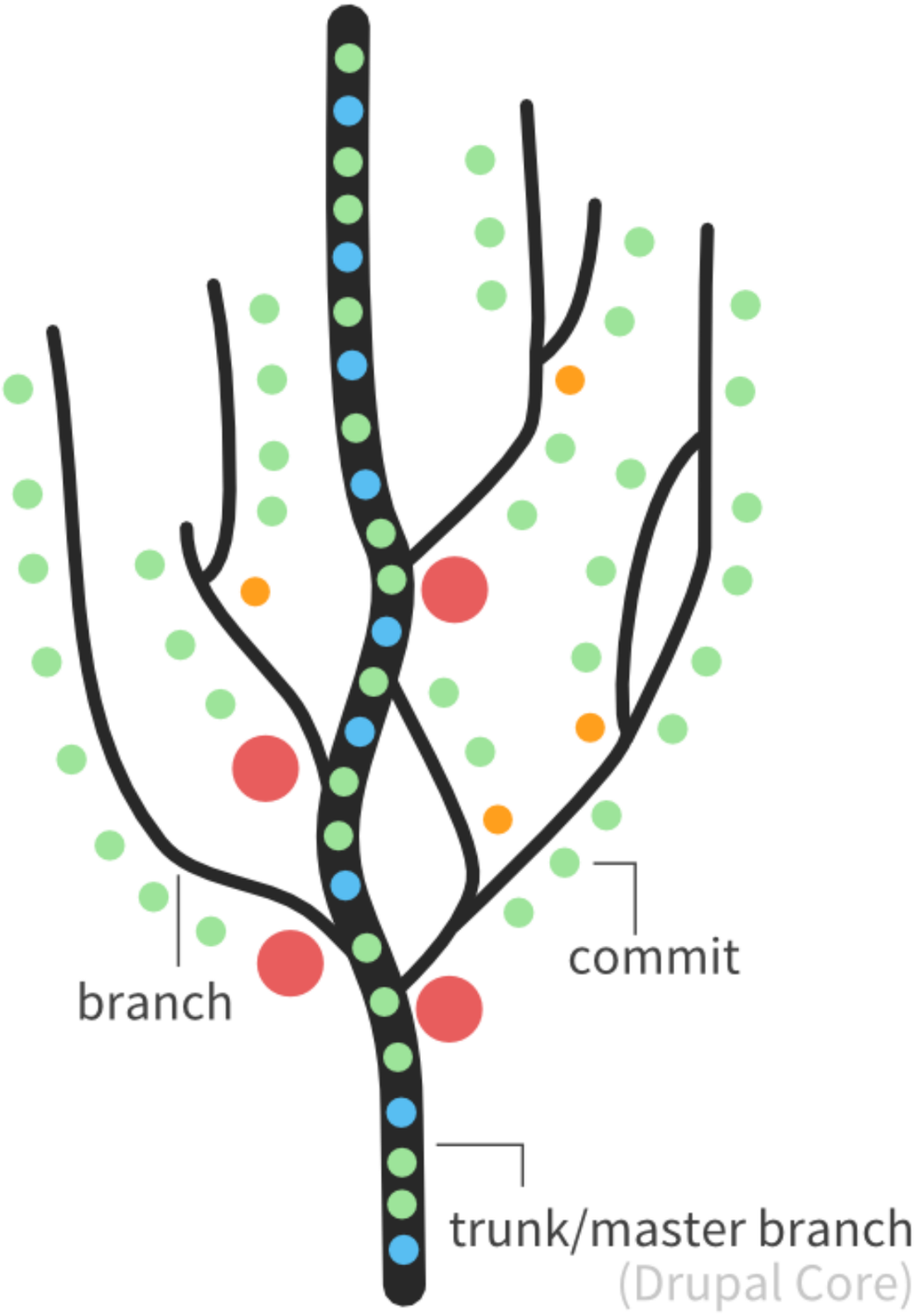
Faculdade IMPACTA TECNOLOGIA

Gestão de Mudanças

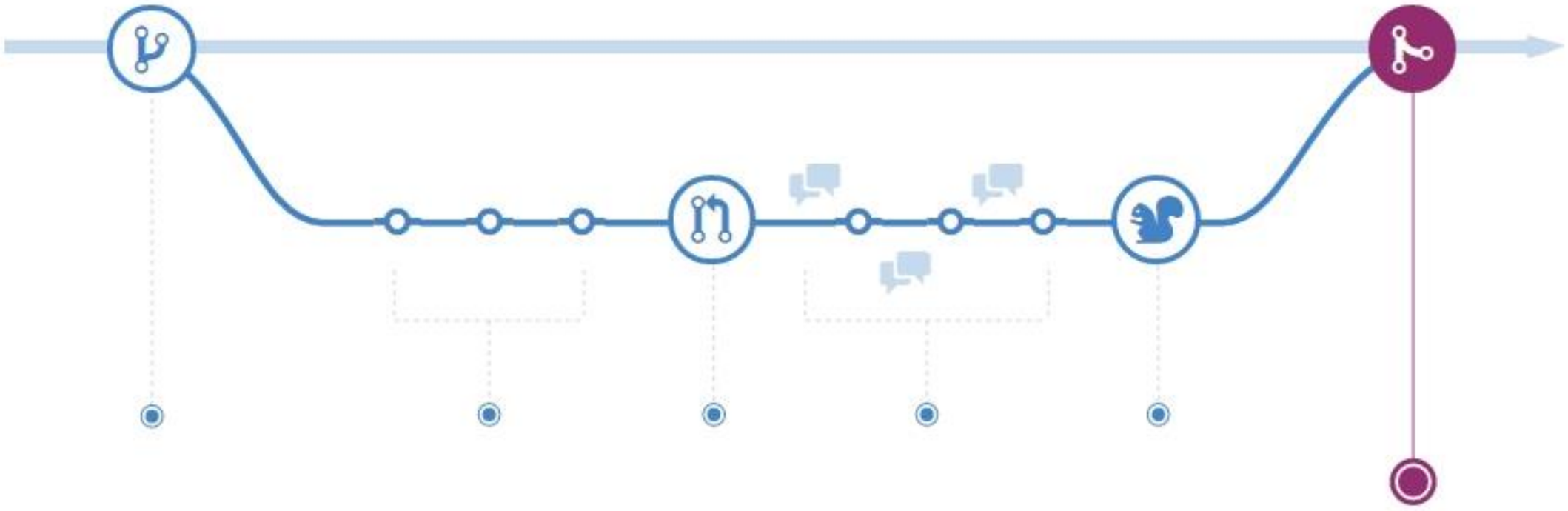
Aula 07 - Parte 1
Disciplina: Ambiente de Desenvolvimento e Operação
Curso: Sistemas da Informação
Turma: 2B
Prof. Edson Benites Silva
2º semestre de 2017

