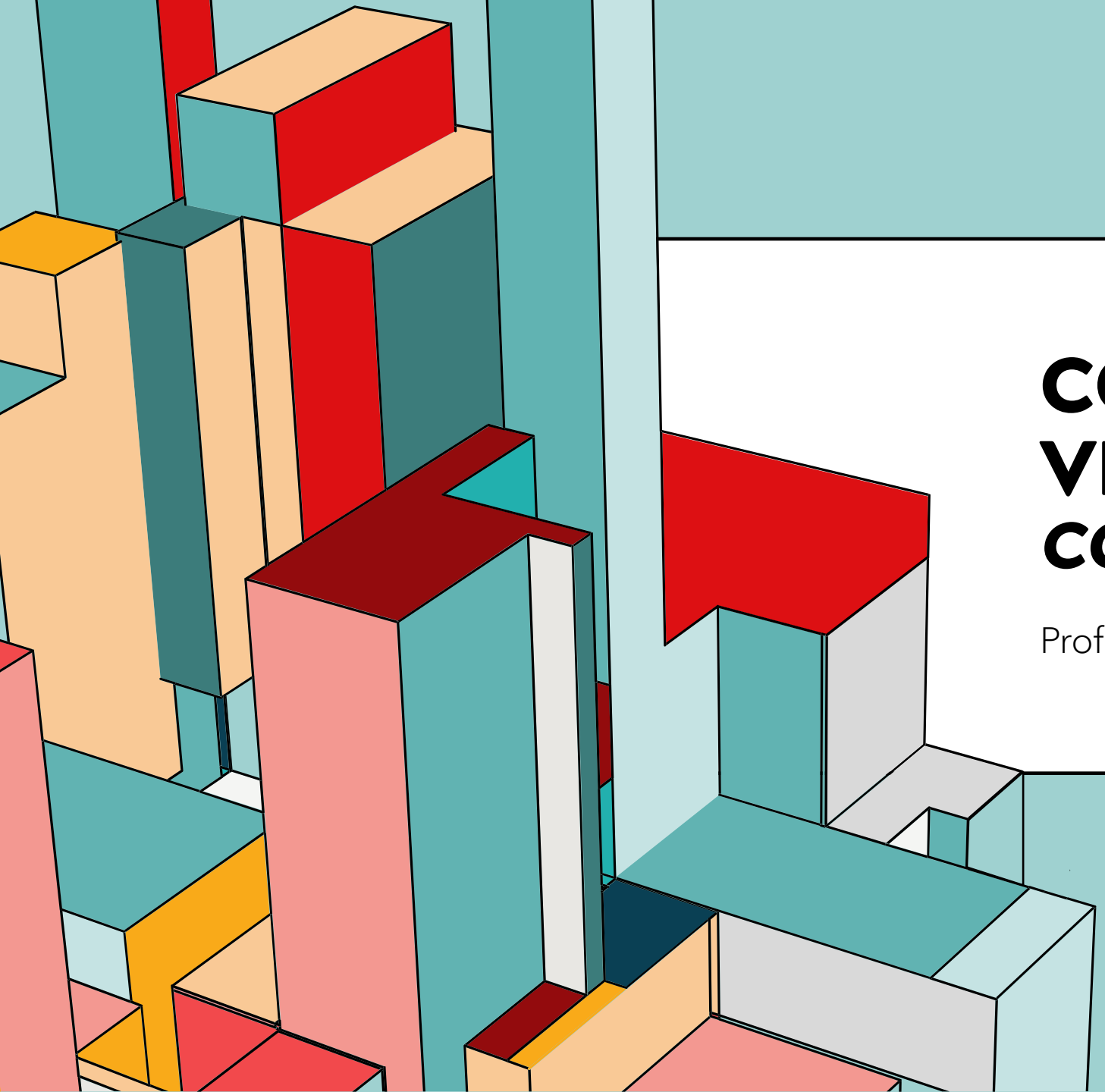


CONTROLE DE VERSÃO (GIT)
COMANDOS BÁSICOS

Prof. Dr. Dilermando Piva Jr

BANCHES ... O QUE SÃO?

