Desenvolvimento WEB

-

Full Stack Completo: Java + React

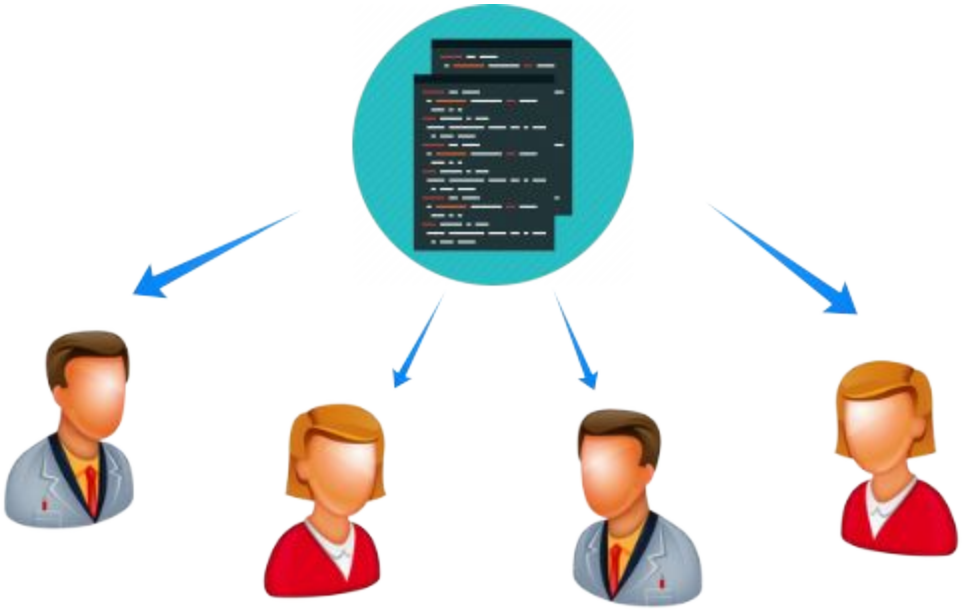
Prof. Roberto Alencar

Controle e Versionamento do Projeto com

Controle de Versão

Motivação

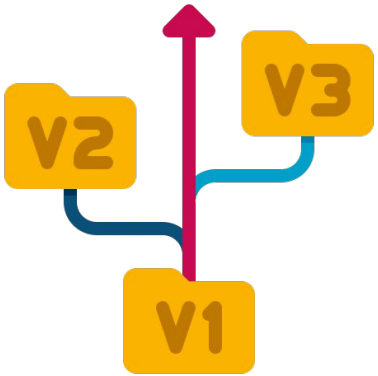
Desenvolvimento distribuído



Controle de Versão

Definição

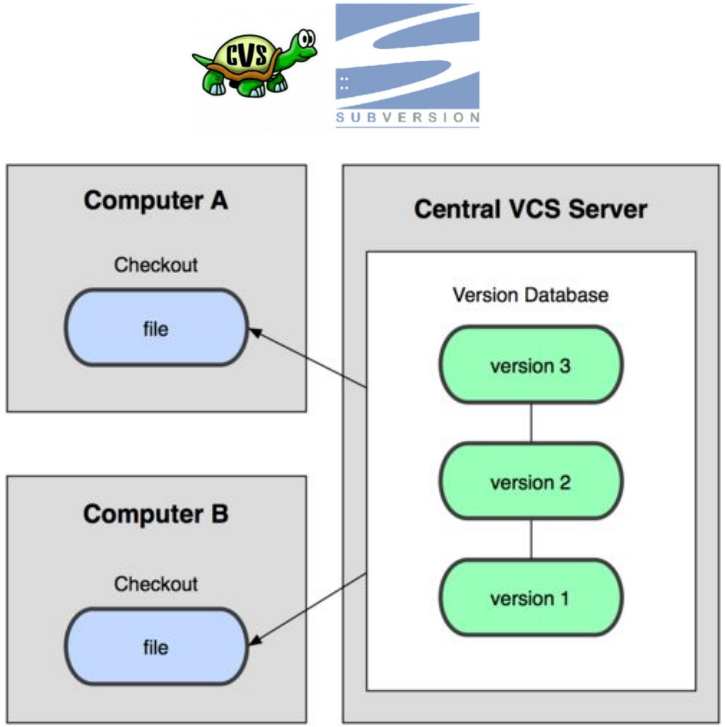
“O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo de forma que você possa recuperar versões específicas”