Árvore de versões

Fonte: http://www.drupal.org/node/991716

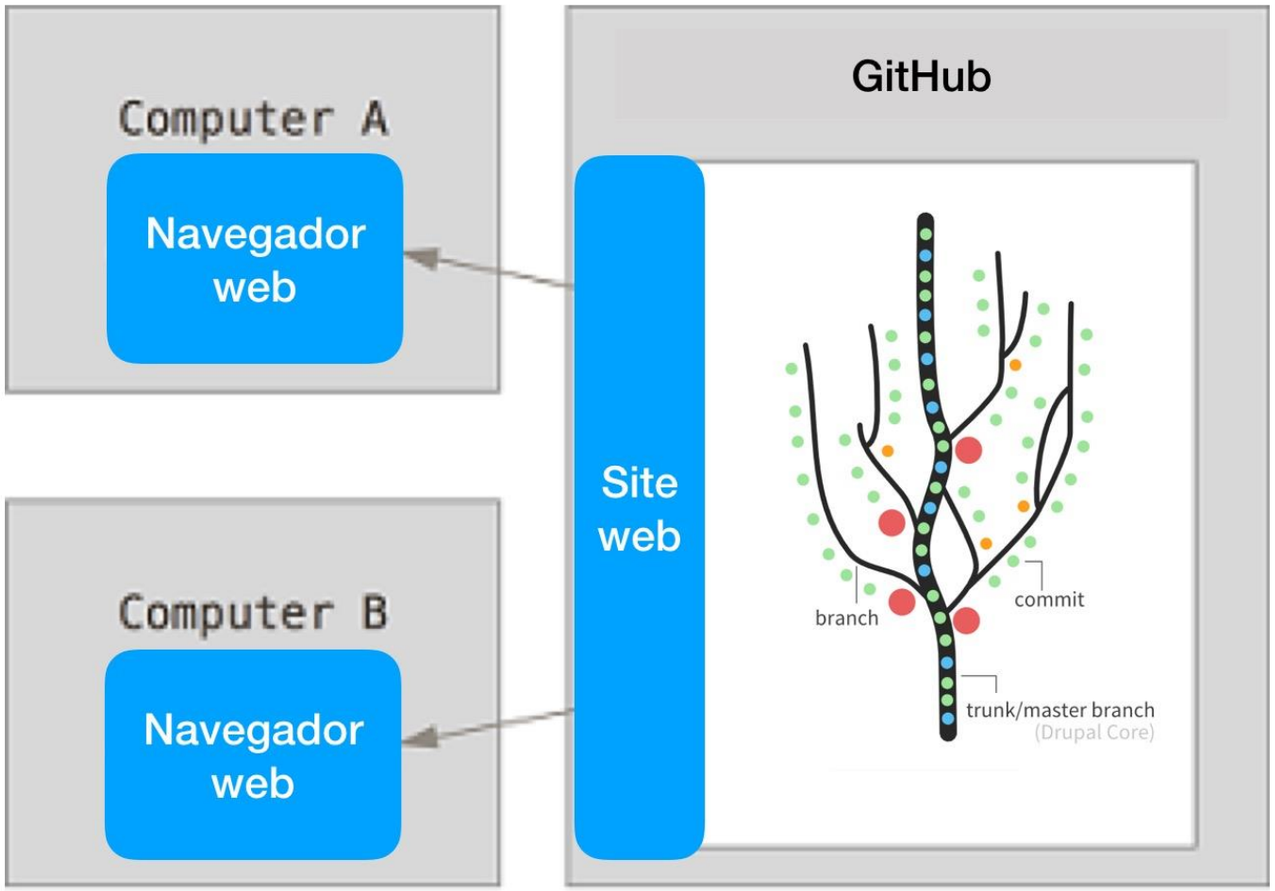


Workflow Github

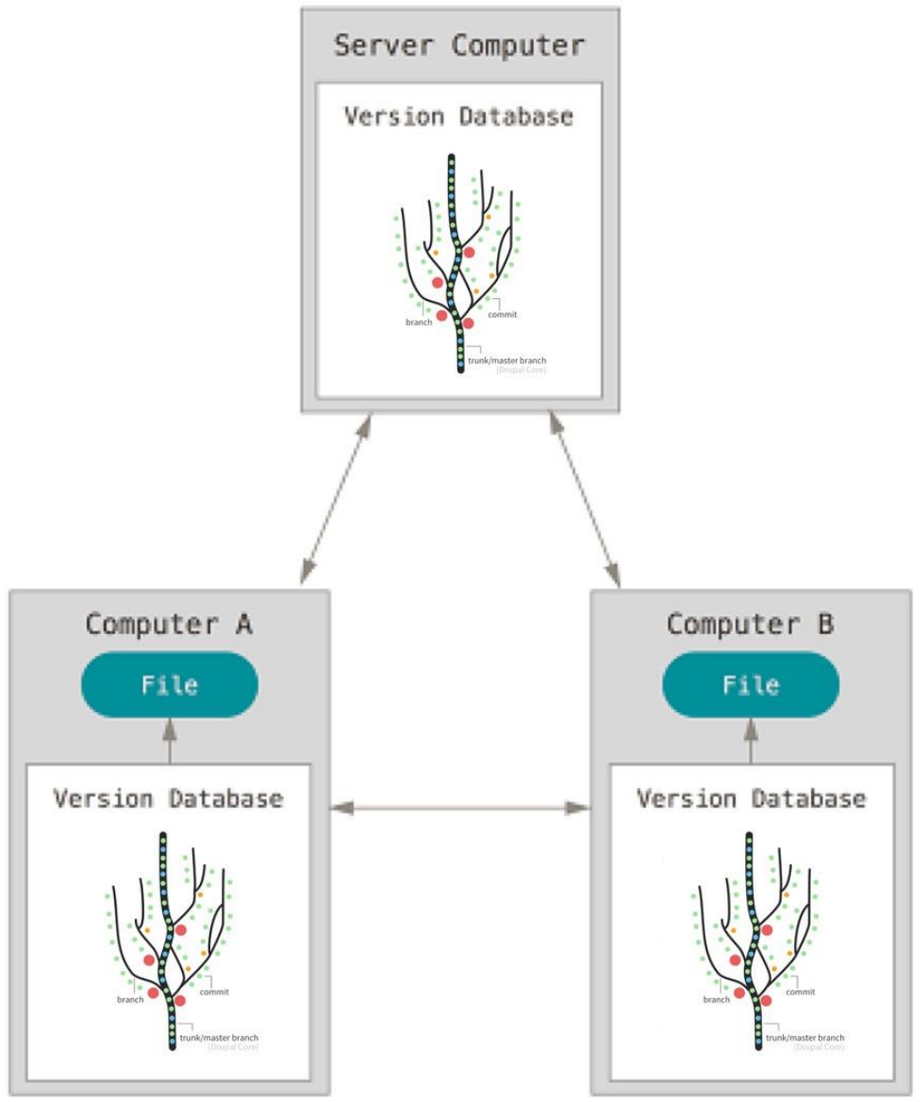


Fonte: https://guides.github.com/introduction/flow

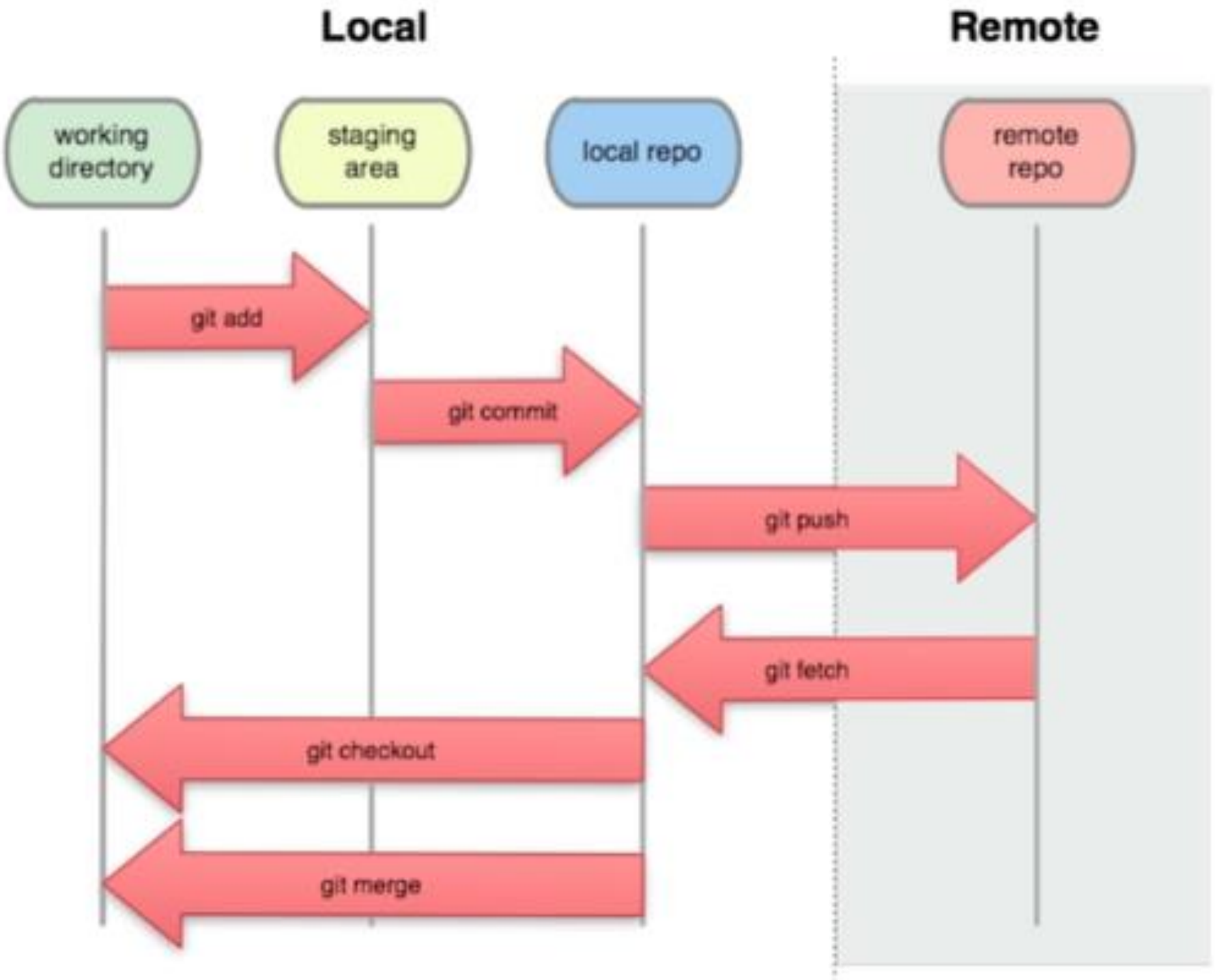
GitHub pela Web



Git distribuído



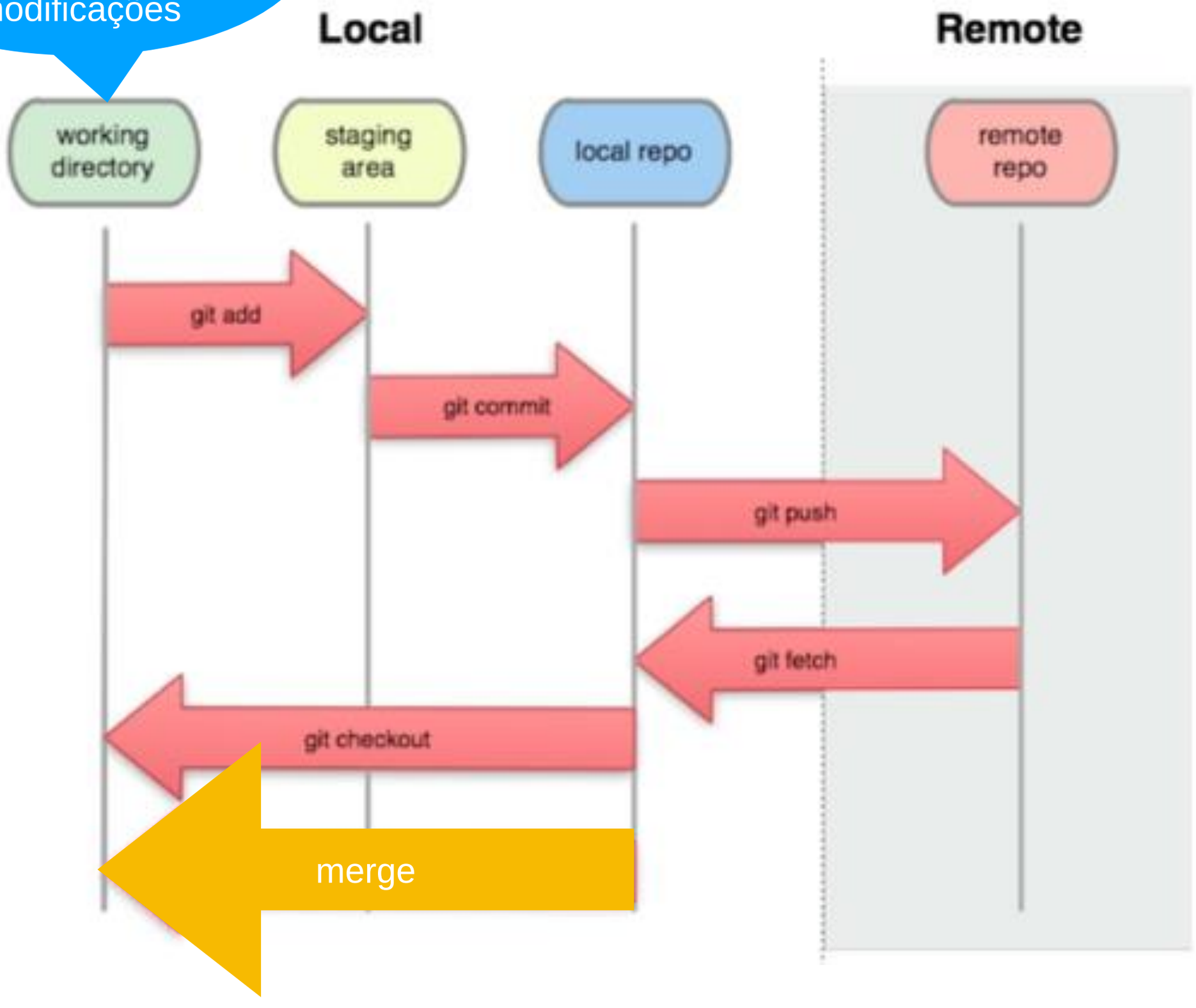
Operações nos repositórios Git



Fonte: https://greenido.wordpress.com/2013/07/22/git-101-useful-commands/

Merge

Deve estar com o *branch* que receberá as modificações



Mudança