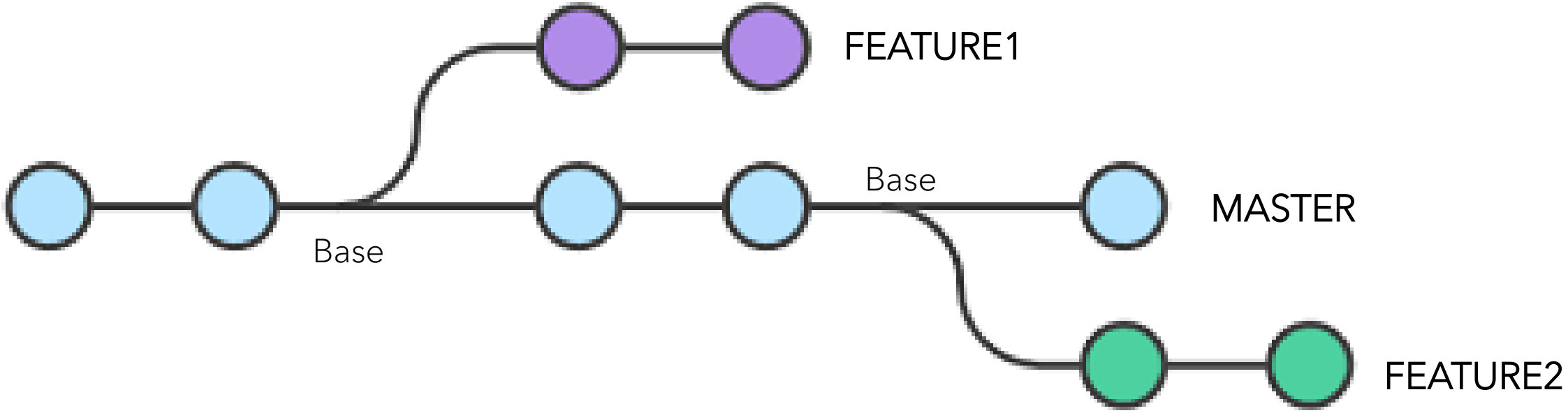
Um branch no Git é simplesmente um ponteiro móvel para um desses commits.

O nome do branch padrão no Git é **master** .

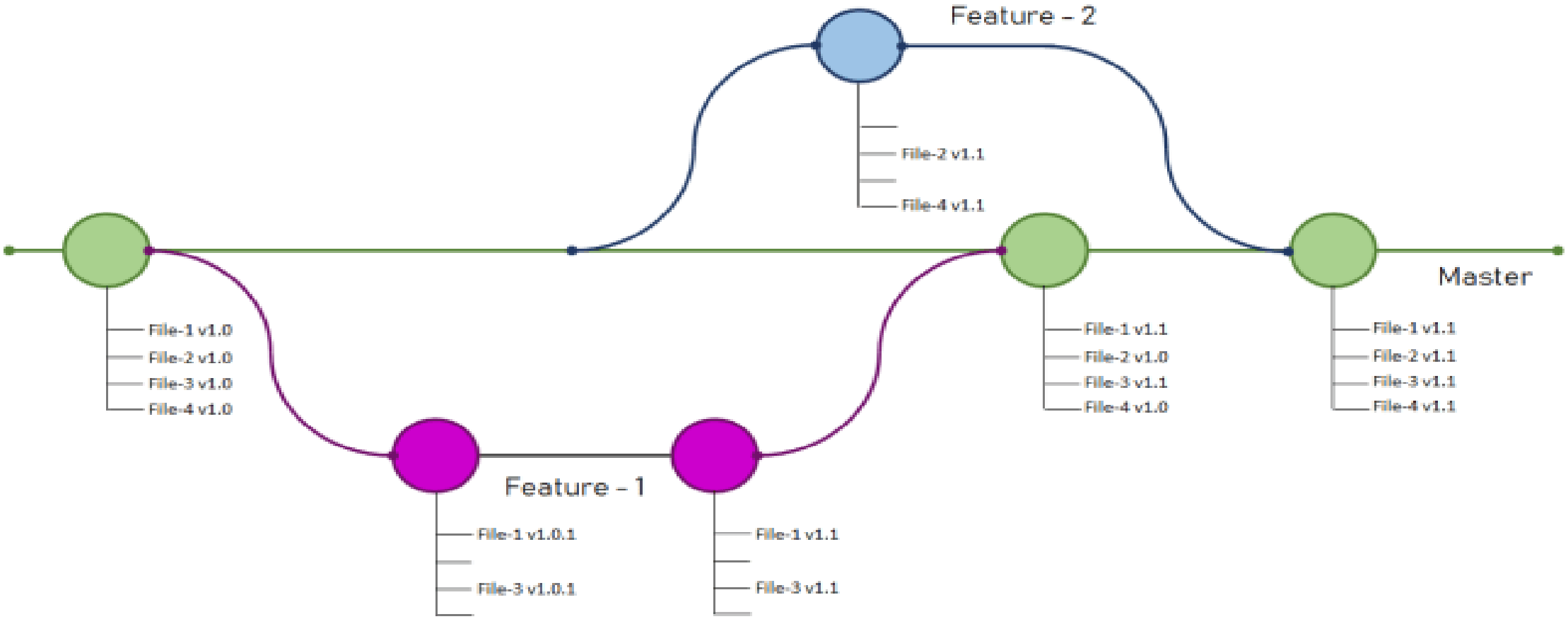
Conforme você começa a fazer commits, você recebe um branch master que aponta para o último commit que você fez.

