**Aula 1**

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 (onde preferir) e dentro dela salve o arquivo **index.html**, com o seguinte conteúdo:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Cursos da Alura</title>

</head>

<body>

<ul>

<li>Vagrant</li>

<li>Docker</li>

<li>Ansible</li>

<li>Integração Continua</li>

</ul>

</body>

</html>COPIAR CÓDIGO

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

OBS: Caso o caminho da sua pasta possua espaços, é preciso colocá-lo entre aspas. Exemplo: cd 'Documents/Curso Git e GitHub'

3) Na pasta do projeto, execute o comando git init para inicializar um repositório Git;

4) Execute o comando git status para garantir que você está em um repositório Git e que o arquivo **index.html** é reconhecido. Leia com calma a saída deste comando. Nem tudo estará claro ainda, mas durante o treinamento ficará!

**Aula 2**

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

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

2) Execute o comando git add index.html para marcar o arquivo para ser salvo (*commitado*);

3) Execute git status e confira que o arquivo agora mudou de estado e está pronto para ser salvo (*commitado*);

4) Após adicionar, execute o comando git commit -m "Criando arquivo index.html com lista de cursos". Sinta-se à vontade para alterar a mensagem de *commit*, se desejar;

5) Altere o arquivo **index.html**. Adicione o acento em "Integração Continua", por exemplo;

6) Adicione o arquivo para ser salvo, com git add .;

7) Execute o comando git commit -m "Acento adicionado no curso de Integração Contínua". Sinta-se à vontade para alterar a mensagem de *commit*, se desejar;

8) Execute o comando git log e analise a sua saída. Execute também git log --oneline, git log -p e outras alternativas que quiser testar;

9) Crie um arquivo vazio com o nome que quiser, por exemplo, ide-config;

10) Crie o arquivo **.gitignore** e adicione uma linha com o nome do arquivo recém-criado (ide-config, no exemplo acima);

11) Execute git status e verifique que o arquivo **ide-config** não está na lista para ser adicionado;

12) Adicione (com git add .gitignore) e realize o commit (com git commit -m "Adicionando .gitignore") o arquivo **.gitignore**.

Nesta aula, aprendemos:

* Que um commit é a forma de salvar um estado ou versão do nosso código;
* Como adicionar arquivos para serem *commitados* com git add;
* Como *commitar* arquivos, utilizando o comando git commit;
* Como verificar o histórico de *commits*, através do git log e algumas de suas opções:
  + git log --oneline
  + git log -p
  + git log --pretty="parametros de formatação"
* Como fazer o Git não monitorar arquivos, através do **.gitignore**
* Que não devemos realizar commit, ou seja, salvar um estado, da nossa aplicação que não esteja funcionando.

**Aula 3**

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;

OBS: Não se esqueça do parâmetro **--bare**. Caso tenha executado o comando init sem esse parâmetro, execute na sequência o seguinte comando: git config core.bare true.

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) Acesse a pasta Projeto e 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 main 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 main 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 main para enviar as suas alterações para o repositório no GitHub.

**Aula 4**

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.

**Aula 5**

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>.

**Aula 6**

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.

Curso 2

1. No repositório do GitHub que você criou utilizando o seu usuário no curso anterior, clique na aba **Issues**;
2. Clique em **New issue** para criar uma nova issue para o seu projeto;
3. Defina o título como "Adicionar título na página" e a descrição como "A página contém apenas a lista dos cursos. Senti falta de um título";
4. Clique em **Submit new issue** para criar a issue em questão;
5. Crie um novo usuário no GitHub, que servirá como um colaborador fictítcio (como o exemplo da **analura**);
6. Com este novo usuário, vá até o seu repositório original e clique em **Fork**, para criar uma cópia dele;
7. Em seu computador, acesse a pasta do projeto como Ana, e execute git pull origin master. Confira que não há este repositório remoto;
8. Adicione o repositório fork ao projeto da Ana, utilizando git remote add origin {url\_do\_repositorio};
9. Execute git pull origin master para trazer todas as alterações para o projeto local da Ana;
10. Logo após a abertura da tag <body> no HTML, adicione <h1>Cursos de DevOps</h1>;
11. Execute o comando git config --local user.email "{email}", substituindo {email} pelo e-mail do usuário fictício que você criou no GitHub, para que o autor dos commits esteja correto;
12. Execute git add index.html e depois git commit -m "Título adicionado", para realizar o commit desta alteração;
13. Adicione uma quebra de linha logo após o título recém adicionado;
14. Execute git add index.html e depois git commit -m "Quebra de linha" para realizar o commit desta alteração;
15. Execute git push origin master para enviar as alterações para o GitHub do usuário fictício;