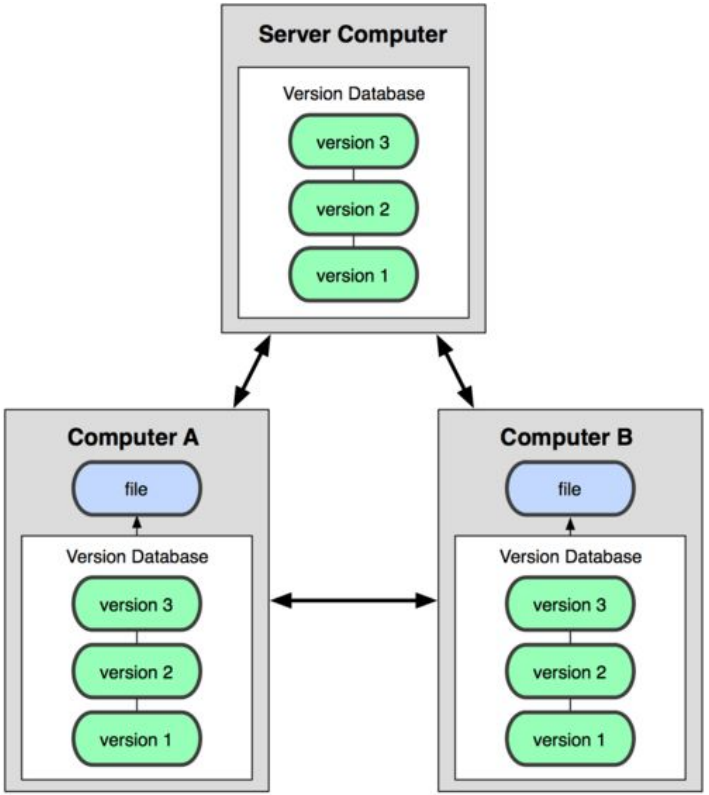


Git

Controle de Versão

Versionamento Central vs Distribuído





Controle de Versão

Sobre o Git

- Velocidade;
- Design simples;
- Suporte robusto a desenvolvimento não linear (milhares de branches paralelos);
- Totalmente distribuído;
- Capaz de lidar eficientemente com grandes projetos como o kernel do Linux (velocidade e volume de dados).