Cada vez que você faz um novo commit, ele avança automaticamente..

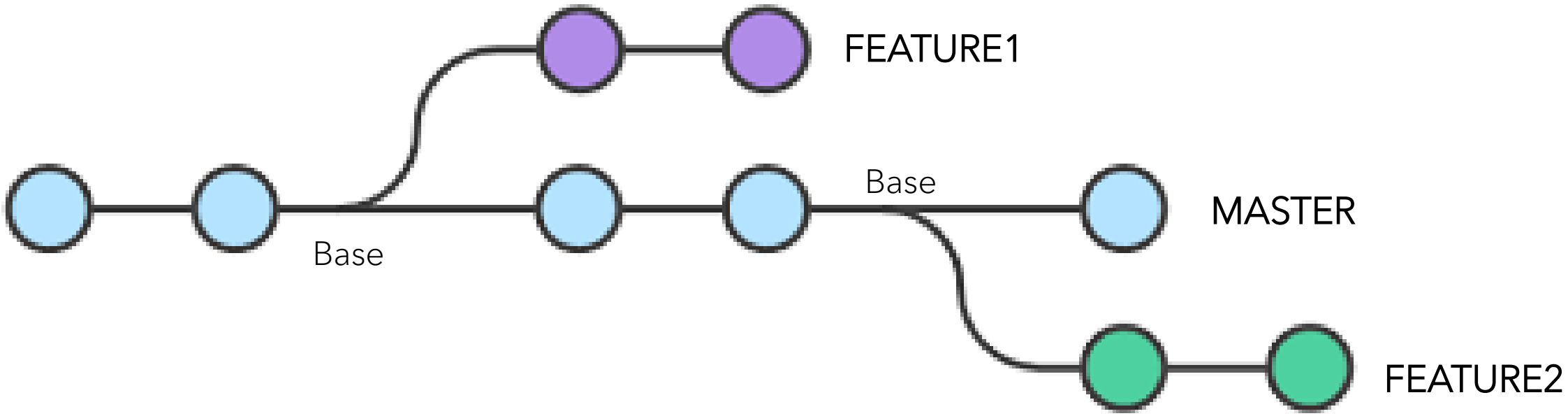
BRANCHES



BRANCHES


Feature – 2
File-2 v1.1
File-4 v1.1
Master
File-1 v1.0
File-2 v1.0
File-3 v1.0
File-4 v1.0
File-1 v1.1
File-2 v1.0
File-3 v1.1
File-4 v1.0
File-1 v1.1
File-2 v1.1
File-3 v1.1
File-4 v1.1
Feature – 1
File-1 v1.0.1
File-3 v1.0.1
File-1 v1.1
File-3 v1.1

BRANCHES



BRANCHES

Vamos verificar quais os branches existentes em nosso projeto:

git branch
* master

Ele retornará que só existe o Branch master. E o asterisco significa que é o Branch em que nós estamos trabalhando no momento.

git status
No ramo master
....

