Workshop de Git/ Github

Como versionar seu código

Lenildo Luan

Davi Sousa

Rafael Casamaximo

Realização:

Patrocínios:

O que é Git

O que é Git

git

É um sistema de controle de versão!

O que é Git

git

- Software que acompanha cada mudança feita nos arquivos;
- Separa quem mexeu em qual arquivo;
- Geralmente é usado para arquivos de código.

Resumindo

git

- Git é uma ferramenta que protege você de você mesmo.
- Usando Git você pode modificar, excluir, alterar, quebrar tudo no seu projeto sabendo que tudo está salvo.

Diferença

- Sistema de controle de versão, uma ferramenta para gerenciar o histórico de código.

- Um serviço de hospedagem de Repositórios Git

Instalando

git

git-scm.com/download

Realização:

Patrocínios:

Instalando

git config --global user.name “Seu nome”

git config --global user.email “seuemail@seuemail.com”

Repositórios

Realização:

Patrocínios:

O que é

- É o diretório (pasta) onde o projeto, ou os arquivos, ficam salvos;
- Contém uma pasta .git, onde são salvos todos os arquivos que o git precisa pra funcionar;
- Essa pasta pode estar oculta, dependendo da configuração.

O que é

- Basta entrar numa pasta já criada e usar o seguinte comando:

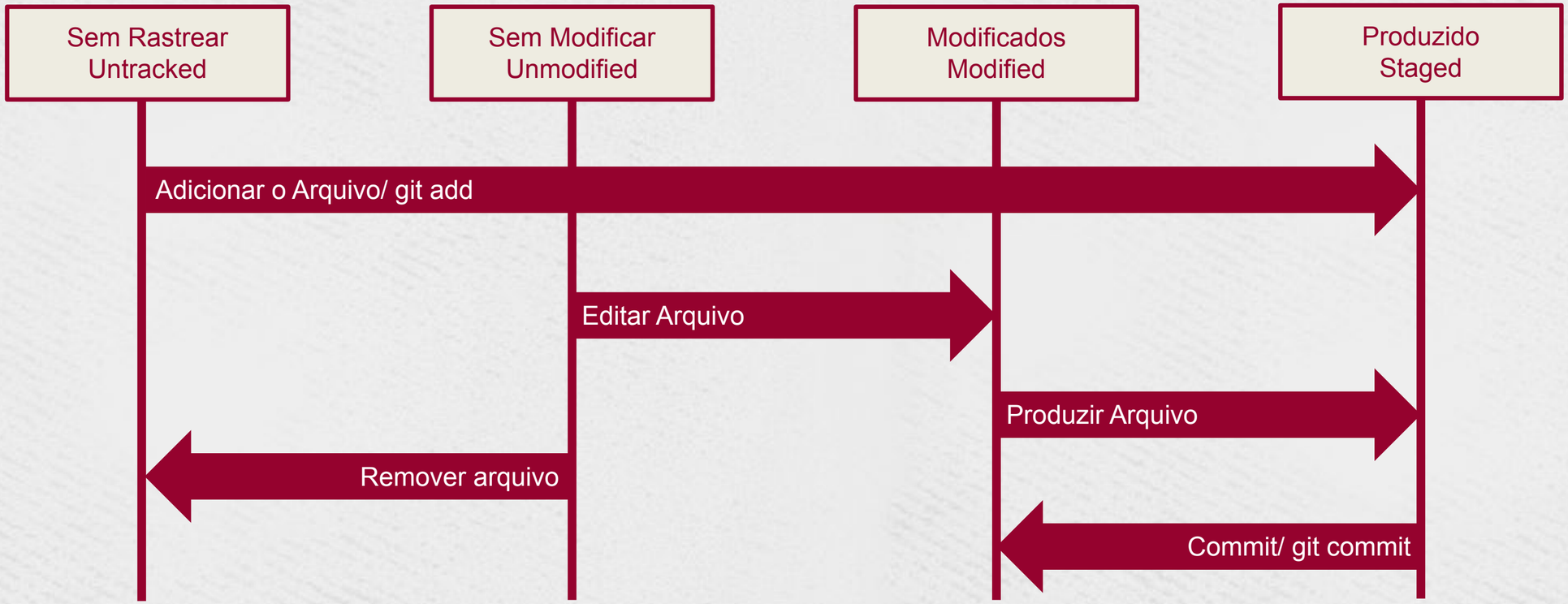
```
C:\Users\RASWEEK>cd caminho/ate/a/pasta/
```

```
C:\Users\RASWEEK\pastaProjeto> git init
```

OBS.: Nenhum arquivo está sendo acompanhado pelo Git ainda. Faremos isso em breve.