Controle de Versão

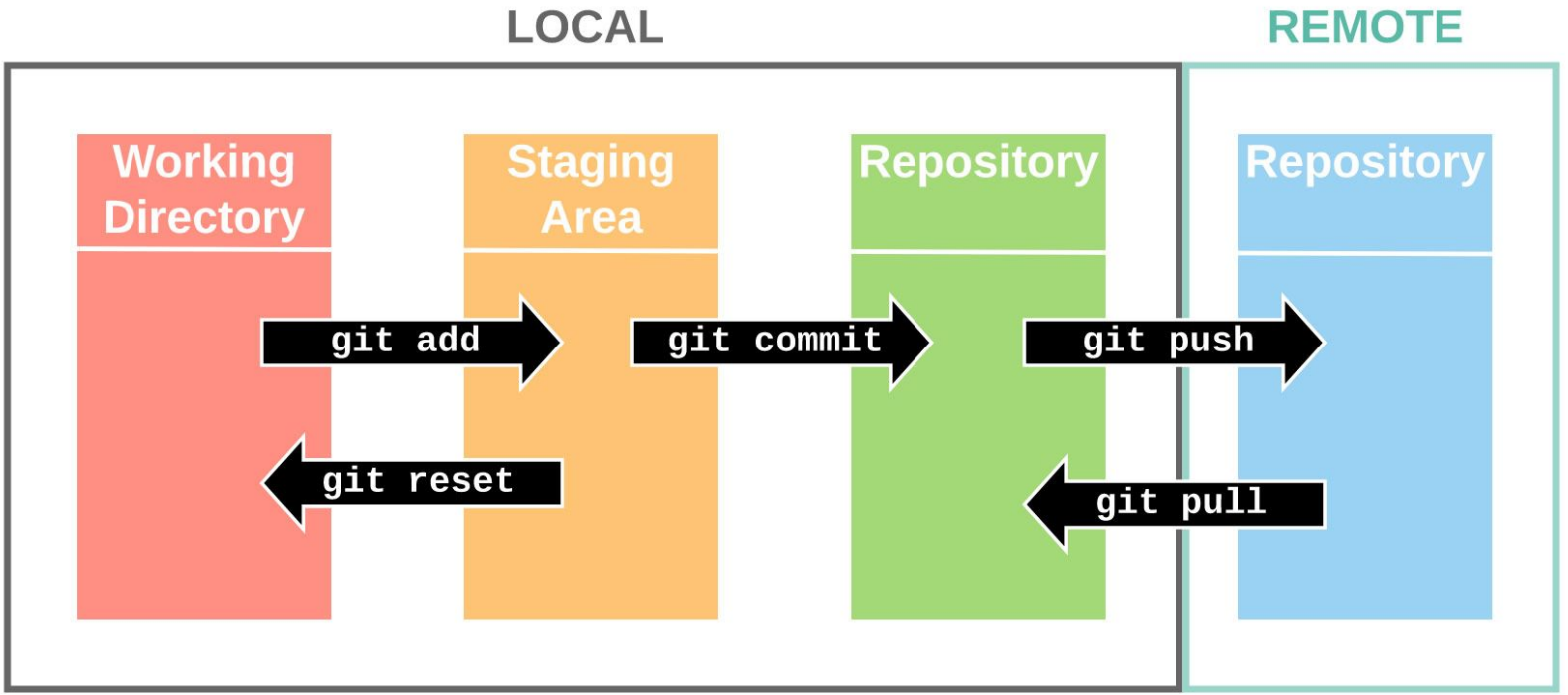
Git :: Comandos Essenciais

- Cria um novo repositório Git em um diretório da sua máquina:
  - `git init`

- Clona um repositório Git de um endereço remoto para um diretório na sua máquina:
  - `git clone [url]`

Controle de Versão

Git :: Comandos Essenciais



Controle de Versão

Git :: Comandos Essenciais

- O comando **`add`** adiciona os arquivos solicitados ao ambiente de `stage`. É uma forma de dizer para o `git` que você deseja que as modificações daquele(s) arquivo(s) sejam marcadas para serem enviadas na próxima remessa ao repositório (`push`):
  - `git add .` *-> adiciona todos os arquivos modificados*
  - `git add *.java` *-> adiciona todos os arquivos de uma determinada extensão que foram modificados*
  - `git add arquivo.txt` *-> adiciona um determinado arquivo modificado ao stage*

- O comando **`commit`** grava as modificações, desta forma é criado um snapshot do estado atual do projeto:
  - `git commit -m "versão inicial do projeto"`

Controle de Versão

Git :: Comandos Essenciais

- O comando `remote` lista todos os `alias` e os respectivos repositórios remotos:
  - `git remote -v`

- Após criar um repositório git local, é possível também vincular este a um repositório remoto através do comando `demote add`:
  - `git remote add [alias_repositorio] [url]`