Adição, modificação ou remoção de hardware, rede, software, aplicação, ambiente, sistema, computadores e documentação associada.

Gestão de mudanças

Define:

- Procedimentos para realizar modificações.
- Documentação necessária.
- Mecanismos para modificar de uma maneira controlada.
- Mecanismos de aprovação de mudanças.

Benefícios da gestão de mudanças

- Controle sobre o escopo do projeto.
- Tratamento das mudanças de forma coordenada.
- Redução dos problemas de comunicação na equipe.
- Maior qualidade pois cada mudança tem seu impacto avaliado antes de ser realizada.

Requisição de mudança

Item formal utilizado para registrar detalhes de um pedido de mudança de qualquer item de configuração ou de procedimentos associados à infra-estrutura.

Agenda de mudanças

Plano que contém detalhes das mudanças aprovadas e das datas propostas para sua realização.

Comitê consultivo de mudanças

- CCB (*Change Control Board*)
- Formado de acordo com as mudanças a serem discutidas.
- Exemplos de participantes:
  - Fornecedores.
  - Gerente de Problemas.
  - Gerente de Nível de Serviço.
  - Equipe de relações com o cliente.
  - etc.

Tipos de mudanças

Mudança padrão: mudança que é pré-autorizada pelo Gerenciamento de Mudanças e que se tornou rotineira, já tendo um script de procedimento para execução. Por este motivo, geralmente o fluxo para execução desta alteração é mais ágil.

Mudança normal: é uma mudança para a qual não existe um script já pronto, e precisa passar pelo fluxo mais extenso para ser autorizada e planejada antes de sua execução.

