Git - Software de Contrle de Versão VCS

GitHub - Plataforma de rede social para programadores

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Crie uma pasta nova em seu computador;

2) No terminal (ou Git Bash, no Windows) navegue até a pasta recém criada (utilize o comando cd para navegar entre pastas);

3) Execute o comando git init --bare;

4) Navegue até a pasta onde se encontra o seu projeto;

5) Execute o comando git remote add local {caminho}. Substitua {caminho} pelo caminho completo da pasta recém criada;

6) Crie uma nova pasta em seu computador, para representar o trabalho de outra pessoa;

7) No terminal (ou Git Bash, no Windows) navegue até a pasta recém criada;

8) Execute o comando git clone {caminho} projeto. Substitua {caminho} pelo caminho completo da pasta que criamos no primeiro passo;

9) Observe que o repositório clonado está vazio;

10) Execute o comando 'git remote rename origin local' para renomear o repositório local da outra pessoa de "origin" para "local";

11) Navegue até a pasta onde se encontra o seu projeto original;

12) Execute o comando git push local master para enviar as suas modificações para o seu servidor;

13) Navegue até a pasta criada no passo 6;

14) Execute o comando git pull local master para baixar as modificações;

15) Abra o seu navegador e acesse http://github.com/;

16) Crie uma conta;

17) Crie um novo repositório, clicando no símbolo de adição no canto superior direito;

18) No terminal (ou Git Bash, no Windows) adicione, ao seu projeto inicial, o repositório remoto recém criado (os comandos são mostrados pelo próprio GitHub);

19) Execute git push origin master para enviar as suas alterações para o repositório no GitHub.

Vimos como é simples resolver conflitos identificados pelo Git ao tentar realizar o merge.

Agora, gere um conflito e, ao invés de utilizar o merge, utilize o rebase para atualizar o master:

* Vá para a *branch* titulo
* *Commite* algo
* Vá para a *branch* master, *commite* uma alteração na mesma linha
* Execute git rebase titulo

Veja a saída do Git e utilize as informações que ela te der para, após corrigir o conflito, continuar o rebase.

Boa sorte! ;-)

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Execute o comando git branch e veja que apenas a *branch* master existe no seu repositório;

2) Execute o comando git branch titulo e logo após execute o comando git branch. Veja que uma nova *branch* foi criada;

3) Agora, para começar a trabalhar nesta *branch*, digite git checkout titulo;

4) Execute novamente git branch e confira que agora você está na *branch* chamado titulo;

5) Altere o título da página **index.html** para "Cursos de DevOps da Alura";

6) Adicione as alterações com git add index.html;

7) Faça o *commit*, com git commit -m "Alterando título da página";

8) Execute o comando git log e confira o novo *commit*;

9) Altere o título da página para "Lista de cursos de DevOps da Alura";

10) Repita os passos 6 e 7, para adicionar um novo *commit*, alterando a mensagem;

11) Repita o passo 8 para conferir o novo *commit*;

12) Execute o comando git checkout master para voltar à linha de desenvolvimento master;

13) Execute git log para conferir que os últimos dois *commits* não estão lá. Confira se o conteúdo do seu arquivo também voltou ao seu estado original;

14) Na pasta criada para representar o trabalho de outra pessoa na aula anterior:

* Execute git checkout -b lista para criar uma nova *branch*, chamada lista e passar a trabalhar nela;
* Adicione o curso de "Kubernetes" na lista;
* Repita os passos 6 e 7 para adicionar um novo *commit*, alterando a mensagem;
* Execute o comando git checkout master para voltar à linha de desenvolvimento master;

15) Volte para a pasta que representa o seu próprio trabalho;

16) Altere o nome do curso de Docker para "Docker: Criando containers sem dor de cabeça";

17) Repita os passos 6 e 7 para adicionar um novo *commit*, alterando a mensagem;

18) Execute o comando git merge titulo para trazer o trabalho feito na *branch* titulo para a *branch* master;

19) Execute o comando git log --graph para ver as linhas de desenvolvimento (*branches*);

20) Execute git checkout titulo para trabalhar na *branch* chamada titulo;

21) Altere o título para ter a palavra "Cursos" com letra maiúscula;

22) Repita os passos 6 e 7 para adicionar um novo *commit*, alterando a mensagem;

23) Execute o comando git checkout master para voltar à linha de desenvolvimento master;

24) Execute o comando git rebase titulo;

25) Execute o comando git log e confira que o *commit* foi adicionado antes do *commit* realizado diretamente na *branch* master;

26) Execute o comando git push local master para enviar suas alterações para o repositório remoto que criamos na última aula;

27) Na pasta criada para representar o trabalho de outra pessoa na aula anterior:

* Execute o comando git pull local master para baixar as alterações que você já realizou;
* Execute o comando git checkout lista para continuar trabalhando na lista de cursos;
* Altere o nome do curso de Docker para "Curso de Docker: Criando containers sem dor de cabeça";
* Repita os passos 6 e 7 para adicionar um novo *commit*, alterando a mensagem;
* Execute o comando git checkout master para voltar à linha de desenvolvimento master;
* Tente juntar seu trabalho com git merge lista;
* Veja que há conflitos. Corrija-os, deixando apenas a linha com o nome correto do curso;
* Execute o comando git add index.html para informar que os conflitos neste arquivo foram corrigidos;
* Execute o comando git commit para que o Git finalize o *merge*;
* Execute o comando git push local master para enviar as suas alterações;

28) Volte para a pasta que representa o seu próprio trabalho;

29) Altere o nome do curso de Vagrant para "Vagrant: Gerenciando máquinas virtuais";

30) Repita os passos 6 e 7 para adicionar um novo *commit*, alterando a mensagem;

31) Tente executar o comando git push local master. Veja a falha;

32) Execute o comando git pull local master para trazer as alterações da outra pessoa;

33) Agora sim, execute o comando git push local master para enviar as alterações.

