Como comentado anteriormente, é sempre interessante mantermos na master o código pronto para produção. Criaremos outro branch para abrigar as alterações feitas, que enviaremos ao master somente quando estiver pronto. Já temos dois branches que não estão sendo utilizados, novo-release e titulo, que removeremos com git branch -d seguido do título.

Caso haja algum conflito, como o branch a ser deletado estar à frente do branch atual, ou seja, se novo-release tivesse alguns commits à frente do master, teríamos que utilizar git branch -D novo-release, com "D" em maiúsculo.

Criaremos um branch de desenvolvimento por meio de git branch development, em que começaremos a testar nossas features, após executarmos git checkout development. Em seguida, partindo deste novo branch criaremos outro, para acrescentarmos uma nova lista de cursos, com git checkout -b feature/lista-cursos-cloud.

No VS Code, digitaremos o seguinte, abaixo da primeira lista de cursos:

<h2>Cursos de Cloud</h2>

<dl>

<dt>Amazon EC2</dt>

<dd>Faça um deploy da sua webapp com alta disponibilidade e escalabilidade.</dd>

<dt>Amazon S3</dt>

<dd>Manipule e armazene objetos na nuvem</dd>

</dl>COPIAR CÓDIGO

Feito isto, criaremos um commit com git add index.html e git commit -m "Cursos de EC2 e S3 adicionados". Depois, adicionaremos mais dois cursos:

<h2>Cursos de Cloud</h2>

<dl>

<dt>Amazon EC2</dt>

<dd>Faça um deploy da sua webapp com alta disponibilidade e escalabilidade.</dd>

<dt>Amazon S3</dt>

<dd>Manipule e armazene objetos na nuvem</dd>

<dt>Amazon Elastic Beanstalk Parte 1</dt>

<dd>Container Docker</dd>

<dt>Amazon ECS</dt>

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

</dl>COPIAR CÓDIGO

Commitaremos com git add index.html e git commit -m "Cursos de Beanstalk e ECS adicionados". Enquanto estamos desenvolvendo, outra pessoa da equipe estará tratando dos cursos de Linux, então, a partir do branch de desenvolvimento, criaremos outro branch: git checkout -b feature/lista-cursos-linux. No VS Code, acrescentaremos mais estas linhas de código:

<h2>Cursos de Linux</h2>

<dl>

<dt>Linux I</dt>

<dd>Conhecendo e utilizando o terminal</dd>

<dt>Linux II</dt>

<dd>Programas, processos e pacotes</dd>

</dl>COPIAR CÓDIGO

Executaremos git add index.html e git commit -m "Cursos de Linux I e II adicionados". Voltaremos ao branch de desenvolvimento com git checkout development, e verificaremos a quantidade de branches que temos no momento com git branch, além da master — um com os códigos em desenvolvimento (development), e um branch para cada feature, um para a lista de cursos Cloud e outro para a lista de cursos Linux.

Finalizada a lista de cursos de Linux, poderemos trazê-la para o branch de desenvolvimento, com git merge feature/lista-cursos-linux. Verificaremos se tudo está dentro dos conformes no VS Code, e com git checkout feature/lista-cursos-cloud faremos uma alteração acrescentando mais um curso à lista de cursos de Cloud.

<dt>Google Cloud</dt>

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

Estando no branch da lista de cursos de Cloud, executaremos git add index.html e git commit -m "Curso de Google Cloud adicionado". Por fim, iremos ao branch de desenvolvimento por meio de git checkout development e faremos um merge com git merge feature/lista-cursos-cloud. Obteremos, porém, um conflito indicando que, no trabalho atual, temos dois branches distintos, um com lista de cursos Linux e outro com a lista de cursos de Cloud.

Como queremos manter ambos, simplesmente corrigiremos o código manualmente. Notem como foi tranquilo lidar com este conflito por estarmos lidando com branches separados, o que traz todas as features de uma só vez para que possamos resolvê-lo. Adicionaremos o index.html e commitaremos para que se continue o merge.

Assim, em nosso branch de desenvolvimento temos as listas de cursos Cloud e Linux, e um branch para cada feature. Poderemos testar como estas duas features desenvolvidas separadamente funcionam em conjunto. Porém, ao voltarmos ao código master com git checkout master, encontraremos um bug, pois há o título "Lista de Cursos da Alura", sendo que poderemos ter cursos presenciais da Caelum.