Mudança emergencial: aquela que precisa ser implementada rapidamente para resolver falhas (incidentes). Neste caso, nem sempre será possível realizar todos os testes. Este tipo de mudança é tratado pelo Comitê Consultivo de Mudanças Emergenciais.

Gerente de mudanças

- Recebe, registra e aloca a prioridade adequada para todas as mudanças.
- Rejeita as mudanças que julga serem improcedentes.
- Prepara a agenda de mudanças a ser discutida com o Comitê Consultivo de Mudanças.
- Convoca os participantes para as reuniões do Comitê Consultivo de Mudanças e divulga a agenda de mudanças que será discutida.
- Efetua a revisão após a implementação de todas as mudanças.
- Elabora relatórios.

7 R's

1. Quem **R**equisitou a mudança?
2. Qual é a **R**azão para a mudança?
3. Qual é o **R**etorno requerido da mudança?
4. Quais são os **R**iscos envolvidos na mudança?
5. Quais são os **R**ecursos necessários para a entrega da mudança?
6. Quem é o **R**esponsável pela construção, teste e implementação da mudança?
7. Qual é o **R**elacionamento entre esta mudança e outras?

Etapas

1. Pedido da mudança.
2. Aprovação da mudança.
3. Atribuição de tarefas.
4. Realização da mudança.
5. Finalização da mudança

Ferramenta de controle de mudanças

- Acompanha o ciclo de vida do pedido de mudança.
- Permite rastrear a mudança pelo pedido e pela identificação na ferramenta de controle de versão.
- Anexa documentos ao pedido.
- Configura o fluxo de trabalho.
- Notifica os interessados.
- Gera relatórios.

Exemplos de ferramentas

- Redmine
- Bugzilla
- Jira
- Trac
- IBM Rational ClearQuest

Exercício

1) Selecione um membro do grupo que será o **dono do repositório**.

2) O **dono do repositório** deverá criar um novo repositório chamado **devops-aula07** que deverá ter somente um arquivo **README.md**.

3) O **dono do repositório** deverá adicionar os demais membros do grupo como colaboradores no repositório.

4) O **dono do repositório** deverá configurar o repositório para proteger o *branch* **master**, ativando a opção *"Require pull request reviews before merging"*.

5) O **dono do repositório** deverá criar os seguintes *milestones*:

- v1.0: Inicialização do tabuleiro
- v1.1: Verificação da validade das jogadas
- v1.2: Verificação do estado do jogo
- v1.3: Jogo entre dois jogadores humanos

6) O **dono do repositório** deverá criar o *project* v1.0.

7) O **dono do repositório** deverá criar 3 *boards* no *project* v1.0:

- ToDo
- Doing
- Done

8) O **dono do repositório** deverá criar os seguintes *labels*:

- análise
- arquitetura
- programação
- testes
- operações

9) O **dono do repositório** deverá criar um novo *branch* a partir do **master** chamado *v1.0*.

10) O **dono do repositório** deverá criar um novo *issue*:

- Título: Especificar inicialização do tabuleiro.
- *Assignees*: todos os membros da área de análise.
- *Labels*: análise
- *Milestone*: v1.0
- *Projects*: v1.0

O **dono do repositório** deverá criar um novo *card* com este *issue* no *board* **ToDo** do *project* **v1.0**.

11) Um dos **membros da área de análise** deverá mover o card "**Especificar inicialização do tabuleiro**" do *board* **ToDo** para o *board* **Doing** no *project* **v1.0**.

A seguir, deverá criar um novo *branch* chamado **v1.0-analise** a partir do *branch* **v1.0** (ATENÇÃO! não é para ser criado a partir do **master**).

12) No *branch* **v1.0-analise**, um dos **membros da área de análise** deverá criar o arquivo **docs/requisitos.md** com o seguinte conteúdo:

```
# Requisitos

## Estados e inicialização do tabuleiro

* O sistema deverá manter o estado de cada uma das casas de um jogo da velha.

* Cada casa do jogo da velha poderá estar vazia, ocupada pelo 1o jogador ou ocupada pelo 2o jogador.
```

13) Ao efetuar a operação de *commit* deste arquivo, adicionar no comentário a identificação do *issue* (se este é o primeiro, provavelmente a identificação é #**1**).

14) Um dos **membros da área de análise** deverá mover o card "**Especificar inicialização do tabuleiro**" do *board* **Doing** para o *board* **Done** no *project* **v1.0**.