15.1. Se estiver utilizando Windows, limpe as credenciais do GitHub

15.2. Vá em **Painel de Controle --> Contas de Usuário --> Gerenciador de Credenciais --> Credenciais do Windows**;

15.3. Encontre a credencial do GitHub e clique em **Remover**, para excluí-la;

15.4. Execute git push origin master para, agora sim, enviar as alterações para o GitHub do usuário fictício;

1. Acesse o repositório cópia do seu usuário fictício;
2. Confira que há alterações que não estão presentes no repositório original;
3. Clique em **New pull request** para criar o pedido de envio de alterações do código original;
4. Analise as alterações, veja a lista de commits, e depois clique em **Create pull request** para confirmar a criação;
5. Defina "Adicionando título na página" como título do pull request e "Adicionando um título na página que antes não tinha." como descrição;
6. Clique em **Create pull request** para confirmar todos os dados;
7. Entre novamente como o seu usuário original do GitHub e confira a aba **Pull requests** em seu repositório;
8. Clique em **Merge pull request** e depois em **Confirm merge** para trazer este pull request para o seu repositório original;
9. Confira no GitHub que o código no repositório original foi atualizado com as alterações do usuário fictício;
10. Acesse, na aba **Issues**, a issue que você criou para adicionar título;
11. Digite na caixa de comentários "Fechado pelo PR #". Note que abrirá uma caixa mostrando o título do pull request. Clique nela;
12. Clique no botão **Close and comment**;
13. Acesse novamente o GitHub, utilizando o usuário fictício criado no passo 5;
14. No repositório do seu usuário original, crie uma nova issue com o título "Trocar ul por dl";
15. Em seu computador, no repositório da Ana, altere as tags <ul> por <dl> e as tags <li> por <dd>;
16. Execute git add index.html e depois git commit -m "Trocando UL por DL" para realizar o commit desta alteração;
17. Separe os títulos das descrições dos cursos, deixando o conteúdo da seguinte forma:
18. <dl>
19. <dt>Vagrant</dt>
20. <dd>Gerenciando máquinas virtuais</dd>
21. <dt>Docker</dt>
22. <dd>Criando containers sem dor de cabeça</dd>
23. <dt>Ansible</dt>
24. <dd>Sua infraestrutura como código</dd>
25. <dt>Integração Contínua</dt>
26. <dd>Maturidade e Produtividade no Desenvolvimento de Software</dd>
27. <dt>Kubernetes</dt>
28. <dd>Introdução a orquestração de containers</dd>

</dl>COPIAR CÓDIGO

1. Execute git add index.html e depois git commit -m "Separando os títulos", para realizar o commit desta alteração;
2. Adicione uma quebra de linha antes de cada título (<dt>);
3. Execute git add index.html e depois git commit -m "Quebras de linha" para realizar o commit desta alteração;
4. Execute git log --oneline para verificar que há três novos commits para a última issue criada;
5. Execute git rebase -i HEAD~3 para analisar quais dos três últimos commits serão mantidos ou unidos ao anterior;
6. Troque a palavra **pick**, nas linhas 2 e 3, por um **s** (de squash) para que ambos sejam unidos ao commit anterior, da primeira linha;

38.1. Se o editor de texto que for aberto for o **Vim**, para excluir uma palavra, digite dw. Para começar a digitar, aperte i, e então você pode adicionar o s na linha. Para salvar, aperte Esc e depois :x e Enter;

1. Na nova tela que abrir, altere a mensagem que será atribuida ao novo commit, para algo como "Trocando UL por DL e separando os títulos";
2. Execute git log --oneline novamente e confira que agora os três commits viraram apenas 1;
3. Repita os passos do 15 ao 27, trocando as mensagens e a issue pelo que fizer sentido para seu caso