Controle de Versão

Git :: Comandos Essenciais

- Para **atualizar o repositório local** com o código do repositório remoto, utilize o comando **`pull`**:
  - `git pull [alias_repositorio] [branch]`

- Para **enviar o código** do repositório local **para o repositório remoto**, utilize o comando **`push`**:
  - `git push [alias_repositorio] [branch]`

Controle de Versão

Git :: Comandos Essenciais

- Atualiza o repositório local baixando as branches e tags novas do repositório remoto:
  - `git pull`
- Verifica em qual `branch` você está, se há algum arquivo modificado e não enviado para o `stage` ou se há arquivos no `stage` e não enviados para o servidor remoto:
  - `git status`
- Desfaz as alterações realizadas em algum arquivo para a versão anterior deste arquivo:
  - `git restore arquivo_modificado.css`

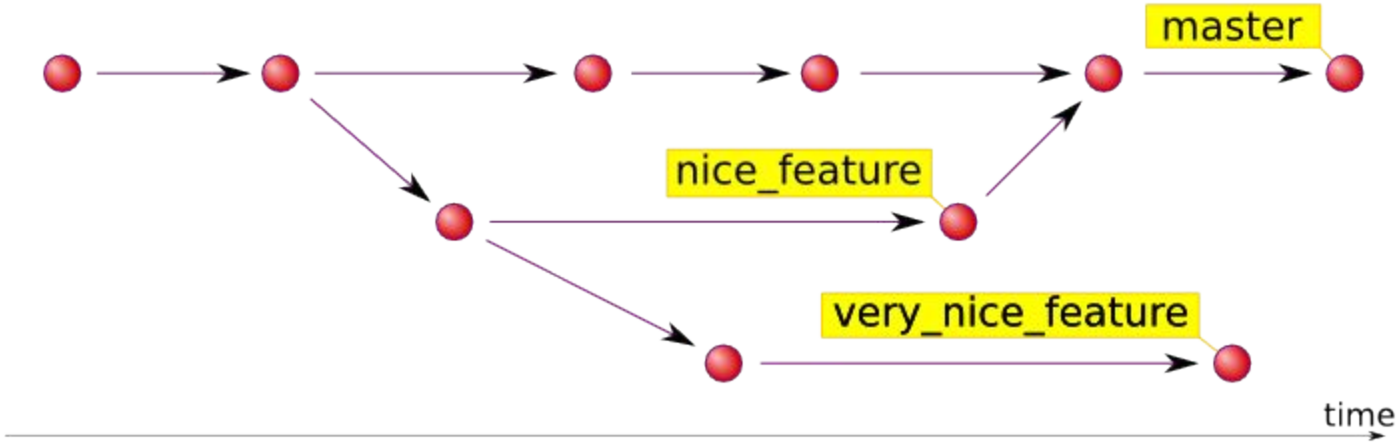
Controle de Versão

Git :: Comandos Essenciais

- Lista as tags de um repositório:
  - `git tag`
- Cria uma nova tag:
  - `git tag -a v1.4 -m “minha versão 1.4”`
- Envia a tag para o repositório remoto:
  - `git push origin v1.4`

Controle de Versão

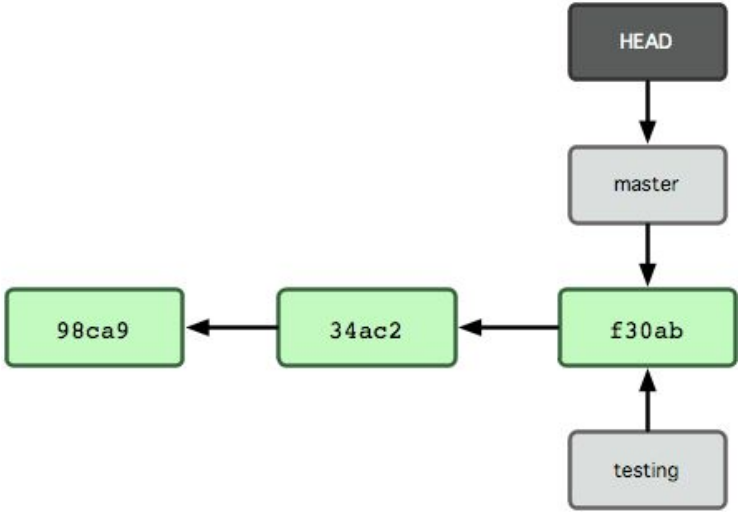
Git :: Criando Ramificações (*Branching*)