Neste caso, por mais urgente que seja a correção de um bug, **não devemos commitar na master**. Portanto, criaremos git checkout -b hotfix/v0.1.1 a partir da master, em que "hotfix" remete a um "conserto rápido de um bug", indicando também a versão em que será implementada esta correção.

Em seguida, trocaremos "Lista de Cursos da Alura" para "Lista de Cursos" manualmente, em index.html, e executaremos git add index.html e git commit -m "Removendo nome da Alura do título" no Terminal. Por se tratar de um bug, não passaremos ao branch de desenvolvimento, e sim à produção, com git checkout master e git merge hotfix/v0.1.1. Como temos uma nova versão a ser lançada, utilizaremos o comando git tag -a v0.1.1. A mensagem para a tag será "Versão com correção no título".

Neste vídeo, criamos dois branches de desenvolvimento, de features, um para trazer todas as features, e um de correção de bug, que enviamos diretamente à master. Agora, é preciso trazer todas as correções ao branch de desenvolvimento, com git merge hotfix/v0.1.1 após git checkout development. Temos, então, tudo o que é necessário para lançarmos uma nova versão no branch de desenvolvimento.

Para isto, é necessário realizarmos alguns testes, por isso criaremos um branch chamado "release", de versão 0.2: git checkout -b release/v0.2.0, com a correção aplicada na versão anterior e, além disso, novas listas. Entretanto, esquecemos de incluir cursos de Shell Script na lista dos cursos de Linux! Faremos isto no branch de release, pois a feature já foi desenvolvida. Teremos:

<h2>Cursos de Linux</h2>

<dl>

<dt>Linux I</dt>

<dd>Conhecendo e utilizando o terminal</dd>

<dt>Linux II</dt>

<dd>Programas, processos e pacotes</dd>

<dt>Shell Script I</dt>

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

<dt>Shell Script II</dt>

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

</dl>COPIAR CÓDIGO

Realizaremos o commit no branch desta release, por meio de git add index.html, git commit -m "Corrigindo bug: Cursos de shell faltantes". A release estará pronta, o código estará compilando sem problemas, então poderemos colocá-lo em produção, mas antes disto precisaremos enviá-lo para a master, com git checkout master e git merge release/v0.2.0.

Assim, no nosso código de produção teremos a correção do título, as novas listas, referentes aos cursos de Linux e Cloud, e a correção da release. E se temos uma nova versão, teremos uma nova tag — executaremos git tag -a v0.2.0 -m "Novas listas de cursos adicionadas". Ao executarmos git branch, verificaremos a existência do branch que terá nosso estado mais próximo da versão futura, junção de todos os branches de feature, as quais se juntarão ao branch de desenvolvimento.

Quando tudo isto estiver mais testado e próximo de ir à produção, criamos um branch de release para passar à equipe de qualidade para que eles possam testar, e se houver algum problema muito sério, pode-se resolver diretamente no branch de release. Mas se surgir um bug de algo que está em produção, precisaremos corrigir a partir da master, criando um novo branch para correção, e enviando de volta para a master e para o branch de desenvolvimento, que precisa estar **sempre atualizado**.

Nesta aula, aprendemos:

* Que é uma convenção bem seguida que a *branch* master tenha apenas os *commits* prontos para ir para produção;
* Que não é interessante realizar trabalho e *commitar* diretamente na *branch* master;
* Como remover uma *branch*:
  + git branch -d {nome\_branch} remove uma *branch* que já tem seu trabalho unido à *branch* atual;
  + git branch -D {nome\_branch} remove uma *branch* mesmo que os *commits* desta *branch* ainda não estejam na *branch* atual, ou seja, força a remoção;
* Um pouco do processo chamado de ***Git Flow***:
  + Entendemos que o estado do código representado pela *branch* master deve ser o mesmo que estará em produção
  + Vimos que deve haver uma *branch* de desenvolvimento (comumente chamado de develop), onde todas as funcionalidades e correções devem ser muito bem testadas antes de ir para produção (master)
  + Vimos que cada funcionalidade deve ser feita em uma *branch* separada, e que é comum que esta *branch* tenha feature/ como prefixo
  + Aprendemos também que *bugs* normalmente são corrigidos em *branches* separadas, com o prefixo hotfix/
  + Além disso, *branches* específicas para cada *release* são criadas para realizar os testes e correções de *bugs* específicos