Aula 2

1. Acesse a pasta do seu repositório no computador;
2. Execute git checkout master para trabalhar na branch master;
3. Execute git pull origin master e git pull local master para garantir que você está com o código atualizado e sincronizado com os dois repositórios remotos;
4. Execute git checkout -b novo-release para passar a trabalhar em um nova branch, chamada novo-release;
5. Substitua as tags <h1> por <h2> no código do arquivo **index.html**;
6. Execute git add index.html e git commit -m "h1 -> h2" para commitar esta alteração;
7. Confira o hash do commit que foi mostrado na saída do comando. Copie ele;
8. Altere o título do arquivo (na tag <title>) para "Lista de cursos";
9. Execute git add index.html e git commit -m "Simplificando o título" para commitar esta alteração;
10. Execute git checkout master para voltar a trabalhar na branch master (pois há um bug que você deve corrigir antes de continuar o trabalho do seu novo release);
11. Execute o comando git cherry-pick {hash} para trazer um commit para a branch master. Substitua {hash} pelo hash que foi copiado no passo 6;
12. Confira que apenas a alteração desejada foi aplicada no master, e não toda a branch novo-release;
13. Execute o comando git rebase novo-release e confira que o Git executa apenas um trabalho, embora hajam dois commits na nova branch, pois um já foi trazido para o master;
14. Altere o título da página (na tag <title>) para "Lista de cursos de DevOps";
15. Execute git add index.html e git commit -m "Adicionando DevOps no título" para commitar esta alteração;
16. Altere o título da página (na tag <title>) para "Lista de cursos da Alura";
17. Execute git add index.html e git commit -m "Mudando título" para commitar esta alteração;
18. Altere o título da página (na tag <title>) para "Cursos da Alura";
19. Execute git add index.html e git commit -m "Mexendo no título" para commitar esta alteração;
20. Altere o título do arquivo (na tag <title>) para "Lista de cursos";
21. Execute git add index.html e git commit -m "Título alterado" para commitar esta alteração;
22. Execute o comando git bisect start para informar ao Git que você vai iniciar uma busca por determinada alteração;
23. Execute o comando git bisect bad HEAD para informar que o estado atual do código está "ruim", ou seja, o título não está no estado que você quer;
24. Executando git log --oneline, copie o hash do commit com a mensagem "Simplificando o título";
25. Execute o comando git bisect good {hash}, substituindo {hash} pelo hash copiado no passo anterior, para informar que o estado atual do código está "bom", ou seja, o título está no estado que você quer;
26. Confira no código que agora o título está como você quer;
27. Execute git bisect good para informar ao Git que neste commit o código ainda está como você quer;
28. Confira no código que agora o título não está mais como você quer;
29. Execute git bisect bad para informar ao Git que neste commit o código não está mais como você quer;
30. Note que o Git encontrou o exato commit onde o título deixou de estar no estado em que você quer;
31. Execute git bisect reset para indicar que você finalizou a busca;
32. Execute o comando git show {hash}, substituindo {hash} pelo hash conferido no passo 30. Veja que é exatamente a alteração que você estava buscando;
33. Para reverter esta alteração, execute git revert {hash}, substituindo {hash} pelo hash conferido no passo 30. Resolva quaisquer conflitos que possam ter sido encontrados;
34. Execute git blame index.html e confira o responsável por cada linha do arquivo.

* Que o comando git cherry-pick pode trazer um *commit* específico para a *branch* atual;
* Como encontrar o *commit* em que determinada alteração foi aplicada, utilizando o git bisect;
* Como encontrar o responsável por determinada linha ou bloco de código, utilizando o git blame;
* Que jamais devemos apontar um culpado por determinado *bug*. Uma equipe deve ser unida e se ajudar;
* Que o comando git show {hash} mostra todas as alterações aplicadas pelo *commit* com o *hash* informado.

Aula 3

1. Execute git log --graph e confira que, embora haja várias linhas de desenvolvimento, tudo começa e termina na linha da branch master;
2. Execute o comando git branch -d titulo para remover a branch titulo;
3. Execute o comando git branch -D novo-release para remover a branch novo-release, mesmo que ela tenha commits à frente do master;
4. Execute o comando git branch development para criar um nova branch de desenvolvimento chamado development;
5. Execute o comando git checkout development para passar a trabalhar na branch development;
6. Execute o comando git checkout -b feature/lista-cursos-cloud para criar um nova branch, chamada feature/lista-cursos-cloud;
7. Adicione o seguinte conteúdo após a lista de cursos de DevOps:
8. <h2>Cursos de Cloud:</h2>
9. <dl>
10. <dt>Amazon EC2</dt>
11. <dd>Faça um deploy da sua webapp com alta disponibilidade e escalabilidade.</dd>
12. <dt>Amazon S3</dt>
13. <dd>Manipule e armazene objetos na nuvem</dd>

</dl>COPIAR CÓDIGO

1. Execute git add index.html e git commit -m "Cursos de EC2 e S3 adicionados" para commitar esta alteração;
2. Adicione a esta lista os dois seguintes cursos:
3. <dt>Amazon Elastic Beanstalk Parte 1</dt>
4. <dd>Container Docker</dd>
5. <dt>Amazon ECS</dt>

<dd>Gerencie Docker na nuvem da AWS</dd>COPIAR CÓDIGO

1. Execute git add index.html e git commit -m "Cursos de Beanstalk e ECS adicionados" para commitar esta alteração;
2. Execute git checkout development para voltar para a branch de desenvolvimento;
3. Execute git checkout -b feature/lista-cursos-linux para passar a trabalhar em um nova branch, chamada feature/lista-cursos-linux;
4. Adicione o seguinte conteúdo após a lista de cursos de DevOps:
5. <h2>Cursos de Linux:</h2>
6. <dl>
7. <dt>Linux I</dt>
8. <dd>Conhecendo e utilizando o terminal</dd>
9. <dt>Linux II</dt>
10. <dd>Programas, processos e pacotes</dd>