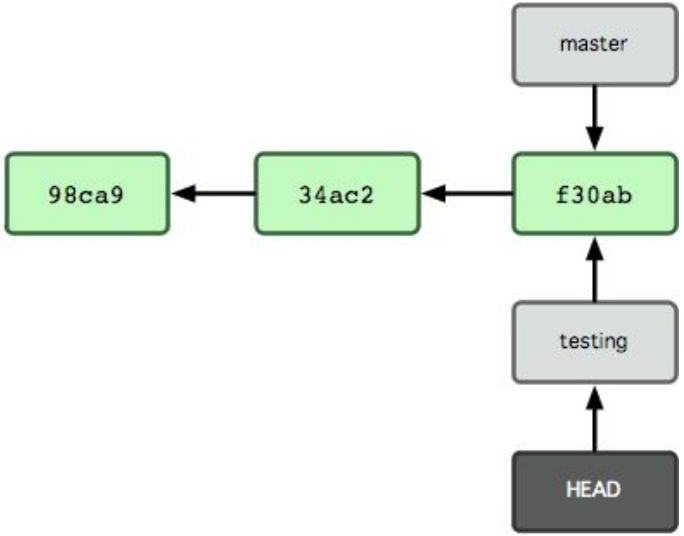


Controle de Versão

Git :: Criando Ramificações (*Branching*)

- `git branch testing` *-> cria uma nova branch chamada testing*
- `git checkout testing` *-> troca o código para a branch testing*

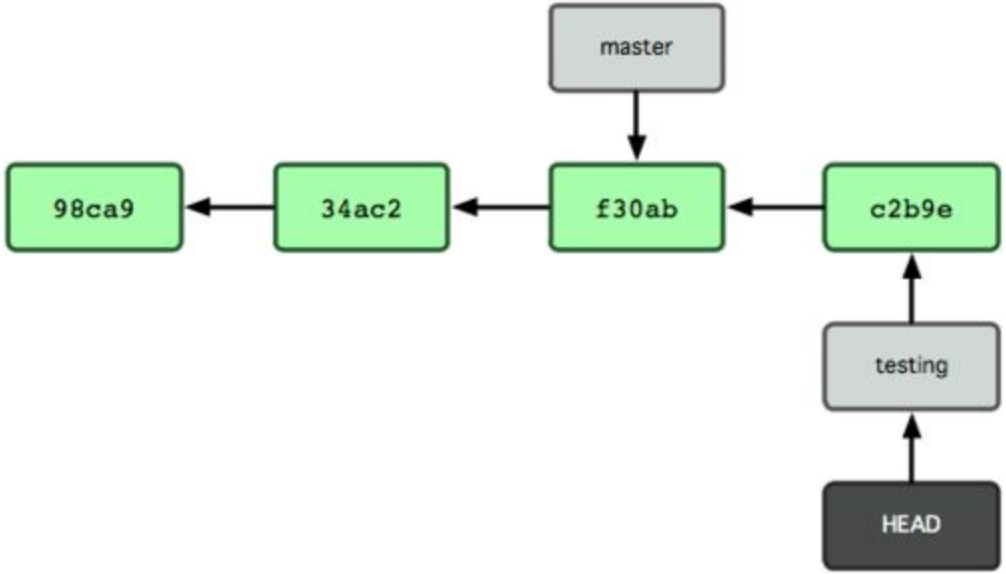




Controle de Versão

Git :: Criando Ramificações (*Branching*)