15) Um **membro da área de análise** deverá criar um novo *pull request*:

- Título: Integrar requisitos v1.0
- Base: v1.0 (ATENÇÃO! Não é para usar o **master** neste campo)
- Compare: v1.0-analise
- *Reviewers*: dono do repositório
- *Assignee*: dono do repositório
- *Label*: análise
- *Milestone*: v1.0
- *Projects*: v1.0

16) O **dono do repositório** deverá abrir o *pull request* e realizar a operação de *merge* solicitada.

A seguir, deverá fechar o *issue* que foi atendido pela área de análise.

17) O **dono do repositório** deverá criar um novo *issue*:

- Título: Definir arquitetura da inicialização do tabuleiro.
- *Assignees*: todos os membros da área de arquitetura.
- *Labels*: arquitetura
- *Milestone*: v1.0
- *Projects*: v1.0

O **dono do repositório** deverá criar um novo *card* com este *issue* no *board* **ToDo** do *project* **v1.0**.

18) Um dos **membros da área de arquitetura** deverá mover o card "**Definir arquitetura da inicialização do tabuleiro**" do *board* **ToDo** para o *board* **Doing** no *project* **v1.0**.

A seguir, deverá criar um novo *branch* chamado **v1.0-arquitetura** a partir do *branch* **v1.0** (ATENÇÃO! não é para ser criado a partir do **master**).

19) No *branch* **v1.0-arquitetura**, um dos **membros da área de arquitetura** deverá criar o arquivo **docs/arquitetura.md** com o seguinte conteúdo:

```
# Arquitetura

## Inicialização do tabuleiro

* As funções relacionadas ao gerenciamento das casas do jogo da velha ficarão no módulo **jogovelha.py**.

* O estado de cada casa do jogo será representada por uma string: "." para casa vazia; "X" para casa ocupada pelo 1o jogador; "O" para casa ocupada pelo 2o jogador

* A função inicializar() retornará uma lista 3x3, onde cada posição conterá uma string para indicar o estado de uma casa do jogo. A função retornará todas as casas inicialmente vazias.

* A função jogar(jogador, linha, coluna) irá posicionar o **jogador** ('X' ou 'O') na posição definida por **linha** e **coluna**.
```

20) Ao efetuar a operação de *commit* deste arquivo, adicionar no comentário a identificação do *issue* (se você está seguindo o tutorial desde o início, provavelmente a identificação é #**3**).

21) Um dos **membros da área de arquitetura** deverá mover o card "**Definir arquitetura da inicialização do tabuleiro**" do *board* **Doing** para o *board* **Done** no *project* **v1.0**.

21) Um **membro da área de arquitetura** deverá criar um novo *pull request*:

- Título: Integrar arquitetura v1.0
- Base: v1.0 (ATENÇÃO! Não é para usar o **master** neste campo)
- Compare: v1.0-arquitetura
- *Reviewers*: dono do repositório
- *Assignee*: dono do repositório
- *Label*: arquitetura
- *Milestone*: v1.0
- *Projects*: v1.0

22) O **dono do repositório** deverá abrir o *pull request* e realizar a operação de *merge* solicitada.

A seguir, deverá fechar o *issue* que foi atendido pela área de arquitetura.

23) O **dono do repositório** deverá criar um novo *issue*:

- Título: Implementar inicialização do tabuleiro.
- *Assignees*: todos os membros da área de programação.
- *Labels*: programação
- *Milestone*: v1.0
- *Projects*: v1.0

O **dono do repositório** deverá criar um novo *card* com este *issue* no *board* **ToDo** do *project* **v1.0**.

24) Um dos **membros da área de programação** deverá mover o card "**Implementar a inicialização do tabuleiro**" do *board* **ToDo** para o *board* **Doing** no *project* **v1.0**.

A seguir, deverá criar um novo *branch* chamado **v1.0-programacao** a partir do *branch* **v1.0** (ATENÇÃO! não é para ser criado a partir do **master**).

25) No *branch* **v1.0-programacao**, um dos **membros da área de programação** deverá criar o arquivo **src/jogovelha.py** com o seguinte conteúdo:

```
TAB = []

def inicializar() :
    TAB.append(['.','.','.'])
    TAB.append(['.','.','.'])
    TAB.append(['.','.','.'])

def jogar(jogador, linha, coluna):
    if jogador !='X' and jogador != 'O':
        raise RuntimeError('Jogador inválido!')
    valores = list(range(0,3))
    if linha not in valores:
        raise RuntimeError('Linha inválida!')
    if coluna not in valores:
        raise RuntimeError('Coluna inválida!')
    TAB[linha][coluna] = jogador

def tabuleiro():
    return TAB

def main():
    inicializar()
    jogar('X', 1, 1)
    print(tabuleiro())

if __name__ == "__main__":
    main()
```

26) Ao efetuar a operação de *commit* deste arquivo, adicionar no comentário a identificação do *issue* (se você está seguindo o tutorial desde o início, provavelmente a identificação é #**5**).