</dl>COPIAR CÓDIGO

1. Execute git add index.html e git commit -m "Cursos de Linux I e II adicionados" para commitar esta alteração;
2. Execute git checkout development para voltar para a branch de desenvolvimento;
3. Execute o comando git branch para conferir quantas e quais branches você têm criadas atualmente;
4. Execute o comando git merge feature/lista-cursos-linux para trazer os cursos de Linux para a branch de desenvolvimento;
5. Execute o comando git checkout feature/lista-cursos-cloud para voltar à branch feature/lista-cursos-cloud;
6. Adicione, à lista de cursos de Cloud, o seguinte curso:
7. <dt>Google Cloud</dt>

<dd>Deploy de uma aplicação em Spring MVC</dd>COPIAR CÓDIGO

1. Execute git add index.html e git commit -m "Curso de Google Cloud adicionado" para commitar esta alteração;
2. Execute git checkout development para voltar para a branch de desenvolvimento;
3. Execute o comando git merge feature/lista-cursos-cloud para trazer os cursos de Cloud para a branch de desenvolvimento;
4. Resolva o conflito mantendo as duas listas novas de cursos;
5. Execute git add index.html e git commit para continuar o processo de merge;
6. Volte para a branch master com o comando git checkout master;
7. Trabalhe em um nova branch, chamada hotfix/v0.1.1, utilizando o comando git checkout -b hotfix/v0.1.1;
8. Altere o título da página (na tag <title>) para "Lista de cursos";
9. Execute git add index.html e git commit -m "Removendo o nome da Alura do título" para commitar esta alteração;
10. Volte para a branch master, com o comando git checkout master;
11. Una o trabalho do hotfix, com o comando git merge hotfix/v0.1.1;
12. Crie uma nova tag que representa uma nova versão, utilizando o comando git tag -a v0.1.1 -m "Versão com correção no título";
13. Vá para a branch de desenvolvimento com git checkout development;
14. Traga a correção do bug com git merge hotfix/v0.1.1;
15. Crie um nova branch, chamada de release/v0.2.0, com o comando git checkout -b release/v0.2.0;
16. Adicione os dois seguintes cursos na lista de cursos de Linux:
17. <dt>Shell Script I</dt>

<dd>Começando seus scripts de automação de tarefas</dd>COPIAR CÓDIGO

<dt>Shell Script II</dt>

<dd>Fazendo monitoramento, agendando tarefas e backup</dd>COPIAR CÓDIGO

1. Execute git add index.html e git commit -m "Corrigindo bug: Cursos de Shell faltantes" para commitar esta alteração;
2. Volte para a branch master, com o comando git checkout master;
3. Una todo o trabalho referente à nova release, com git merge release/v0.2.0;
4. Crie uma nova tag que representa uma nova versão, utilizando o comando git tag -a v0.2.0 -m "Novas listas adicionadas";
5. Execute o comando git branch e confira todos as branches criadas neste processo;

Aula 5

1. Na linha de comando, acesse, dentro da pasta do seu repositório, a pasta **.git/hooks**;
2. Dentro dessa pasta, crie o arquivo **pre-commit**, com o seguinte conteúdo:
3. #!/bin/sh

echo "Você está prestes a commitar. :-)"COPIAR CÓDIGO

1. Se você estiver em uma plataforma diferente do Windows, digite chmod u+x pre-commit para dar permissão de execução a este arquivo;
2. Execute alguma alteração qualquer no código;
3. Digite os comandos git add index.html e git commit -m "Commit teste" para realizar um commit;
4. Verifique que a mensagem "Você está prestes a commitar. :-)" foi exibida, logo, o seu hook foi executado com sucesso;
5. Fora desta pasta, crie uma nova pasta chamada **web**;
6. Dentro da pasta que você criou no curso anterior para ser o seu servidor remoto, acesse a pasta **.git/hooks**;
7. Nesta pasta, crie um arquivo chamado **post-receive** com o seguinte conteúdo:
8. #!/bin/sh

git --git-dir={caminho\_da\_pasta\_do\_servidor} --work-tree={caminho\_da\_pasta\_web} checkout -fCOPIAR CÓDIGO

1. Substitua acima {caminho\_da\_pasta\_do\_servidor} pelo caminho da pasta criada no curso anterior para ser seu servidor remoto, e {caminho\_da\_pasta\_web} pelo caminho da pasta criada no passo 7;
2. De volta para a pasta que contém o seu código, execute o comando git push local master para enviar as alterações feitas no passo 4 para o servidor;
3. Verifique que agora o arquivo **index.html** está presente na pasta criada no passo 7, ou seja, você fez o deploy da sua aplicação.