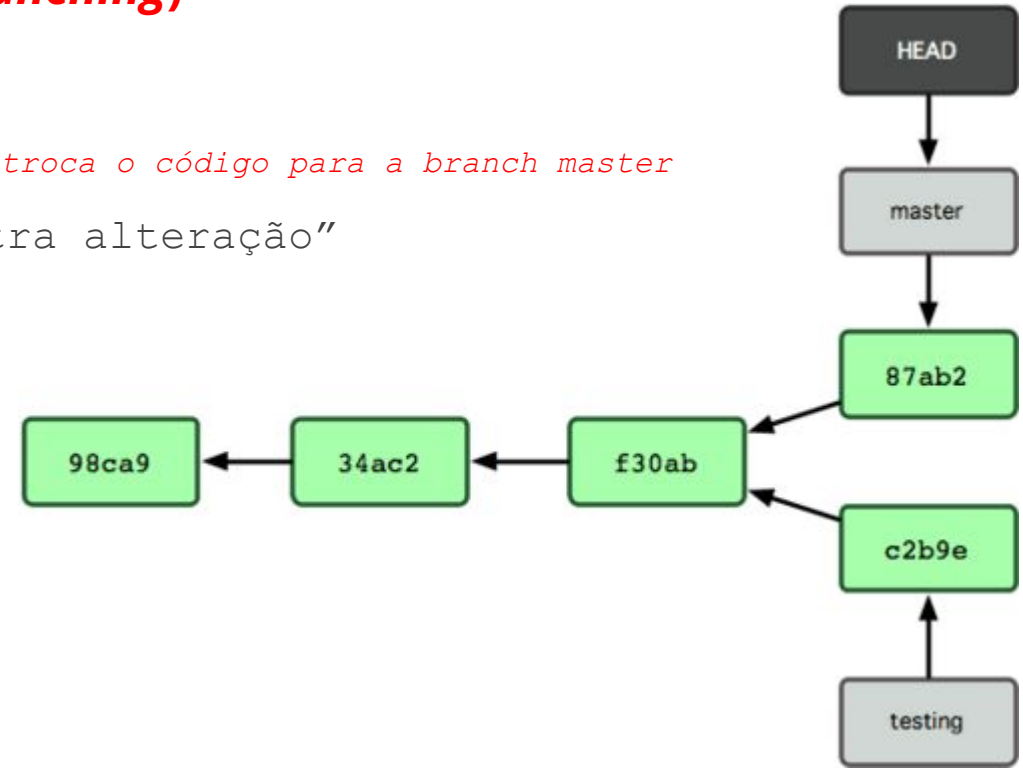
- `git commit -a -m "fiz uma alteração"`



Controle de Versão

Git :: Criando Ramificações (*Branching*)

- `git checkout master` *-> troca o código para a branch master*
- `git commit -a -m "fiz outra alteração"`



Controle de Versão

Git :: Exercício de Ramificações (*Branching*)

1) Cria uma nova branch chamada cadastro_cliente:

```
-> git branch cadastro_cliente
```

2) Entra na branch cadastro_cliente:

```
-> git checkout cadastro_cliente
```

3) Cria um novo arquivo .txt na branch cadastro_cliente

4) Da um commit e um push para enviar a branch cadastro_cliente para o github

```
-> git add .

-> git commit -m "codigo alterado na branch"

-> git push origin cadastro_cliente
```

...

Controle de Versão

Git :: Exercício de Ramificações (*Branching*)

...