BRANCHES

Para criar um novo Branch:

git branch feature1 ((feature1 é o nome da branch))

O nome tem que ser o mais sugestivo possível. Pois se a branch não tiver um nome que o associe ao que você estará fazendo vai ficar difícil gerenciar.

```
git branch
  feature1
* master
```

BRANCHES

E para mudar para a branch criada ou qualquer outra que exista?

```
git checkout feature1
```

((muda para a branch nomeada))

Agora se verificarmos as branchs...

```
git branch
 * feature1
   master
```

BRANCHES

Para retornar a branch principal (master) basta utilizar o mesmo comando...

```
git checkout master
```

((muda para a branch nomeada))

Agora se verificarmos as branchs...

```
git branch
    feature1
  * master
```

BRANCHES

Vamos imaginar que você criou um branch para testar uma determinada funcionalidade e quando você terminou os testes, não teria mas razão desse branch existir. Você pode então deletá-lo

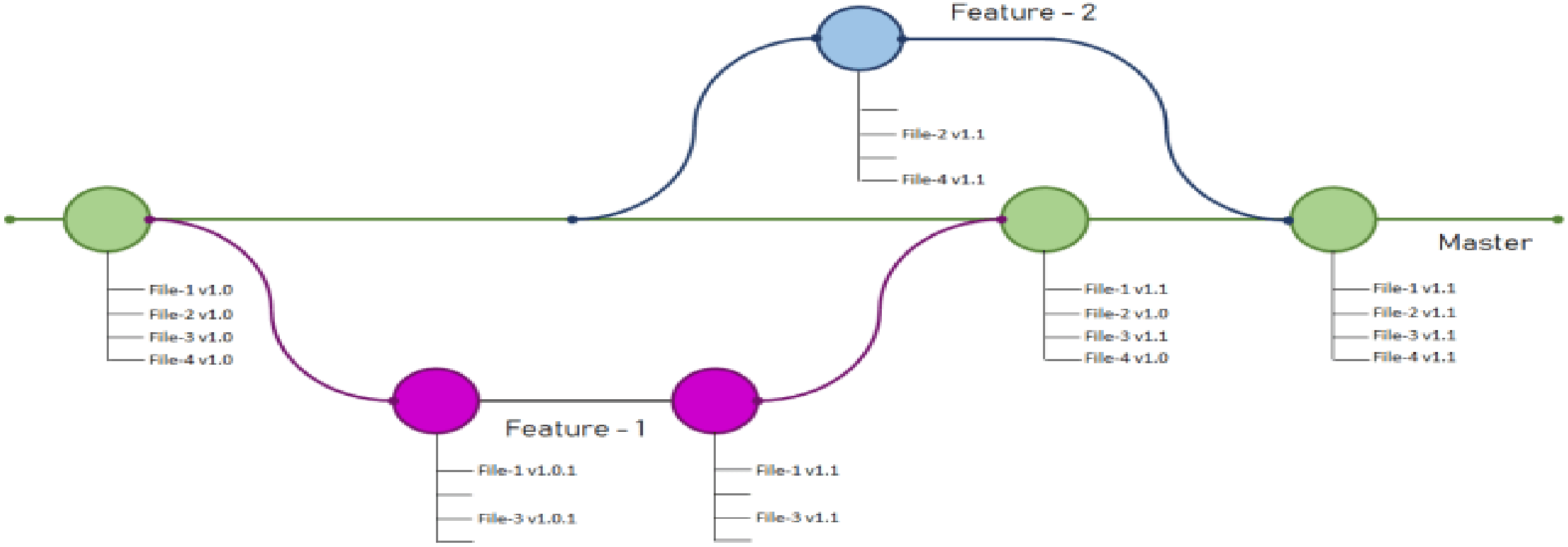
git branch –D feature1 ((deleta a branch nomeada sem merge))

Agora se verificarmos as branchs...

```
git branch
* master
```

BRANCHES

Depois que implementamos, testamos e aprovamos a funcionalidade, podemos fazer com que ela faça parte do projeto central (vá para a master).