Salvando alterações

Áreas do Repositório



Working Directory

- O *working directory* é o ‘local’ onde estão os arquivos modificados, os não modificados e até mesmo os que não são acompanhados.
- Ele é, como o nome já diz, o **diretório de trabalho** do usuário.

Staging Area

- A *staging area* é um local que guarda as alterações do repositório a serem salvas, antes de você confirmar a mudança. Quase como uma lista de alterações.
- **Apenas o que foi incluído na staging area vai ser salvo.** O que não for vai continuar constando como modificado.

Diretorio .git

- É onde os arquivos ficam depois das mudanças serem confirmadas. Essa mudanças passam a fazer parte do working directory novo, até que sejam modificadas novamente.

Comandos

Realização:

Patrocínios:

Visualizando Mudanças

- Você pode visualizar as alterações em arquivos usando o comando:

```
git status
```

- Ou ainda pode ver todas as versões do seu código usando o comando:

```
git log
```

Visualizando Mudanças

- Você pode usar um outro comando, para visualizar mais profundamente o que foi alterado em cada arquivo, utilizando o comando:

```
git diff
```

Adicionando Arquivos

- Para adicionar arquivos à staging area, é necessário usar o seguinte comando:

```
git add <arquivo> [<arquivo2> <arquivo3> ...]
```

```
git add <diretorio> [<diretorio2> <diretorio3> ...]
```

```
git add .
```

Commitando Arquivos

- Commits são a ação mais importante de todo o git.
- O commit é o que salva as alterações que estão na staging area e salva uma nova “versão segura” do projeto. É como um checkpoint em um jogo.
- O git nunca irá mudar um commit, a menos que explicitamente pedido (e é muito difícil de fazer).

Commitando Arquivos

- Importante: o commit é apenas para o repositório local. Caso outra pessoa tenha o seu repositório, esse commit não irá alterar nada no computador dela.
- Todo commit tem uma mensagem, que é uma descrição do que aconteceu nele. Ela serve para uma fácil visualização posteriormente.

Commitando Arquivos

- Esse é o comando básico para fazer um commit:

```
git commit -m "<mensagem>"
```

- Nesse comando, a mensagem é feita diretamente nele. Sempre coloque mensagens explicativas sobre o que aquele commit está fazendo.

Commitando Arquivos

Enviando Mudanças

- Depois de fazer o commit de alguns arquivos, você pode enviar essas mudanças para um repositório remoto. Mas antes disso, você precisa se conectar a um repositório remoto. Para isso, use:

```
git remote add origin <servidor>
```

- Coloque no campo do servidor, o endereço do repositório que você criou no github.

Enviando Mudanças

- Depois de fazer o commit de alguns arquivos, você pode enviar essas mudanças para um repositório remoto. Mas antes disso, você precisa se conectar a um repositório remoto. Para isso, use:

```
git push origin master
```

Recebendo Mudanças

- Para atualizar o repositório local com a versão mais nova, use na pasta do repositório:

```
git pull
```

Voltando Mudanças

- Uma das funções do git é atuar como um backup dos arquivos em seu repositório. Feito isso, é necessário saber voltar esse “backup” feito.
- É isso que será mostrado nessa seção.

Voltando Mudanças

- Durante o desenvolvimento de um projeto, é comum querermos visualizar commits antigos. Ou até mesmo voltar completamente um arquivo para como ele era em um commit anterior.
- É possível fazer isso com o comando:

```
git checkout <commit>
```

- Substitua <commit> pelo numero do commit (tag), que é mostrado no comando *git log,* explicado anteriormente.