5) Volta para a branch principal (main ou master)

```
-> git checkout master
```

6) Atualiza o código da branch principal com o código da branch cadastro_cliente

```
-> git pull origin cadastro_cliente
```

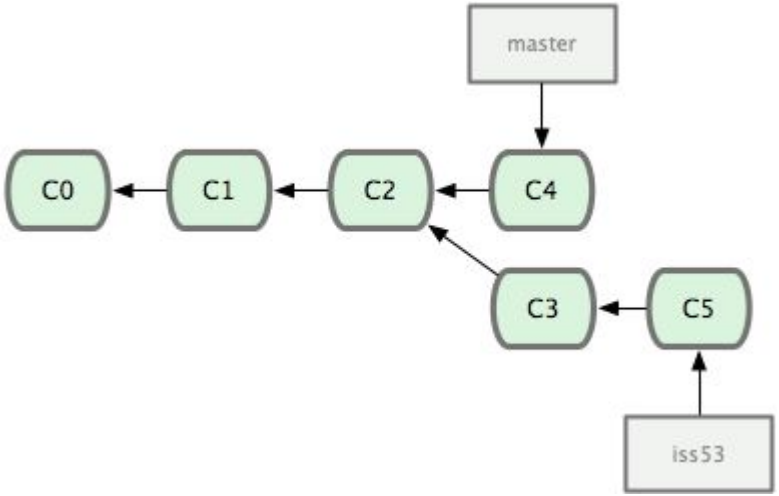
7) Atualiza o código da branch principal do repositório remoto:

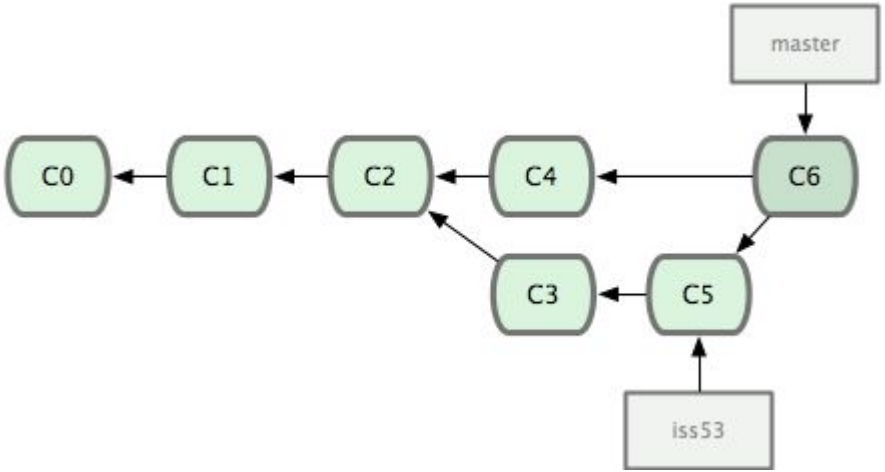
```
-> git push origin master
```

Controle de Versão

Git :: Merge

- `git checkout pull origin testing` *-> puxa o código da branch testing*





Controle de Versão

Git :: Merge

```
<<<<<<< HEAD:index.html
<div id="footer">contato : email.support@github.com</div>
=======
<div id="footer">
  por favor nos contate em support@github.com
</div>
>>>>>>> iss53:index.html
```

Controle de Versão

Git :: Merge

- Instale na sua máquina a ferramenta de merge chamada **Meld**

Controle de Versão

Git :: Exercício de Merge

1) Atualize o repositório local com o código do repositório remoto:

```
-> git pull origin master
```

2) No Github, altere o conteúdo de uma linha de algum arquivo (grave essas alterações no Github dando um commit no próprio Github)

3) No repositório local, altere o conteúdo do mesmo arquivo na mesma linha (grave essas alterações com um commit)

4) Atualize o repositório local, com o código do repositório remoto:

```
-> git pull origin master
```

5) Resolva o conflito com a ferramenta Meld

```
-> git mergetool
```

6) Envie as alterações para o repositório remoto:

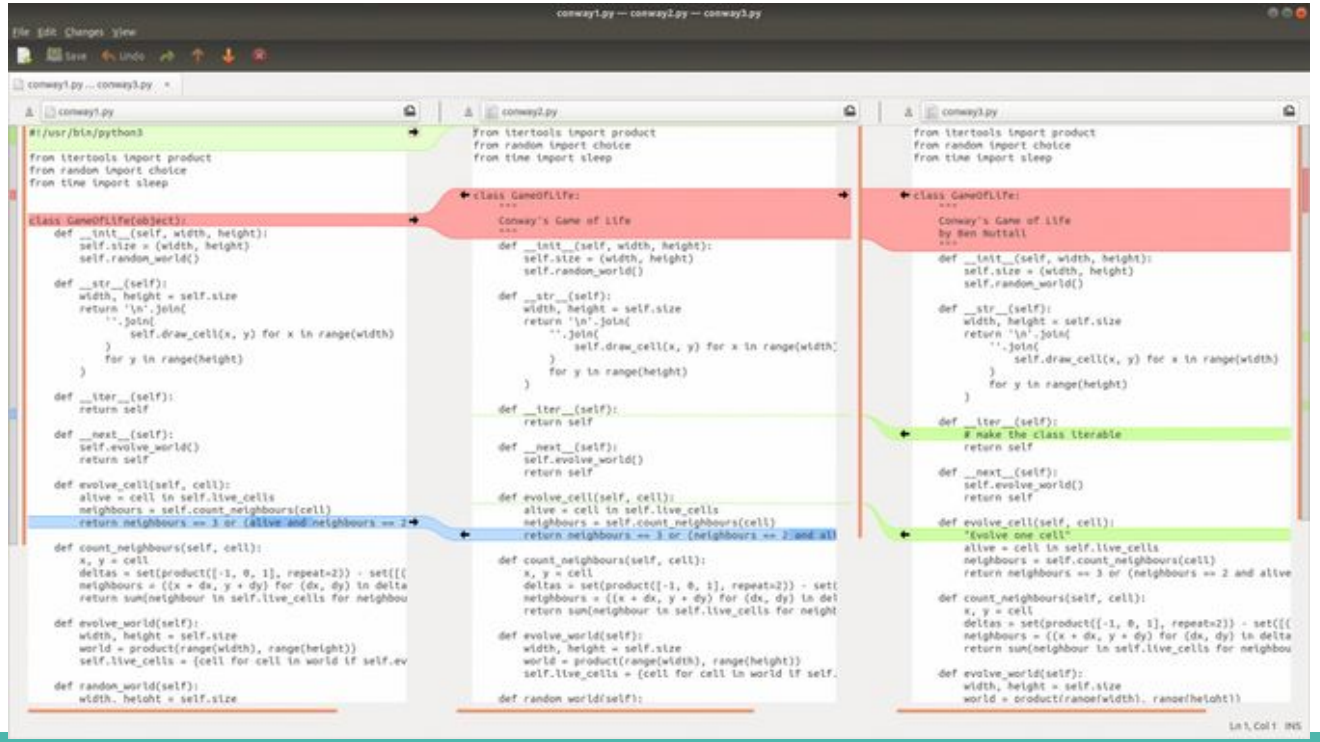
```
-> git push origin master
```

Controle de Versão

Git :: Merge

- `git mergetool`

Esse comando abre a ferramenta de merge para resolver os problemas de conflito de código.



Controle de Versão

Boas Práticas

- O código estável sempre deve estar na branch main (ou master)
  - *não se deve desenvolver na branch main (master)*

- Branch developer;
  - *criar uma branch de integração para testar as tarefas integradas antes de jogar o código para o master*

- Cada tarefa deve ser desenvolvida em uma branch própria para funcionalidade (*feature branch*);

Controle de Versão

Boas Práticas

- Pro Git Book
  - https://git-scm.com/book/pt-br/v2

- Try Git
  - https://try.github.io

Dúvidas

Obrigado !