MERGE

Para juntar um branch na master, basta ir para a master:

```
git branch master
```

E então fazer o merge...

```
git merge feature1
```

Se olhar o log, verá que esse merge, realizou o commit na master.

```
git log                (HEAD → master, feature1)
```

MERGE

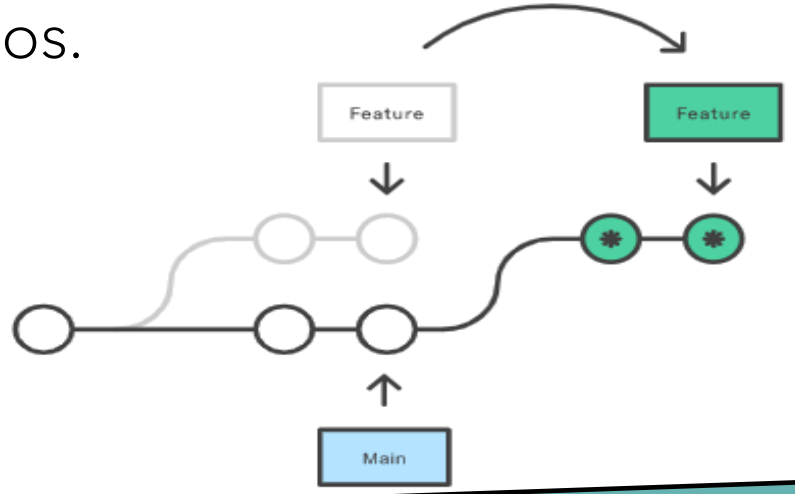
Depois que realizamos o marge na master, e não precisamos mais do branch que estávamos trabalhando, basta deletá-lo:

```
git branch –d feature1
```

Veja que agora podemos passar o d (minúsculo).

REBASE

A partir da perspectiva de conteúdo, o rebase é o processo de alterar a base da ramificação do commit para outra, fazendo parecer como se você criou a ramificação a partir de um commit diferente.
De um jeito intrínseco, o Git realiza isso criando novos commits e aplicando-os à base especificada.
É muito importante entender que, mesmo que a ramificação pareça a mesma, ela é composta de commits novos completos.



REBASE

O principal motivo para fazer o rebase é **manter um histórico de projeto linear**.

Por exemplo, considere uma situação em que a ramificação principal progrediu desde que você começou a trabalhar em uma ramificação de recurso. Você quer colocar as atualizações mais recentes da ramificação principal na ramificação de recurso, mas também quer o histórico da ramificação limpo, para que pareça que você esteve trabalhando com a ramificação principal mais recente.

Assim você tem o benefício posterior do merge limpo da ramificação de recurso de volta para a ramificação principal.

REBASE

Por que é interessante manter um "histórico limpo"? Os benefícios de ter um histórico limpo ficam tangíveis ao realizar operações do Git para investigar a introdução de uma regressão. Um cenário mais realista seria:

1. Um bug foi identificado na ramificação principal. Um recurso que estava funcionando bem agora está com falha.
2. Um desenvolvedor examina o histórico da ramificação principal usando **git log**. Por causa do "histórico limpo", o desenvolvedor é capaz de analisar rápido o histórico do projeto.
3. O desenvolvedor agora não consegue identificar quando o bug foi introduzido usando **git log**, então, ele executa **git bisect**.
4. Como o histórico do git está limpo, o **git bisect** tem um conjunto refinado de commits para comparar ao observar a regressão. O desenvolvedor encontra logo o commit que introduziu o bug e consegue agir de acordo.

REBASE

Vamos criar um novo branch (para criar e já mudar para esse branch utilize):

git checkout –b feature1

Faça a criação de um arquivoA.py

Volte para o master

git rebase feature1

Ele vai fazer a mesma coisa, nesse caso, que o MERGE.

CLONE

O git clone é usado sobretudo para apontar para um repositório existente e fazer um clone ou cópia deste repositório no novo diretório, em outro local.

O repositório original pode estar localizado no sistema de arquivos local ou em protocolos com suporte a acesso por máquinas remotas.

Crie em outro local um subdiretório. Entre nesse subdiretório. Digite:

git clone <caminho completo> . (note que tem um espaço e um ponto – clonar aqui)

Todos os arquivos do repositório anterior, serão transferidos para essa nova pasta.