Voltando Mudanças

- Para voltar para a versão mais atual do seu projeto, utilize o comando:

```
git checkout master
```

Voltando Mudanças

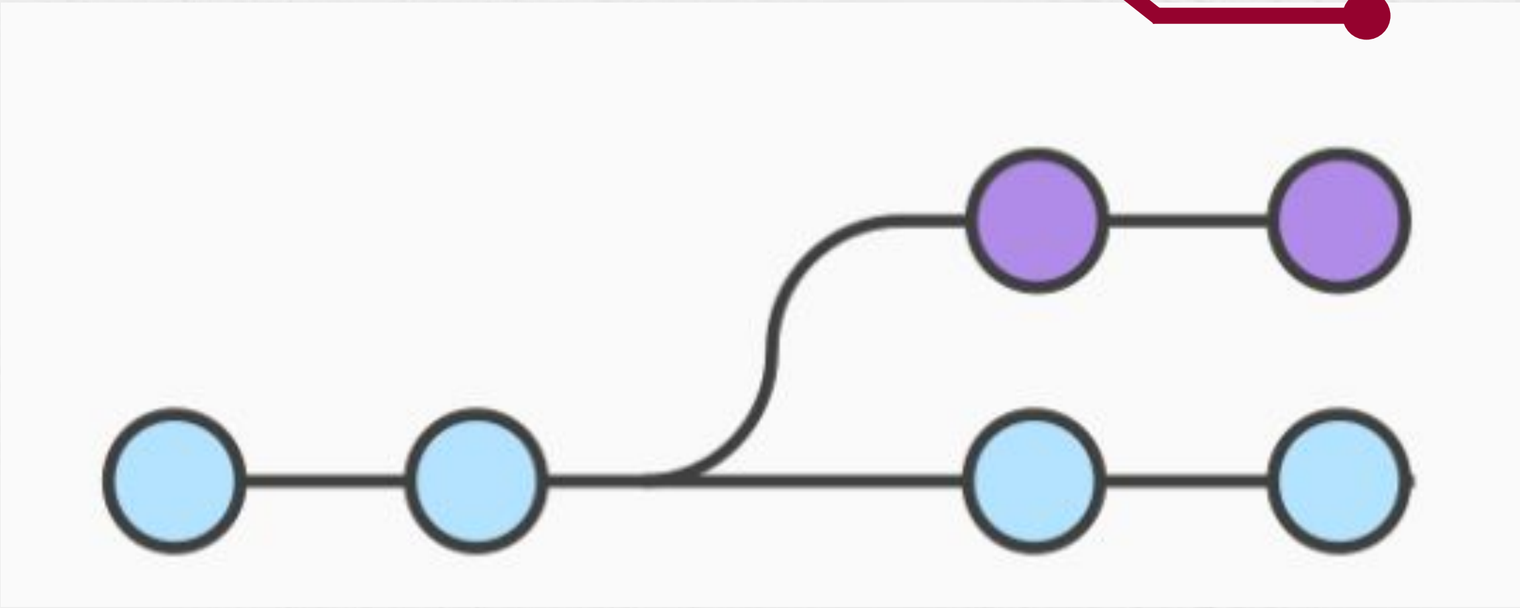
- A vantagem de se utilizar o Sistema de Controle de Versão é ter uma cópia “segura” do projeto. Ou seja, você não precisa se preocupar em fazer alterações e perder as modificações anteriores.
  O git se preocupa por você.
- Quando você faz um checkout de um commit inteiro, é como se fosse uma alteração read-only. Não tem como “quebrar a corrente” de commits criado ao longo do projeto.

Outras Funções

Ramificações

- Um branch é uma linha de desenvolvimento paralelo dentro do projeto.
- Quando um novo branch é criado, é como se o git criasse novos working directory, staging area e um histórico de commits, a partir dos já existentes.
- Todos os commits novos são salvos na branch atual.

Ramificações



- Cada círculo é um commit. Ao criar um novo branch, é gerada uma bifurcação no projeto.