27) Um dos **membros da área de programação** deverá mover o card "**Implementar a inicialização do tabuleiro**" do *board* **Doing** para o *board* **Done** no *project* **v1.0**.

28) Um **membro da área de programação** deverá criar um novo *pull request*:

- Título: Integrar implementação v1.0
- Base: v1.0 (ATENÇÃO! Não é para usar o **master** neste campo)
- Compare: v1.0-programacao
- *Reviewers*: dono do repositório
- *Assignee*: dono do repositório
- *Label*: programação
- *Milestone*: v1.0
- *Projects*: v1.0

29) O **dono do repositório** deverá abrir o *pull request* e realizar a operação de *merge* solicitada.

A seguir, deverá fechar o *issue* que foi atendido pela área de arquitetura.

30) O **dono do repositório** deverá criar um novo *issue*:

- Título: Testar inicialização do tabuleiro.
- *Assignees*: todos os membros da área de testes.
- *Labels*: testes
- *Milestone*: v1.0
- *Projects*: v1.0

O **dono do repositório** deverá criar um novo *card* com este *issue* no *board* **ToDo** do *project* **v1.0**.

31) Um dos **membros da área de testes** deverá mover o card "**Testar a inicialização do tabuleiro**" do *board* **ToDo** para o *board* **Doing** no *project* **v1.0**.

A seguir, deverá criar um novo *branch* chamado **v1.0-testes** a partir do *branch* **v1.0** (ATENÇÃO! não é para ser criado a partir do **master**).

32) No *branch* **v1.0-testes**, um dos **membros da área de testes** deverá criar o arquivo **src/testes_inic.py** com o seguinte conteúdo:

```
import jogovelha
import sys

erroInicializar = False

jogovelha.inicializar()
jogo = jogovelha.tabuleiro()

if len(jogo) != 3:
    erroInicializar = True
else:
    for linha in jogo:
        if len(linha) != 3:
            erroInicializar = True
        else:
            for elemento in linha:
                if elemento != '.':
                    erroInicializar = True

if erroInicializar:
    print('Erro!')
    sys.exit(1)
else:
    sys.exit(0)
```

33) No *branch* **v1.0-testes**, um dos **membros da área de testes** deverá criar o arquivo **src/testes_jogar.py** com o seguinte conteúdo:

```
import jogovelha
import sys

erro = False
lin = 1
col = 1
jogador = 'X'
jogovelha.inicializar()
jogovelha.jogar(jogador, lin, col)
jogo = jogovelha.tabuleiro()

if len(jogo) != 3:
    erro = True
else:
    for linha in range(0,3):
        if len(jogo[linha]) != 3:
            erro = True
        else:
            for coluna in range(0,3):
                if linha == lin and coluna == col:
                    if jogo[linha][coluna] != jogador:
                        erro = True
                elif jogo[linha][coluna] != '.':
                    erro = True

if erro:
    print('Erro!')
    sys.exit(1)
else:
    sys.exit(0)
```

34) **Um dos membros da área de testes** deverá executar os **scripts de teste** (**teste_inic.py** e **teste_jogar.py**) e continuar somente em caso de sucesso.

35) Ao efetuar a operação de *commit* deste arquivo, adicionar no comentário a identificação do *issue* (se você está seguindo o tutorial desde o início, provavelmente a identificação é #**7**).

36) Um dos **membros da área de programação** deverá mover o card "**Testar a inicialização do tabuleiro**" do *board* **Doing** para o *board* **Done** no *project* **v1.0**.

37) Um **membro da área de testes** deverá criar um novo *pull request*:

- Título: Integrar testes v1.0
- Base: v1.0 (ATENÇÃO! Não é para usar o **master** neste campo)
- Compare: v1.0-testes
- *Reviewers*: dono do repositório
- *Assignee*: dono do repositório
- *Label*: testes
- *Milestone*: v1.0
- *Projects*: v1.0

38) O **dono do repositório** deverá abrir o *pull request* e realizar a operação de *merge* solicitada.

A seguir, deverá fechar o *issue* que foi atendido pela área de arquitetura.

39) O **dono do repositório** deverá criar um novo *pull request*:

- Título: Gerar release v1.0
- Base: **master**
- Compare: v1.0
- *Reviewers*: membros da área de operações
- *Assignee*: membros da área de operações
- *Label*: operações
- *Milestone*: v1.0
- *Projects*: v1.0

40) Um dos membros da **área de operações** deverá abrir o *pull request,* adicionar um ***review*** e realizar a operação de *merge* solicitada.

41) Um dos membros da **área de operações** deverá criar um ***release*** chamado **v1.0**.

Faculdade

IMPACTA

TECNOLOGIA