CLONE

Depois da clonagem, verifique:

git status

Your banch is up to date with ‘origin/master’.

Depois que você fizer essa clonagem, você precisa fazer a configuração:

git config user.name “novo usuário”
git config user.email “novo email”

CLONE

- Crie um novo arquivo neste novo repositório.
- Adicione o arquivo (git add .)
- Faça o commit

Digite agora:

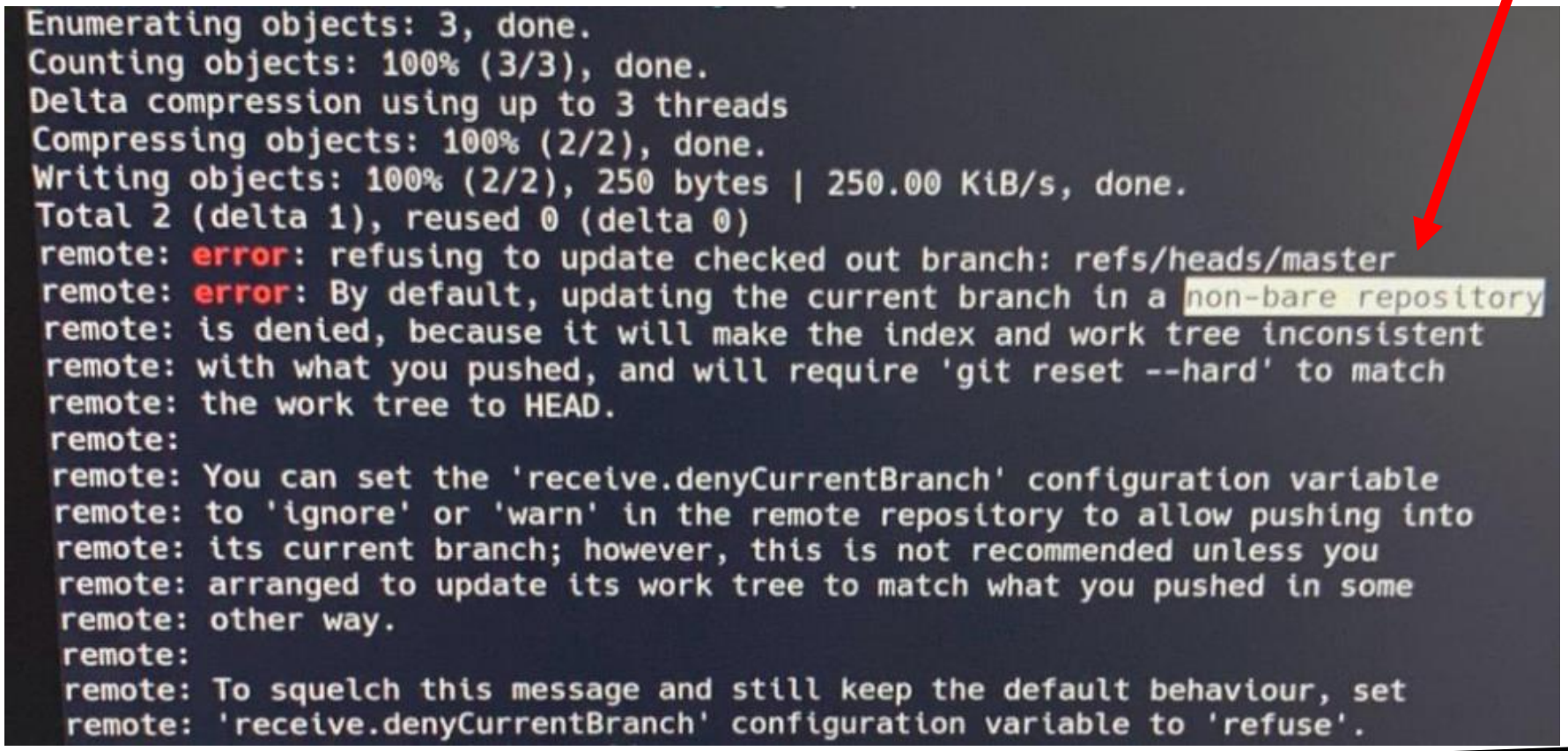
git status

Seu ramo está a frente de 'origin/master'. por 1 submissão
(use "git push" to publish your local commits)

PUSH

Se tentarmos fazer o push agora, ocorrerá um erro...

git push



FETCH

git fetch é um comando básico usado para baixar conteúdos de um repositório remoto.

