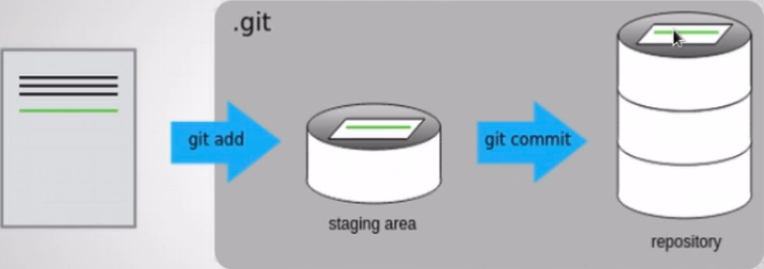
* Nas próximas vezes

**$ git push <remote> <branch>**

**$ git push**

**11. Estado dos Arquivos no Git**

|  |  |
| --- | --- |
| Não monitorado (untracked) | Arquivo novo (recém-criado) |
| Modificado (modified) | Arquivo legado do repositório, após uma modificação de conteúdo |
| Preparado (staged) | Após o git add, antes do git commit |
| Consolidado (commited) | Após o git commit |



* **Diff**
* Exibir diferenças entre commits e branchs

**$ git diff**

**$ git diff [path]**

* Diferença no diretório (mostrar o que foi alterado nos últimos commits)

**$ git diff <commitBefore> <commitAfter>**

**$ git diff HEAD~1** *# comparar o conteúdo do repositório local*

*# com o penúltimo commit realizado*

|  |  |
| --- | --- |
| HEAD | Último commit realizado na branch |
| HEAD~n | n commit anterior ao atual/último realizado (n de 1 até ...) |

* Ver diff no GitHub
* Selecionar um commit já realizado e verificar as diferenças (semelhante ao comando git diff)

**Seção: 3 Histórico e Conflitos**

**13. Comandos Clone e Pull**

* **Git Clone**
* Baixar o repositório remoto
* Outra forma de criar um repositório local
* Já vem com o remote configurado

**$ git clone <URL>**

* URL no GitHub
* No repositório GitHub, clicar em "Clone or download"
* Copiar a URL "SSH" do repositório
* Executar o comando no terminal

**$ git clone git@github.com:wjuniori/StarWarsRepo.git**

* **Git Pull**
* Baixar as alterações do repositório remoto
* Manter o repositório local sincronizado com os últimos commits de uma branch (do repositório remoto)

**$ git pull**

**14. Colaboração em repositórios**

* Colaboradores têm permissão de executar o comando git pull para um mesmo repositório remoto. Isto permite trabalhar conjuntamente em um mesmo código
* **Configurar colaborador no GitHub**
* No repositório GitHub, ir em "Settings" => "Collaborators"
* Realizar a busca por um colaborador
* Clicar em "Add Collaborator"

**15. Navegar no histórico com git checkout**

* Ver como um arquivo ou todo o repositório estava em um determinado commit
* Alterar o repositório para o estado daquele commit
* Útil para fazer testes antes e depois de alterações

**$ git checkout <commit> <file>**

* Para retornar o repositório ao último commit da branch

**$ git checkout <branch>**

* Exemplo:

**$ git log**

*# Copiar o código do commit desejado (pode ser*

*# somente uma parte que identifica unicamente o commit)*

**$ git checkout 304287013**

*# O repositório local fica no estado 'detached HEAD' para indicar*

*# que não trata-se do commit mais atual, mas do commit 304287013*

**$ git checkout master**

**16. Desfazer alterações**

* **Git Checkout**
* Desfazer as alterações que não estejam no Stage desde o último commit (descartar modificações nos arquivos ainda não adicionados - antes da execução do git add)

**$ git checkout -- <path\_or\_file>**

**$ git checkout -- .** *# desfazer as alterações em todos os arquivos*

* Exemplo:

**$ gedit <arquivo>**

*# não foi realizado o git add*

**$ git checkout -- <arquivo>**

* Desfazer as alterações desde o último commit, incluindo o Stage (descartar as modificações nos arquivos já adicionados - após execução do git add e antes da execução do git commit)

**$ git checkout HEAD -- <path\_file>**

* Exemplo:

**$ gedit <arquivo>**

**$ git add <arquivo>**

*# não foi realizado o git commit*

**$ git checkout HEAD -- <arquivo>**

* **Git Revert**
* Irá criar um novo commit que desfaz as alterações do commit especificado (após execução do git commit)
* Útil para desfazer um commit antigo

**$ git revert <commit>**

* Exemplo:

**$ gedit <arquivo>**

**$ git add .**

**$ git commit** *# commit b61cf4f56def06e93d6474a708ceeeb35a79a38d*

**$ git revert b61cf4f5**

*# É adicionado automaticamente o comentário inicado com 'Revert "..."*

*# This reverts commit b61cf4f56def06e93d6474a708ceeeb35a79a38d.'*

**$ git diff b61cf4f5 e9895871**

* **Git Reset**
* Resetar o repositório para um determinado commit (ou seja, permite refazer o commit, não perde as alterações realizadas, basta realizar o commit de novo) – antes da execução do git push

**$ git reset <commit>**

* Resetar e remover todas as alterações
* Cuidado ao usar! Não usar se já estiver publicado no GitHub (após o git push), pois descarta um determinado commit (que já pode ter sido baixado por outro colaborador)
* Útil para desfazer últimos commits

**$ git reset <commit> --hard**

* Exemplo:

**$ gedit planetas**

**$ git add .**

**$ git commit**

*# não foi realizado o git push*

**$ git reset HEAD~1**

*# retornar ao exato estado de um determinado commit, ou seja, permite*

*# refazer esse mesmo commit, no caso, por exemplo, de ter esquecido de*

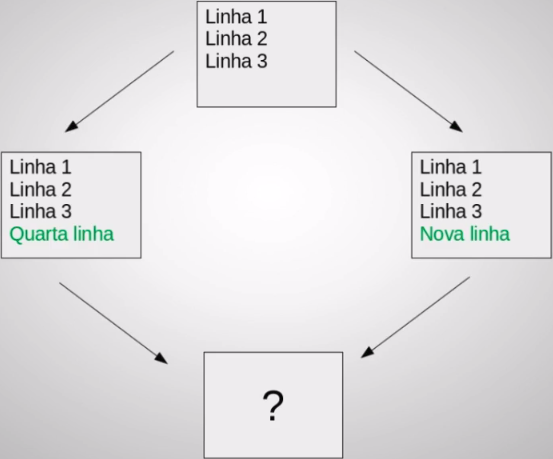
*# incluir algum arquivo. O repositório permanece o mesmo (já com as*

*# alterações), mas o commit foi desfeito*

**$ git diff**

**$ git reset HEAD~1 –hard** *# apagar o commit e voltar pra estado anterior*

*# (descarta as modificações do commit)*

**18. Conflitos**

* Conflitos podem acontecer ao unirmos alterações
* Acontecem quando versões diferentes possuem as mesmas linhas nos mesmos arquivos editadas diferentes
* O git identifica os conflitos e fica aguardando a solução deles
* Ao resolver os conflitos, deve ser feito um commit

*# Colaborador A*

**$ gedit <arquivo>**

**$ git add .**

**$ git commit**

**$ git push**

*# Colaborador B*

**$ gedit <arquivo>**

**$ git add .**

**$ git commit**

**$ git push**

*# [ ! [rejected] master -> master (fetch first)]*

**$ git pull**

*# [Mesclagem automática de* <arquivo>

*# CONFLITO (conteúdo): conflito de mesclagem em* <arquivo>

*# Automatic merge failed; fix conflicts and then commit the result]*

**$ git status**

*# [ambos modificados:* <arquivo>*]*

**20. Resolvendo conflitos com Merge**

*# Colaborador B*

**$ gedit <arquivo>**

<<<<<<< HEAD

Dantooine

=======

Jakku

>>>>>>> c4edb4a4bec3c492310a5e006a890bd3db27ee17

*# Realizar as modificações necessárias (resolver o conflito)*

*# e salvar o arquivo*

**$ git add .**

**$ git status**

**$ git commit**

*# [comentário automático]*

*# Merge branch 'master' of github.com:wjuniori/StarWarsRepo*

**$ git push**

*# Colaborador A*

**$ git pull**

**21. Visualizando o histórico em uma Interface Gráfica**

* **GitHub**
* No seu repositório, em "Insights" => "Network", ver a linha do tempo do desenvolvimento (com os conflitos, inclusive)
* **GitEye**
* Adicionar um repositório
* Clicar em "Add an existing local Git repository"
* Na próxima janela, clicar em "Browse..."
* Escolher o repositório local
* Clicar em "Finish"
* Após clicar no repositório, na janela "Create Project for <repository>", deixar marcado a única opção
* Na aba "History"
* Clicar no ícone "Show All Branches and Tags"
* Verificar todo o histórico de commits
* Na aba "Git Files", é possível realizar o git add, git commit e git push