Ramificações

- É recomendado que qualquer novo desenvolvimento, como uma funcionalidade ou mudança de layout, seja feito em um branch novo. Dessa forma, códigos instáveis ou com conflitos não afetam o código principal.

Ramificações

- O comando git branch é usado para criar, renomear e deletar branches.

```
git branch
```

- Esse comando lista todos os branches já existentes no repositório atual.

Ramificações

- O comando para criar um novo branch é:

```
git branch <nome>
```

- Com esse comando você não entra dentro do branch que acabou de criar. Para criar um branch e entrar dentro dele, use:

```
git checkout -b <nome>
```

Ramificações

- Se você quiser deletar um branch chamado <nome>, use o comando:

```
git branch -d <nome>
```

- Se esse branch tiver commits que não foram incorporados a outro branch, o git não permite deletar. Para deletar mesmo assim, use:

```
git branch -D <nome>
```

Ramificações

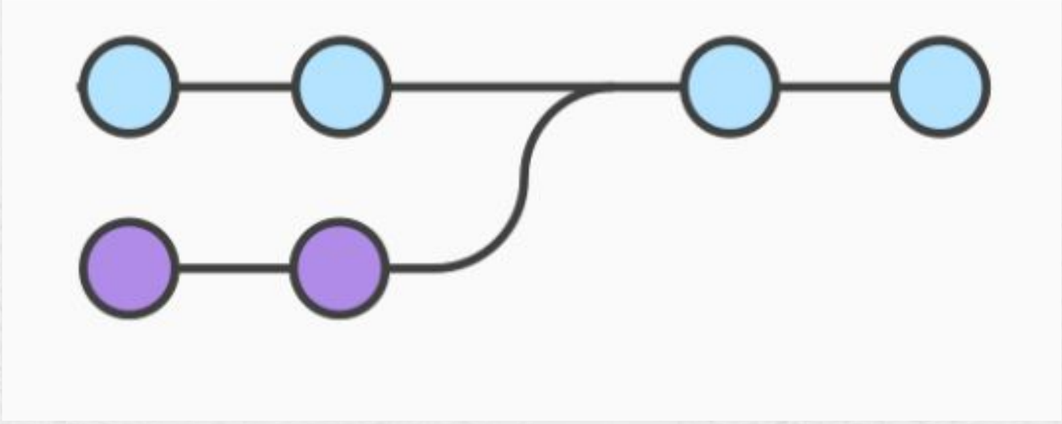
- O comando para criar renomear um branch já criado e:

```
git branch -m <novo-nome>
```

- Caso o branch já tenha sido enviado para o repositório remoto, a branch do remoto não será renomeada.

Ramificações

- Dentro de uma ramificação, você pode fazer commits e alterações necessárias.
- Após finalizar o necessário, você pode juntar de volta essas ramificações!