Nesta aula, aprendemos:

* Que uma *branch* (ou ramo) é uma linha de *commits* separada, e que pode ser utilizada para desenvolver funcionalidades independentes;
* Que com *branches* separados, podemos evitar que o código de uma funcionalidade interfira em outra;
* Como trazer o trabalho realizado em uma *branch* para outra *branch*, como por exemplo, o master, através do comando git merge;
* Que o git merge gera um novo *commit*, informando que houve uma mescla entre duas *branches*;
* Como trazer os *commits* de uma *branch* para outra, com o git rebase
* Que o git rebase não gera um *commit* de merge, simplificando o nosso *log*;
* Como os conflitos são apresentados pelo Git;
* Como resolver os conflitos e manter apenas as alterações desejadas com o Git.

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Na pasta que representa o seu projeto, faça uma alteração qualquer no arquivo **index.html**;

2) Execute o git status e veja que há uma alteração para adicionar;

3) Execute o comando git checkout -- index.html. Confira se sua alteração foi desfeita;

4) Novamente, faça alguma alteração no arquivo **index.html**;

5) Execute o comando git add index.html;

6) Execute o comando git reset HEAD index.html para trazer o arquivo **index.html** de volta para a HEAD do projeto (remover do stage, que é o que será enviado para o commit);

7) Repita o passo 3;

8) Faça mais uma vez alguma alteração no código;

9) Execute o comando git add index.html e o comando git commit -m "Alterando o código" para realizar um commit;

10) Execute o comando git log e copie o hash deste *commit* recém criado;

11) Rode o comando git revert {hash}, substituindo {hash} pelo *hash* que você copiou anteriormente;

12) Confira que suas alterações foram desfeitas;

13) Mude o nome do curso de Ansible para "Ansible: Infraestrutura como código";

14) Execute o comando git stash para salvar estas alterações na stash;

15) Altere o nome do curso de Kubernetes para "Kubernetes: Introdução a orquestração de containers";

16) Execute o comando git add index.html e o comando git commit -m "Alterando o nome do curso de Kubernetes" para realizar um *commit*;

17) Execute o comando git stash pop para trazer a última alteração da *stash*;

18) Execute o comando git add index.html e o comando git commit -m "Alterando o nome do curso de Ansible" para realizar um *commit*;

19) Execute o comando git push local master para enviar todas as suas alterações;

20) Execute o comando git log --oneline para ver os *commits* de forma resumida. Copie o *hash* do *commit* de merge com a *branch* lista;

21) Execute o comando git checkout {hash} substituindo {hash} pelo *hash* que você copiou;

22) Veja que diversas alterações não estão mais presentes;

23) Execute git checkout master para voltar à linha principal de desenvolvimento.

Nesta aula, aprendemos:

* Que o Git pode nos ajudar a desfazer alterações que não vamos utilizar;
* Que, para desfazer uma alteração antes de adicioná-la para commit (com git add), podemos utilizar o comando git checkout -- <arquivos>;
* Que, para desfazer uma alteração após adicioná-la para commit, antes precisamos executar o git reset HEAD <arquivos> e depois podemos desfazê-las com git checkout -- <arquivos>;
* Que, para revertermos as alterações realizadas em um commit, o comando git revert pode ser a solução;
* Que o comando git revert gera um novo commit informando que alterações foram desfeitas;
* Que, para guardar um trabalho para retomá-lo posteriormente, podemos utilizar o git stash;
* Que, para visualizar quais alterações estão na stash, podemos utilizar o comando git stash list;
* Que, com o comando git stash apply <numero>, podemos aplicar uma alteração específica da stash;
* Que o comando git stash drop <numero> remove determinado item da stash;
* Que o comando git stash pop aplica e remove a última alteração que foi adicionada na stash;
* Que o git checkout serve para deixar a cópia do código da nossa aplicação no estado que desejarmos:
  + git checkout <branch> deixa o código no estado de uma branch com o nome <branch>;
  + git checkout <hash> deixa o código no estado do *commit* com o hash <hash>.
* Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.
* 1) Execute o comando git log -p para ver, junto a cada *commit*, as alterações nele realizadas;
* 2) Execute agora o comando git log --oneline;
* 3) Execute o comando git diff {hash do commit de merge com lista}..{hash do último commit realizado};
* 4) Execute alguma (pequena) alteração no **index.html**;
* 5) Execute o comando git diff e veja esta alteração;
* 6) Desfaça esta última alteração;
* 7) Execute o comando git tag -a v0.1.0 para criar uma *tag* no seu código;
* 8) Execute o comando git push origin v0.1.0 para enviar esta *tag* para o GitHub;
* 9) Abra a página do repositório do GitHub que você criou e confira a aba de ***Releases***.

Nesta aula, aprendemos:

* Que é possível visualizar quais alterações foram realizadas em cada arquivo, com o comando git diff;
* Que, digitando apenas git diff, vemos as alterações em nossos arquivos que não foram adicionadas para commit (com git add);
* Que é possível comparar as alterações entre duas *branches* com git diff <branch1>..<branch2>
* Que é possível comparar as alterações feitas entre um commit e outro, através do comando git diff <commit1>..<commit2>;
* Que o Git nos possibilita salvar marcos da nossa aplicação, como por exemplo, lançamento de versões, através do git tag;
* Que o comando git tag -a é utilizado para gerar uma nova *tag*;
* As ***Releases*** do GitHub, que são geradas para cada *tag* do Git criada em nosso repositório.