**Seção: 4 Branching, Merge e Rebase**

**22. Branching**

* Criando ramificações do repositório
* **Branch**
* É uma lista de commits
* Representa ramificações no repositório
* Muito útil para trabalhos colaborativos
* Branchs de desenvolvimento facilitam o controle
* Branch master é a padrão
* Listar todas as branchs (com \* é a branch atual)

**$ git branch** *# git status também informa a branch atual*

* Criar uma nova branch

**$ git branch <nova\_branch>**

Ao criar uma nova branch, isto ainda não foi refletido no repositório remoto. Desta forma, para fazer o primeiro upload, deve-se usar:

**$ git push -u origin <nova\_branch>**

Nas próximas vezes:

**$ git push**

* Excluir uma branch

**$ git branch -d <branch>**

* Mudar para uma branch

Seu repositório passa a ter os commits que a branch possui e novos commits serão adicionados à ela.

**$ git checkout <branch>**

* Criar uma branch e já mudar para a branch criada

**$ git checkout -b <branch>**

**24. Git Merge**

* Aplicar os commits de uma branch na branch atual
* Encontrar um commit comum (base) entre as branchs e aplica todos os commits que a branch atual não possui
* Caso existam commits na branch atual que não estão na outra, será criado um commit de merge

**$ git merge <branch>**

* Caso ocorra conflito, é necessário abrir o arquivo, resolver o conflito e realizar o commit do resultado

(master) **$ git merge novaBranch**

*# Mesclagem automática de personagens*

*# CONFLITO (conteúdo): conflito de mesclagem em personagens*

*# Automatic merge failed; fix conflicts and then commit the result*

(master) **$ git status**

*# ambos modificados: personagens*

(master) **$ gedit personagens**

<<<<<<< HEAD

Jabba the Hutt

=======

Chewbacca

Han Solo

>>>>>>> novaBranch

*# Realizar as modificações necessárias (resolver o conflito)*

*# e salvar o arquivo*

(master) **$ git add .**

(master) **$ git commit**

*# Merge branch 'novaBranch'*

(master) **$ git push**

**25. Git Rebase**

* Semelhante ao merge, porém é diferente na ordem de aplicar os commits
* No rebase, os seus commits na frente da base são removidos temporariamente, os commits de outra branch são aplicados na sua branch e, por fim, seus commits são aplicados um a um
* Pode acontecer conflitos que serão resolvidos para cada commit

**$ git rebase <branch>**

* É preferível usar o git merge quando há a previsão de muitos conflitos, ou seja, os branchs estão bastante divergentes. Caso não haja muitos conflitos (não há muita divergência entre as branchs), preferir o git rebase

(master) **$ git checkout -b branch2**

*# Switched to a new branch 'branch2'*

(branch2) **$ git checkout master**

(master) **$ gedit personagens**

(master) **$ git add .**

(master) **$ git commit**

(master) **$ git log**

(master) **$ git checkout branch2**

(branch2) **$ gedit personagens**

(branch2) **$ git add .**

(branch2) **$ git commit**

(branch2) **$ git rebase master**

*# Mesclagem automática de personagens*

*# CONFLITO (conteúdo): conflito de mesclagem em personagens*

*# error: Failed to merge in the changes.*

(branch2) **$ git status**

*# rebase in progress; onto 3c6feba*

*# ambos modificados: personagens*

(branch2) **$ gedit personagens**

<<<<<<< HEAD

Rey

Finn

=======

Kylo Ren

>>>>>>> Adicionado Kylo Ren.

*# Realizar as modificações necessárias (resolver o conflito)*

*# e salvar o arquivo*

(branch2) **$ git status**

*# rebase in progress; onto 3c6feba*

*# ambos modificados: personagens*

(branch2) **$ git add .**

(branch2) **$ git rebase –continue**

(branch2) **$ git diff HEAD~1 HEAD**

**27. Git Fetch**

* Baixar as atualizações do remote (repositório remoto), porém não aplicá-las no repositório
* Permitir fazer o rebase de uma branch em vez de fazer o merge
* git pull = git fetch + git merge
* Fetch e rebase é melhor para manter histórico do desenvolvimento

**$ git fetch**

* Exemplo:

*# StarWarsRepo-PRIMEIRO(master)*

**$ gedit personagens**

**$ git add .**

**$ git commit**

**$ git push**

*# StarWarsRepo-SEGUNDO(master)*

**$ gedit personagens** *# Adicionado personagem Poe Dameron*

**$ git diff**

**$ git add .**

**$ git commit**

**$ git fetch**

**$ git log**

*# NÃO ATUALIZOU O REPOSITÓRIO LOCAL COM AS MODIFICAÇÕES DO REPOSITÓRIO REMOTO*

*# commit 38576c81acdb6447b4039af4fe4f0b3d65072c22 (HEAD -> master)*

*# Author: wjuniori <wjuniori.si@gmail.com>*

*# Date: Fri May 11 17:47:12 2018 -0300*

*#*

*# Adiciona Poe Dameron.*

**$ git rebase** *# Fazer o rebase da base com essa alterações baixadas*

*# Mesclagem automática de personagens*

*# CONFLITO (conteúdo): conflito de mesclagem em personagens*

*# error: Failed to merge in the changes.*

**$ gedit personagens**

<<<<<<< 00d33fc4ff7d9434dd86ddf3b638a8284b47a3505

Chewbacca

Han Solo

Jabba the Hutt

Rey

Finn

Kylo Ren

=======

Poe Dameron

>>>>>>> Adicionado Poe Dameron.

*# Realizar as modificações necessárias (resolver o conflito)*

*# e salvar o arquivo*

**$ git status**

*# rebase in progress; onto 00d33fc*

*# ambos modificados: personagens*

**$ git add .**

**$ git rebase –continue** *# Applying: Adiciona Poe Dameron.*

**$ git log** *# manter todo o histórico do desenvolvimento*

**$ git push**

**28. Tags**

* Útil para definir versões estáveis do projeto
* Semelhante a Branch, porém não recebe mais commits
* Guarda um estado do repositório
* Criar tag

**$ git tag <nomeTag>**

* Mostrar a versão da tag

**$ git tag**

* Enviar tag para GitHub

**$ git push <remote> <tag>**

* Mudar para uma tag

**$ git checkout <tag>**

Como não é possível realizar commits na tag, pode-se criar uma branch, a partir da tag:

**$ git checkout -b <new-branch-name>**

* Exemplo:

**$ git pull**

**$ git log**

**$ git tag v1.0**

**$ git tag** *# v1.0*

**$ git push origin v1.0**

**$ git checkout v1.0**

*# O repositório local fica no estado 'detached HEAD' para indicar*

*# que não se trata do commit mais atual, mas a tag v1.0*

**Seção: 5 Colaboração com Open Source**

**30. Interfaces Gráficas**

* Pelo terminal se faz tudo
* Interfaces gráficas ajudam na visualização e auxiliam em algumas tarefas
* Recomendadas:
* Egit - Plugin para Eclipse
* GitEye - Linux, Windows e Mac
* Não recomendado: GitHub for Windows

**31. Dando estrelas e seguindo no GitHub**

* GitHub e colaboração com Open Source
* GitHub é uma Rede Social
* Adicionar um repositório aos favoritos (Star)
* Seguir outros usuários (Follow)

**32. Fork e Issues**

* **Fork no GitHub**
* Copiar um repositório de outro usuário para o seu usuário no GitHub
* É assim que começa a contribuição para outros projetos, caso você não seja um colaborador
* Posteriormente, é possível pedir pra juntar suas modificações ao repositório original (pull request)
* Com o Fork, você tem uma cópia independente do repositório original, podendo fazer quaisquer modificações
* **Issues no GitHub**
* Tradução: Questões
* Reportar bugs
* Organizar tarefas a serem feitas
* Permitir discussão entre os usuários
* Pode ser referenciada por commits
* Commit: "Closes #3" -- Fechar a issue número 3 ao realizar o commit.
* Exemplo: se a issue #3 for um bug, ao corrigir esse bug e escrever "Closes #3" na mensagem de commit, ocorrerá o fechamento desta issue
* Em repositório público, qualquer usuário pode criar e comentar issues
* Para criar uma issue
* No repositório GitHub, clicar em "Issue" e depois em "New issue"
* Adicionar um "Title" e um "comment" (pode-se utilizar sintaxe Markdown)
* Clicar em "Submit new issue". A issue será criada com status "Open"
* Em "Labels", pode-se adicionar labes (exemplo: bug)
* Para fechar uma issue (com commit)
* Efetuar a correção da issue (bug)
* Realizar o git add, o git commit (com a mensagem "Closes #1") e o git push
* No GitHub, verificar o status "Closed" da issue e o link com o referido commit

**33. Pull Request e Workflow do GitHub**

* **Pull Request**
* O grande símbolo de colaboração
* É quando você solicita que suas alterações sejam unidas a uma branch no mesmo repositório ou a um repositório que sofreu o fork
* Enviar os commits para o repositório original e decidir se serão unidos (ou não), através de um merge
* Igual a uma issue, porém com uma branch (lista de commits) associada
* Muito útil para o trabalho colaborativo
* Para criar um pull request
* Com a branch aberta no GitHub, clicar em "Compare & pull request"
* Adicionar um "Title" e um "comment" (pode-se utilizar sintaxe Markdown)
* Clicar em "Create pull request"
* **Fluxo do GitHub**
* <https://guides.github.com/introduction/flow/>
* Criar uma branch
* Adicionar commits
* Abrir um Pull Request
* Discutir e revisar seu código (com a comunidade)
* Deploy (implantar as modificações para teste final)
* Merge

**35. Verificar e aceitar Pull Request**

* **Checkout em Pull Request**
* Criar uma branch com os commits do pull request

**$ git fetch origin pull/<IDPullRequest>/head:<branch>**

* Exemplo
* No repositório GitHub, aparece 1 "Pull requests" com status "Open", um "title" e um "comment" (semelhante a issue)
* Pegar o código do "Pull request" (Exemplo: #2)

(master) **$ git fetch origin pull/2/head:github\_link**

(master) **$ git checkout github\_link**

(github\_link) **$ git log**

*# rodar aplicação e realizar testes*

* No "Pull requests" do repositório GitHub, clicar em "Merge pull request", adicionar uma mensagem e "Confirm merge" (status alterado para "Merged")

(github\_link) **$ git checkout master**

(master) **$ git pull**

*# rodar aplicação e verificar as modificações*

**36. Caso de Exemplo em Open Source**

* Extensão para GNOME: Desktop Scroller (<https://github.com/BrOrlandi/Desktop-Scroller-GNOME-Extension>)
* Criar uma borda de rolagem entre os Desktops
* Configurar onde a borda é ativa
* Hoje é mantido por outros usuários

**Seção: 6 Além do Básico**

**37. Git Ignore**

* Arquivo .gitignore (na raiz do repositório)
* Configurar arquivos que devem ser ignorados (que não participam do versionamento do git)
* Contém arquivos, caminhos e patterns

**$ gedit .gitignore**

.project

node\_modules/

bower\_components/

\*\*/\*.css

**38. Git Commit Amend**

* Alterar o último commit realizado (IMPORTANTE: antes da execução do git push)
* Mensagem de commit
* Adicionar arquivos

**$ git commit --amend**

* Exemplo

**$ gedit planetas**

**$ git add .**

**$ git commit**

*# não foi realizado o git push*

**$ gedit personagens**

**$ git add .**

**$ git commit –-amend**

*# alterar a mensagem e incluir o arquivo personagens,*

*# além do arquivo planetas*

**39. Git Stash**

* Guardar as alterações do Working Directory
* Permitir, sem a necessidade de fazer um commit antes, rebase, merge e/ou trocar de branch

**$ git stash**

* Listar a pilha de stash armazenada

**$ git stash list**

* Aplica o último stash armazenado

**$ git stash pop**

* Exemplo

(novaBranch) **$ gedit planetas**

(novaBranch) **$ git checkout master**

(master) **$ gedit planetas**

(master) **$ git checkout novaBranch**

*# git checkout novaBranch error: Your local changes to the following*

*# files would be overwritten by checkout: planetas*

*# Please commit your changes or stash them before you switch branches.*

*# Aborting*

(master) **$ git stash**

(master) **$ git stash list**

(master) **$ git checkout novaBranch**

(novaBranch) **$ git checkout master**

(master) **$ git stash pop**

**40. Git Cherry-pick e Git Blame**

* **Git Cherry-pick**
* Aplicar as alterações de um commit na branch atual, criando um novo commit
* Útil para recuperar histórico ou fazer alterações em branchs bastante diferentes

**$ get cherry-pick <commit>**

**$ git cherry-pick <branch>~2** *# pegar somente o penúltimo*

*# commit de <branch>*

* Exemplo

(master) **$ git cherry-pick novaBranch**

*# pegar o último commit de novaBranch e adicionar em master*

* **Git Blame**
* Mostrar as alterações feitas por linha em um arquivo
* Mostrar o autor e o commit que foi feito aquela linha
* Útil para verificar quando as alterações foram feitas, por que e por quem

**$ git blame <arquivo>**

**41. Git Bisect**

* Fazer uma busca binária nos commits para encontrar uma alteração
* Útil para alterações que modificaram o comportamento e não podem ser identificadas por código facilmente
* Quando a alteração for bastante antiga
* Exemplo: Encontar qual o commit, no arquivo test.txt, ocorreu a substituição da palavra "boat" por "car" do repositório git\_bisect\_tests (<https://github.com/wjuniori/git_bisect_tests>)
* Solução git blame

**$ git blame test.txt**

*# ...*

*# 83ba926c (Paul Williams 2015-05-24 16:48:24 +0100 5) car*

*# ..*

**$ git show 83ba926c** *# Ver informações do commit,*

*# inclusive diff das modificações*

*# -boat*

*# +car*

* Solução git bisect

**$ git bisect start** *# ativado estado de busca (árvore de commits)*

*# no repositório*

**$ git log**

*# informar um <commit> ruim (o qual possui o*

*# comportamento incorreto ou erro)*

*# e um <commit> bom (o qual NÃO possui o*

*# comportamento incorreto ou erro).*

*# Quando for omitido <commit>, trata-se do commit atual*

**$ git bisect bad** *# com o comportamento incorreto ou erro*

**$ git bisect good 92777e8** *# com o comportamento correto ou sem erro*

*# é listado alguns commits. Deve-se indicar*

*# se os commits listados são bad ou good*

*# Bisecting: 2 revisions left to test after this (roughly 1 step)*

*# [83ba926c7b9f71a827c948b7745ffdb1869c856d] Changing the word*

*# 'boat' to 'car'*

**$ cat test.txt**

**$ git bisect bad**

*# o processo repete-se até encontrar/identificar*

*# o commit good (com o comportamento correto)*

*# Bisecting: 0 revisions left to test after this (roughly 0 steps)*

*# [7deb4fbed869540364d92de10be2c65d10d917b8] Adding the word 'gently'*

**$ car test.txt**

**$ git bisect good**

*# 83ba926c7b9f71a827c948b7745ffdb1869c856d is the first bad commit*

*# commit 83ba926c7b9f71a827c948b7745ffdb1869c856d*

**$ git bisect reset** *# sair do estado bisect*

**42. GitHub Pages**

* Hospedagem de website estático
* Site para usuários, organizações e repositórios
* Para usuário (ex.: hospedar portfólio)
* Criar um repositório <username>.github.io
* Na branch master, possuir o arquivo html
* Branch especial 'gh-pages'
* Deve conter um index.html
* Gerador de páginas do GitHub
* Criar uma nova branch gh-pages

**$ git checkout --orphan gh-pages** *# Criar branch gh-pages,*

*# sem pais (é órfão)*

* Remover todos os arquivos para criar um diretório de trabalho vazio

**$ git rm -rf .** *# Remover todos os arquivos*

*# da árvore de trabalho anterior*

**$ rm '.gitignore'**

* Adicionar contéudo e realizar o git push

**$ echo "My Page" > index.html**

**$ git add index.html**

**$ git commit -a -m "First pages commit"**

**$ git push -u origin gh-pages**

* Acessar o link https://<username>.github.io/<repository>/
* Caso optar por um tema

**$ rm index.html**

**$ git add index.html**

**$ git commit**

* No repositório GitHub, clicar em "Settings"
* Em "GitHub Pages", selecionar "gh-pages branch" em "Source"
* Clicar em "Choose a theme", escolher um tema e clicar em "Select theme"
* Editar o arquivo index.md, manter selecionado "Commit directly to the gh-pages branch." e clicar em "Commit changes"
* Acessar o link https://<username>.github.io/<repository>/
* Para editar o arquivo html ou md, acessar a branch gh-pages e realizar as modificações

**43. GitHub Milestones**

* Grupo de issues associado a um objetivo
* Possui uma data (prazo de conclusão)
* Acompanhar a % de issues resolvidas
* Existe em outros servidores Git
* Guia sobre issues: <https://guides.github.com/features/issues/>
* Instruções
* No repositório GitHub, clicar em "Issues"
* Em seguida, clicar em "Milestones" => "Create a Milestone"
* Inserir um "Title", uma "Due date" e uma "Description"
* Clicar em "Create milestone"
* Clicar em "Issues" => "New issue"
* Inserir um "Title", um "comment" e selecionar uma "Milestone"
* Clicar em "Submit new issue"
* Na milestone criada, ver todas as issues associadas
* A medida que as issues são fechadas, a milestone exibe o progresso
* Em "Issues" => "Milestones" => "Create a Milestone", clicar em "Close" para fechar uma milestone

**44. WebHooks**

* **Hooks e Serviços**
* Integrar eventos do repositório hospedado no GitHub com outras aplicações
* Alguns serviços já compatíveis
* Travis-CI, Slack
* <https://github.com/integrations>
* Instruções
* No repositório GitHub, clicar em "Settings"
* Clicar em "Webhooks" => "Add webhook" e preencher as informações
* Clicar em "Integrations & services" e escolher um serviço em "Add service"
* Em alguns repositório GitHub (ex.: <https://github.com/necolas/react-native-web>), no README, tem alguns ícones (ex: npm, build, PRs, etc)
* Ao clicar em "build", é aberto o Travis-CI, onde está hospedado os testes
* Em "Pull requests", para cada pull request é informado se os testes rodaram com sucesso ou falha. A falha não impede que ocorra o merge, mas a informação é evidenciada

**45. GitKraken GUI**

* Interface gráfica lançada recentemente (em 2016)
* <https://www.gitkraken.com/>

**Seção: Exercícios**

* **Exercício de commit**
* Remova uma linha de um arquivo
* Adicione uma nova linha no mesmo arquivo
* Adicione um novo arquivo com pelo menos uma linha
* Faça commit e o push das alterações
* Veja as diferenças do commit pelo GitHub
* **Exercício de colaboração em repositórios**
* Configure um colaborador para seu repositório
* Ele deve clonar seu repositório e fazer um commit nele
* Você deve atualizar seu repositório com o novo commit
* Invertam os papéis
* **Exercício de conflitos**
* Crie um conflito no repositório do seu par
* Seu par deve editar o mesmo arquivo que você
* Você deve fazer o pull, resolver o conflito e fazer o push das suas alterações
* Seu par deve ver que a sua alteração foi adicionada junto à alteração dele
* Invertam os papéis
* **Exercício de branchs**
* Crie uma nova branch no seu repositório
* Mude para esta branch e faça pelos menos 2 commits
* Faça upload e veja sua nova branch no GitHub
* Faça um commit na master que altere as mesmas linhas
* Veja como as branchs divergem no GitHub
* **Exercício de merge**
* Fazer o merge da nova branch na master
* Resolver o conflito e visualizar no GitHub o gráfico das branchs.
* **Exercício de rebase**
* Faça um commit na master e outro na branch
* Faça o rebase da branch com a master
* Veja a ordem dos commits
* Mesmo que tenha conflitos, o histórico de commits é preservado
* **Exercício de fetch**
* Com a branch master sincronizada, você e seu par devem fazer commits na master
* Seu par deve fazer o push dos commits dele.
* Antes de você fazer o push dos seus commits, você deve fazer o rebase com os commits do seu par
* **Exercício de fork e pull request**
* Faça um fork do repositório <https://github.com/BrOrlandi/GitTrainingWall>
* Rode o projeto
* Instruções no README.md
* Para instalar o pip (gerenciador de pacotes Python), seguir, com permissão de administrador, os passos <https://pip.pypa.io/en/stable/installing/>
* Sua tarefa é criar um template com informações do seu usuário no GitHub
* Faça o trabalho em uma branch separada e depois crie um Pull Request para o repositório original
* **Git Game**
* Série de desafios para resolver com git (<https://www.git-game.com/>)