O git fetch é usado em conjunto com git remote , git branch , git checkout e git reset para atualizar um repositório local ao estado de um remoto.

FETCH

- Volte para o repositório inicial.
- Crie um novo arquivo
- Adicione e faça o commit.

Volte ao repositório clonado.

`git fetch`

Ele baixa os arquivos sem realizar o merge. Para atualizar e ver o novo arquivo, faça o rebase:

`git rebase`

PULL

Simplesmente é a junção do fech + rebase

Volte ao projeto base
Crie um novo arquivo, adicione e faça o commit.

Volte ao projeto clone.

`git pull`

Ele vai abrir um editor de texto, para você digitar o que está ocorrendo.
Digite crtl+O e crtl+X

BARE REPOSITORY

Se tentarmos fazer o push agora, ocorrerá um erro...

Tornar o repositório central

git push

```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 3 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 250 bytes | 250.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: error: refusing to update checked out branch: refs/heads/master
remote: error: By default, updating the current branch in a non-bare repository
remote: is denied, because it will make the index and work tree inconsistent
remote: with what you pushed, and will require 'git reset --hard' to match
remote: the work tree to HEAD.
remote:
remote: You can set the 'receive.denyCurrentBranch' configuration variable
remote: to 'ignore' or 'warn' in the remote repository to allow pushing into
remote: its current branch; however, this is not recommended unless you
remote: arranged to update its work tree to match what you pushed in some
remote: other way.
remote:
remote: To squelch this message and still keep the default behaviour, set
remote: 'receive.denyCurrentBranch' configuration variable to 'refuse'.
```

BARE REPOSITORY

Vamos criar um novo subdiretório ao lado dos dois primeiros criados.
Vamos dar o nome de <projeto>-bare
Vamos entrar nesse subdiretório e vamos inicializar com o seguinte comando:

git init --bare

Veja que ele criou uma série de subdiretórios dentre desse novo projeto.

Ele vai servir como projeto central.

BARE REPOSITORY

Vamos criar um novo subdiretório ao lado desses três primeiros criados.
Vamos dar o nome de <nomeA>
Vamos entrar nesse subdiretório e vamos criar um novo subdiretório: <projeto>
Vamos entrar nesse subdiretório e vamos clonar o <projeto-bare>

git clone <path completo> .

Agora temos o projeto bare clonado aqui.

BARE REPOSITORY

Vamos criar um novo subdiretório ao lado desses três primeiros criados.
Vamos dar o nome de <nomeA>
Vamos entrar nesse subdiretório e vamos criar um novo subdiretório: <projeto>
Vamos entrar nesse subdiretório e vamos clonar o <projeto-bare>

git clone <path completo> .

Agora temos o projeto bare clonado aqui.

BARE REPOSITORY

Vamos criar um novo arquivo no novo subdiretório.
Adicionar e faze o commit

Se fizer o git status, vai verificar que o repositório local está a frente do origin/master.

Agora sim, podemos fazer o push:

git push

Agora ele atualizou o projeto original.

TAGS

git tag é em geral usado para capturar um ponto no histórico que é usado para uma versão marcada (por exemplo, v1. 0.1).

Um marcador é como uma ramificação que não muda.

Diferente das ramificações, os marcadores depois de criados não têm mais histórico de commits.

git tag v1.01

Agora quando você fizer o commit e o push, ele registrará no branch principal essa tag.

TAGS

Essa tag só é conhecida no diretório que foi criada.
Para submeter ao projeto origem....

git push origin v1.01

Agora a nova tag está disponível a todos.

Nos novos usuários agora podem atualizar essa nova tag, com:

git pull

Nota... Quando fazemos isso, não existe o commit dessa tag ela apenas fica disponível.

TAREFAS...

- ...

OBRIGADO

Prof. Dr. Dilermando Piva Jr.

https://piva.pro.br

piva.jr@fatec.sp.gov.br

https://pypro.com.br