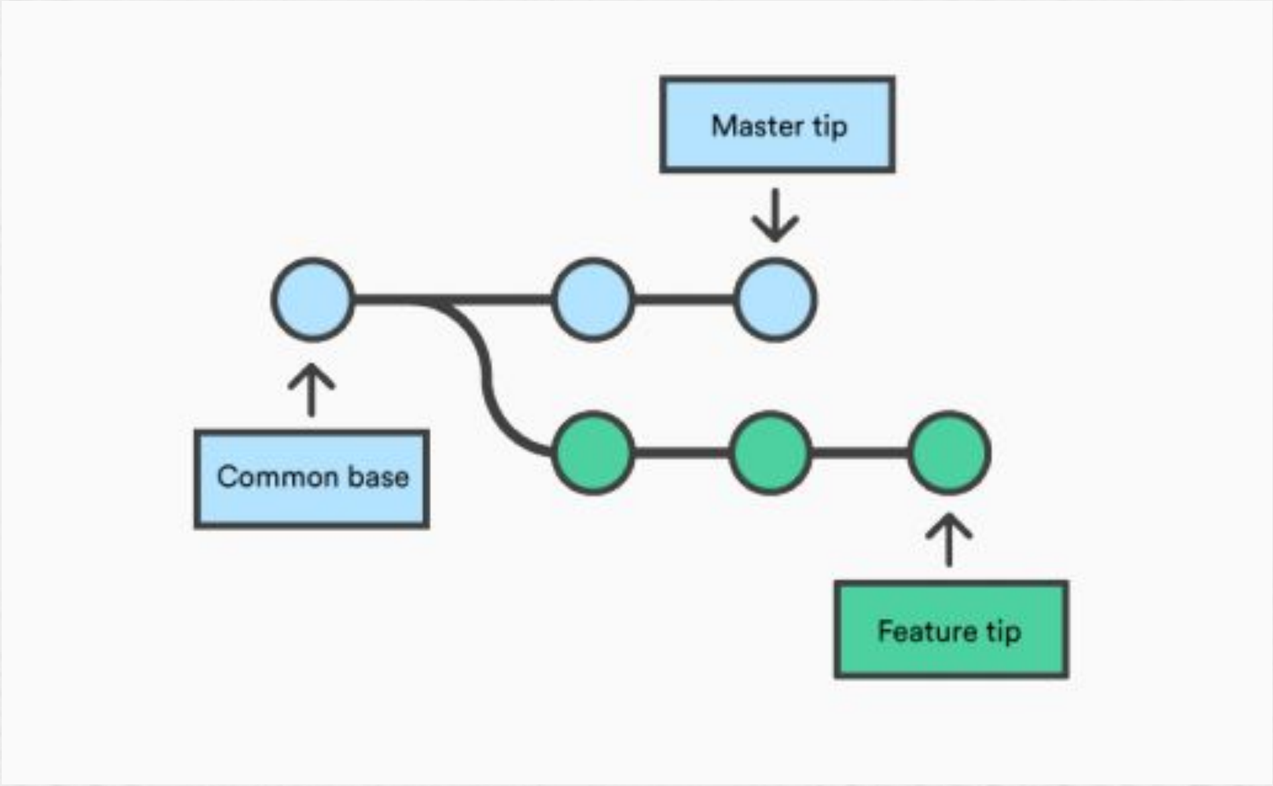


Juntando Ramificações

- Para juntar dois branches, você pode usar o comando:

```
git merge
```

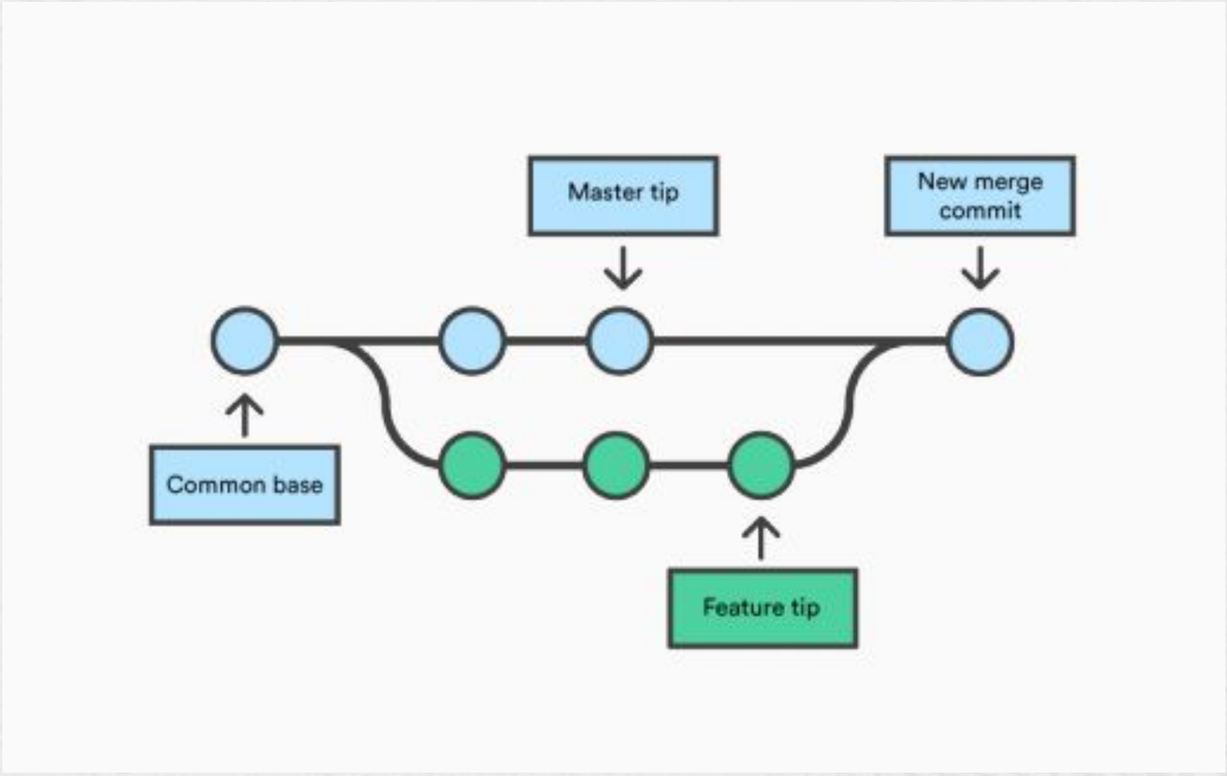
Juntando Ramificações



Realização:

Patrocínios:

Juntando Ramificações



Juntando Ramificações

- Para fazer o merge em outro branch, use o comando:

```
git merge <alvo>
```

- Esse comando traz os arquivos da branch <alvo> para a branch atual. Para isso, um novo commit é criado no branch atual com as alterações adicionadas pelos commits no branch <alvo>
- OBS: Lembre-se de remover o branch quando ele passar a ser desnecessário, isso evita poluir o seu repositório com muitas branchs antigas.

Conflitos

- Pode ser que, ao tentar fazer um merge, ocorra algum conflito. Ou seja, uma mesma linha de código é modificada em ambos os branches a serem mesclados.

O Git não escolhe nenhum.

Ele força você a escolher manualmente!

Conflitos

- Exemplo de conflitos:

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
please contact us at support@github.com
</div>
>>>>>>> <branch>:index.html
```

Conflitos

- Nesse exemplo de conflito, temos três mensagens padronizadas que são mostradas na tela:

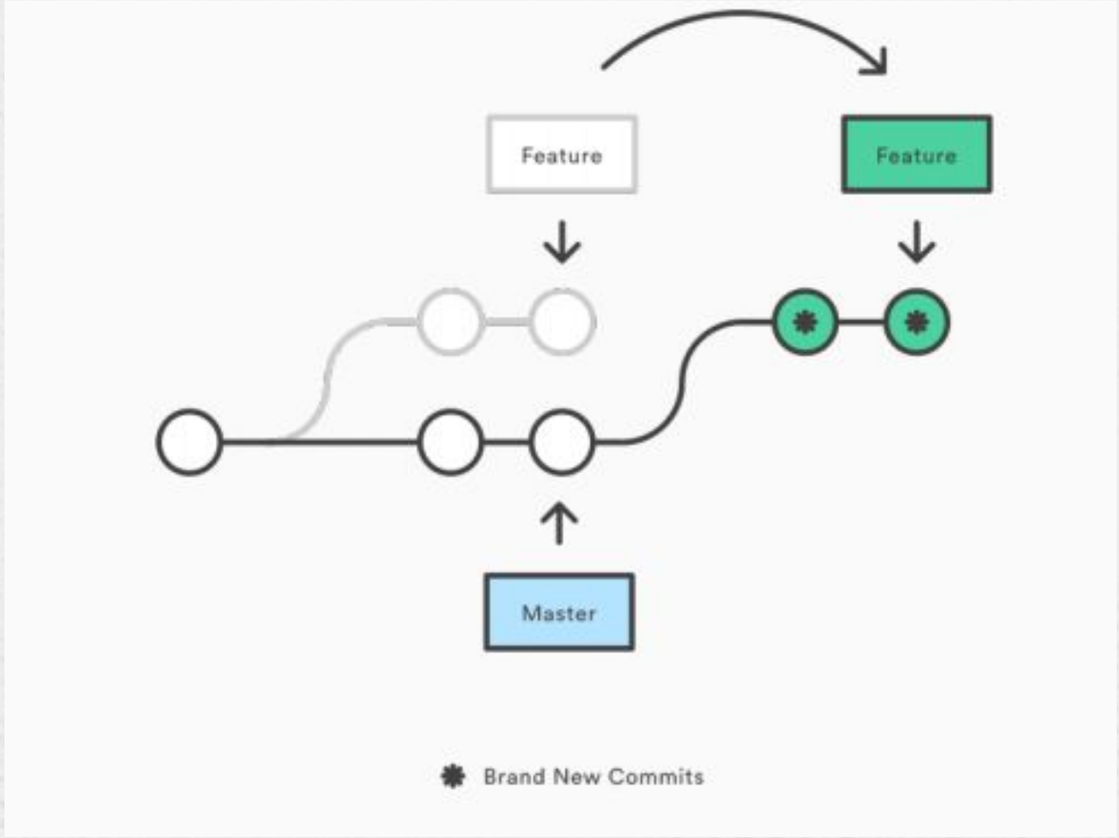
1. ==========
2. <<<<<<<< HEAD:<arquivo>
3. >>>>>>>> <branch>:<arquivo>

- O número 1 é um separador.
- O que está entre o número 2 e o separador é o que está no branch atual.
- Já o que está entre o 3 e o separador é o que está no branch alvo, que está sendo mesclado.

Rebase

- Existe um outro comando com a mesma funcionalidade do git merge, que deixa os commits lineares. Esse comando muda a base de um branch para um novo commit.

Rebase



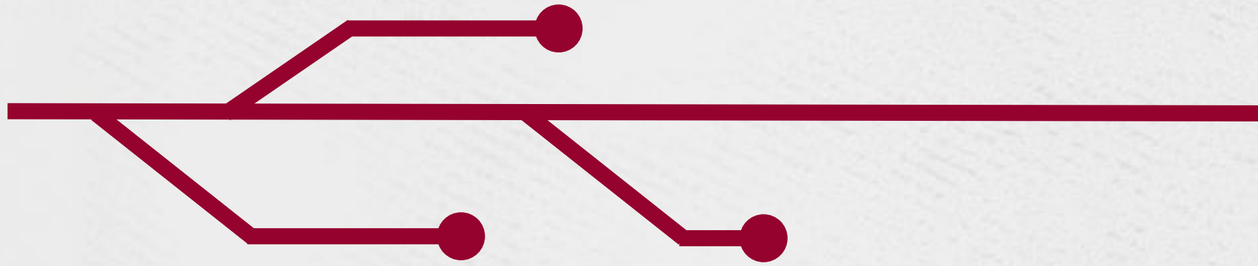
Rebase

- O comando para fazer isso é:

```
git rebase <nova-base>
```

- Esse comando faz o rebase do branch atual para <nova-base>. <nova-base> pode ser qualquer tipo de referência a um commit, como o identificador, um nome de branch ou uma tag.

Git Flow



Week

Realização: RAS UFPB · IEEE RAS UFCG · RAS UEL · RAS Robotics & Automation Society · RASIFPB · IEEE RAS UNIFACS Student Chapter Robotics & Automation Society · IEEE RAS CIMATEC Robotics and Automation Society · RAS Student Chapter Robotics and Automation Society IEEE UFRB

Patrocínios: Byteflop Componentes Eletrônicos · teoriental! · CNN Centro Avançado de Neuro e Coluna

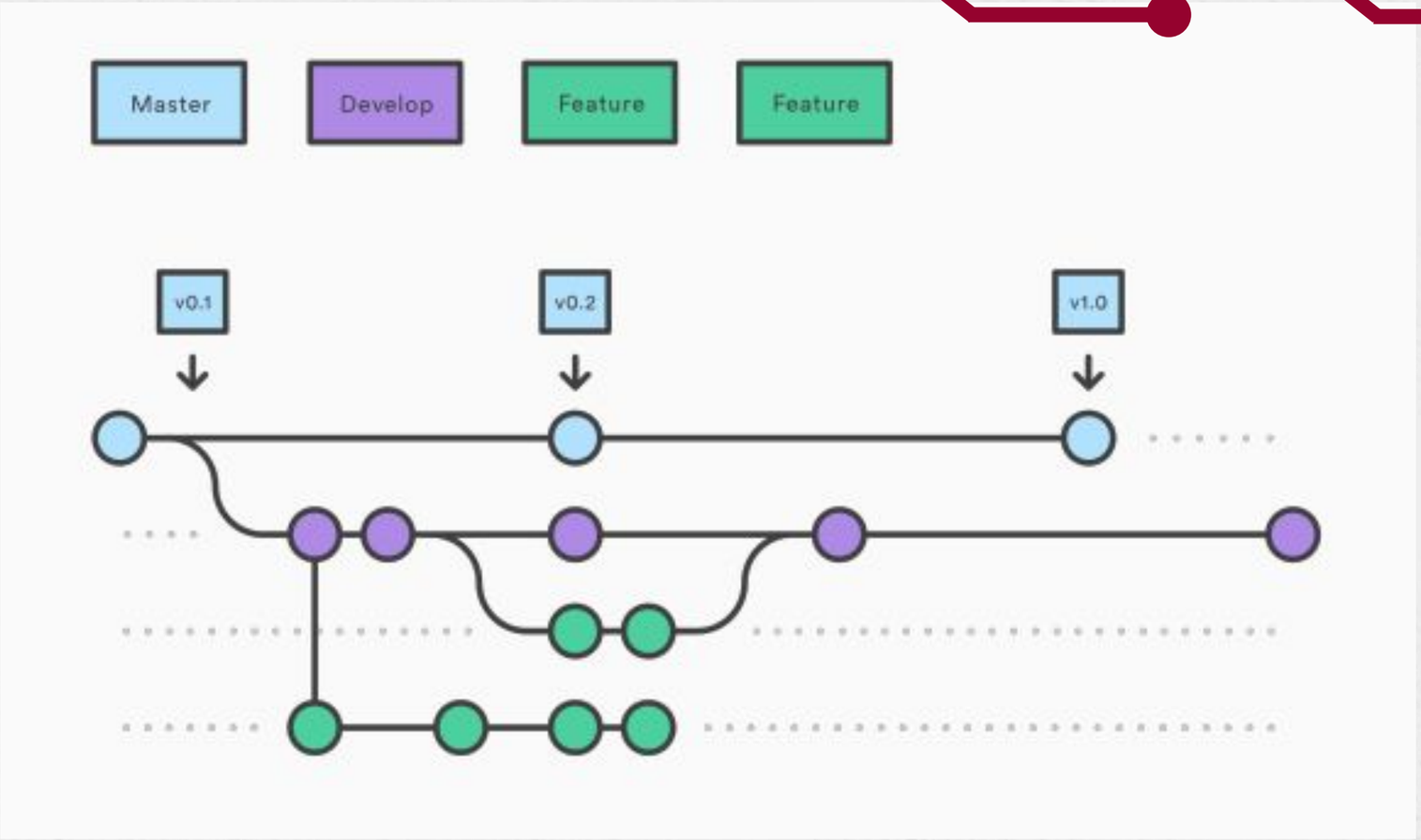
Git Flow

- Uma convenção usada no mercado de trabalho para lidar com branches e conflitos é o Git Flow.
- Nele, existe uma branch principal, a master, que sempre estará o código principal, estável,
- Com base na master, é criada a branch develop, que fica todo o código que está sendo desenvolvido. Ao terminar uma versão estável do código na develop, ela é integrada na master.

Git Flow

- Para a criação de funcionalidades novas baseadas na develop, são criados os features branches.
- Neles são desenvolvidas as funcionalidades que, quando finalizadas, são reintegradas na develop.

Git Flow



Realização:

Patrocínios:

Praticando

- Outros serviços de hospedagem Git.

Obrigado pela atenção!

Dúvidas?

Realização:

Patrocínios: