WBA0890_v1.0

Práticas da cultura DevOps no desenvolvimento de sistemas

Selecionando e conhecendo ferramentas do DevOps no desenvolvimento web

Ferramentas DevOps

Bloco 1

Stella Marys Dornelas Lamounier

Sistema de controle de versões (versionamento)

É um software que se encarrega de gerenciar mudanças em arquivos físicos.

Gerenciamento de múltiplas versões: documentos; projetos; software; sistemas de arquivos; suíte de escritórios; gerenciamento de software.

Permite consultar revisões anteriores.

Comparar revisões.

Trabalho cooperativo.

Sistema de controle de versões (versionamento)

Toda alteração realizada é registrada.

Rápido acesso à informação.

Descartar código ruim sem remorso.

Manutenção de código.

Apontar quais são os desenvolvedores responsáveis por cada trecho de código.

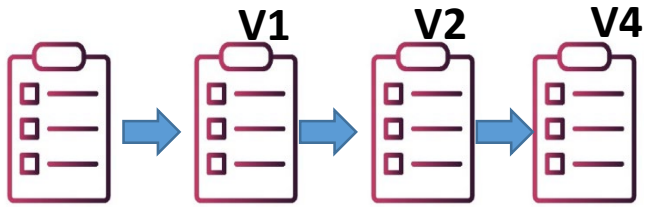
Comparação entre versões de projetos.

Segurança.

Colaboração.

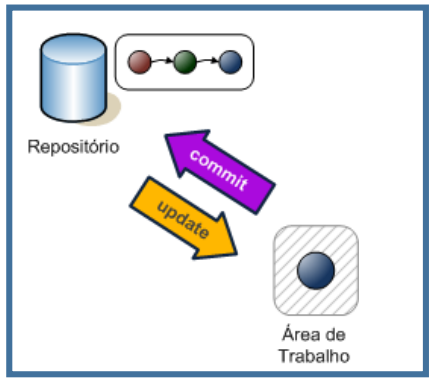
Sistema de controle de versões (versionamento)

Figura 1 – Versionamento



Fonte: elaborada pela autora.

Figura 2 – Controle de versão entre usuário e repositório



Fonte: Dias (2016).

Ferramentas DevOps – Controle de versão

Modelo centralizado:

- Repositório local para revisões.
- Modelo baseado em cliente-servidor.
- Utilizado por equipes menores.
- A equipe obtém cópias a partir de repositórios e alterações de outros desenvolvedores.

Ferramentas DevOps – Controle de versão

Modelo centralizado – Vantagens

Controle de acesso.

Cópia de segurança.

Controle de qualidade.

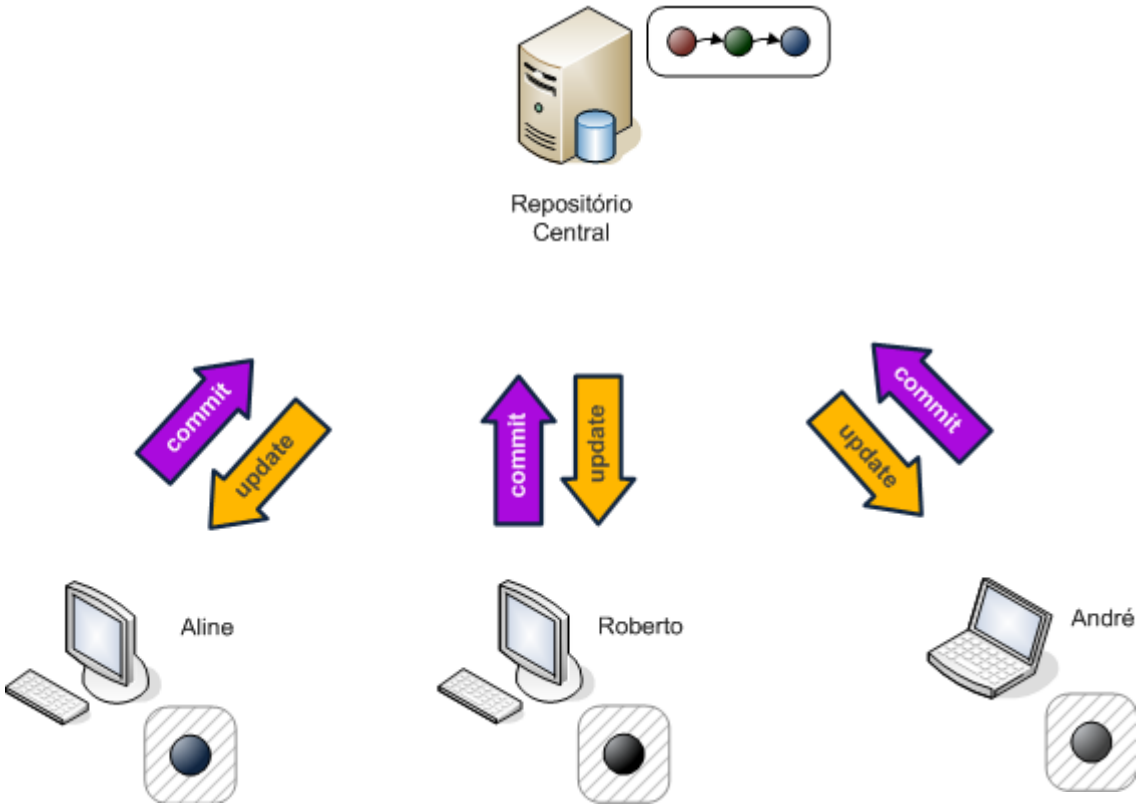
Modelo centralizado – Desvantagens

Dependência de um repositório central.

Ponto único de falha.

Como funciona o controle de versão centralizado?

Figura 3 – Controle de versão centralizado



Fonte: Dias (2016).

Ferramentas DevOps – Controle de versão

Modelo descentralizado:

Cada desenvolvedor tem seu próprio repositório.

Desenvolvedores copiam repositórios e alterações de outros desenvolvedores.

Utilizados por equipes maiores, em lugares diferentes.

Ferramentas DevOps – Controle de versão

Modelo descentralizados – Vantagens

Permite trabalhar de off-line.

Melhor suporte a ramificação/mesclagem.

Mais rápido.

Independência da rede.

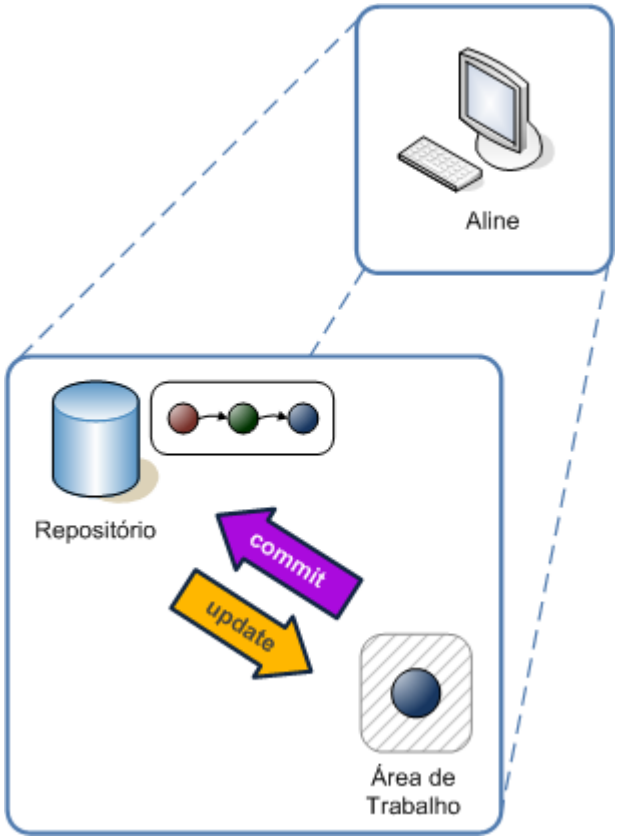
Modelo descentralizados - Desvantagens

Estimula o isolamento de desenvolvedores.

Questões de privacidade e segurança.

Como funciona o controle de versão centralizado?

Figura 4 - Controle de versão descentralizado



Fonte: Dias (2016).

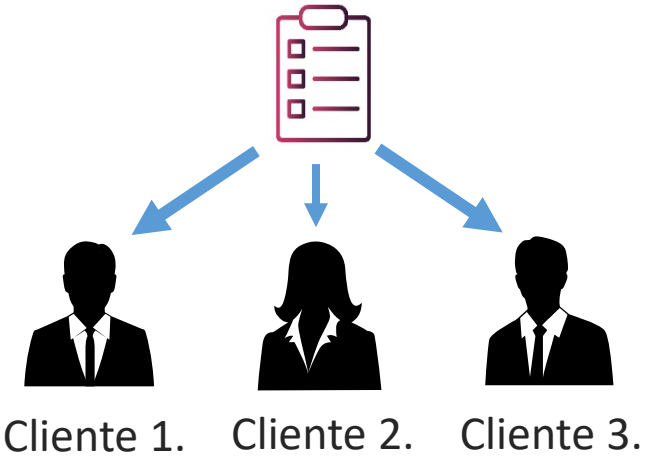
Repositório

Local de armazenamento do projeto com todas as suas alterações.

Reflete a história do projeto.

Podem sofrer atualizações ao longo do tempo. (commit)

Figura 5 - Repositório



Fonte: elaborada pela autora.

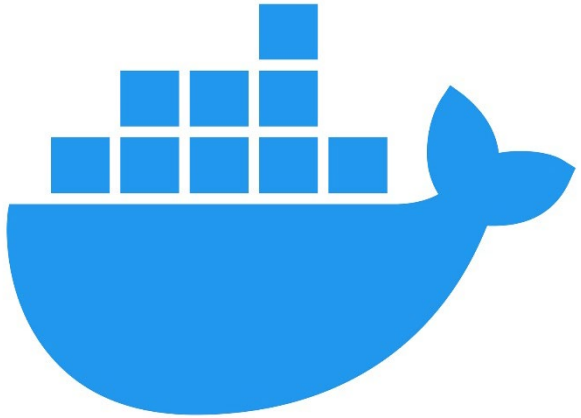
Ferramenta DevOps

O controle de versionamento torna o projeto mais escalável, por trazer monitoramento dos históricos, no qual permite a entrega rápida de artefatos para produção.

Figura 6 – Ferramenta GIT

Fonte: https://pt.wikipedia.org/wiki/Git#/media/Ficheiro:Git-logo.svg. Acesso em: 14 jan. 2022.

Figura 7 – Ferramenta DOCKE



Fonte: Oleg Mishutin/iStock.com.

Selecionando e conhecendo ferramentas do DevOps no desenvolvimento web.

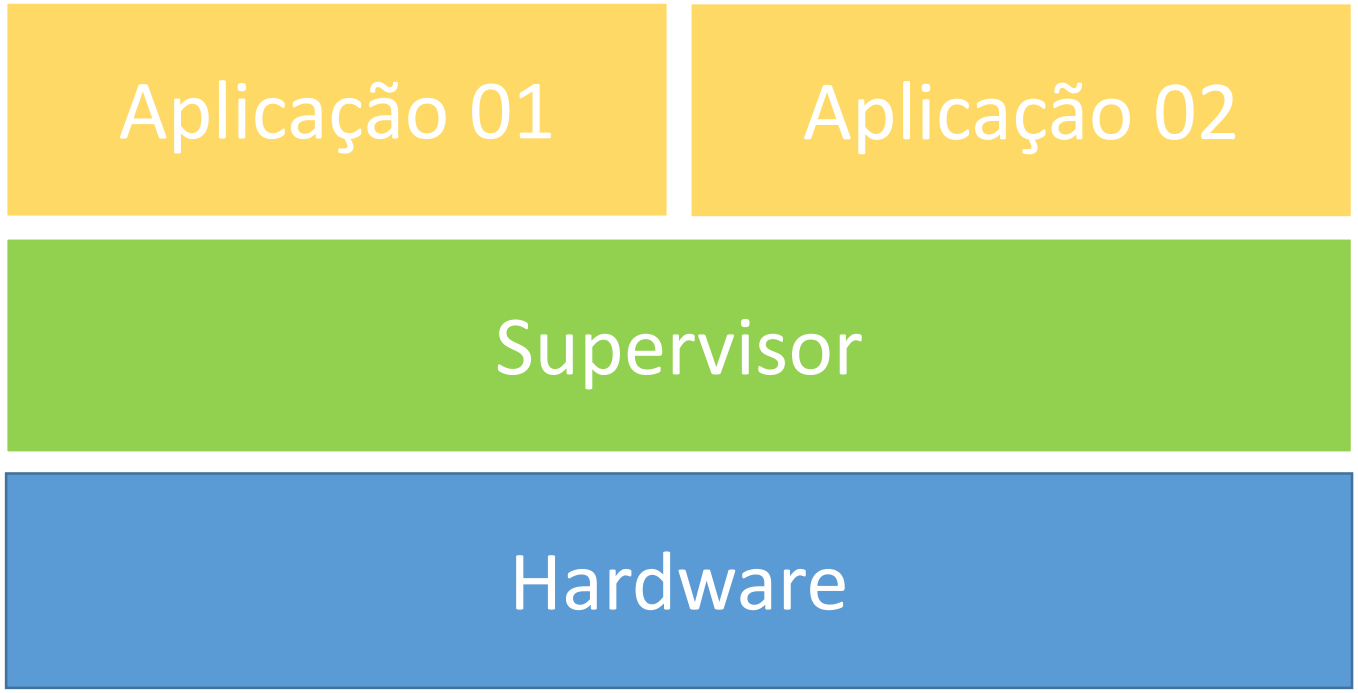
Docker e Máquinas Virtuais (MV)

Bloco 2

Stella Marys Dornelas Lamounier

Máquinas virtuais

Figura 8 – Máquina virtual



Fonte: adaptada de Duarte (2018).

Máquinas virtuais

Quadro 1 – Vantagens versus desvantagens da MV

Vantagens	Desvantagens
Compartilhar um sistema com muitos ambientes virtuais. Melhor aproveitamento de hardware. Economia. Uso de sistemas operacionais antigos. Diversidade de plataformas.	Sobrecarga de tarefas. Segurança. Dependência. Espaço em disco. Sem acesso direto ao hardware. Acúmulo de máquinas virtuais.

Fonte: elaborado pela autora.

Docker

Docker é uma plataforma que permite "criar, enviar e executar qualquer aplicativo, em qualquer lugar“, utilizando container.

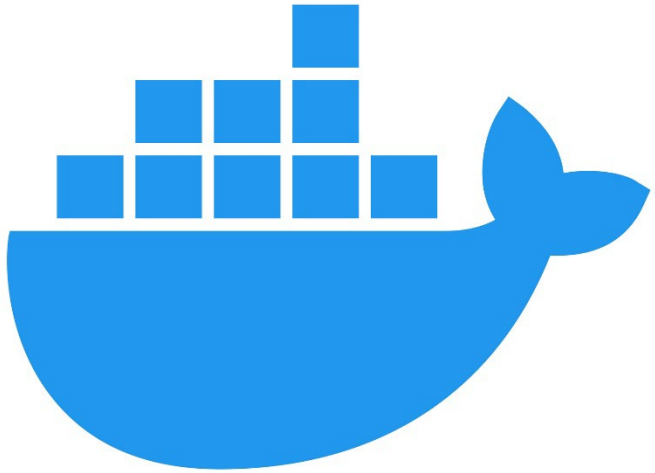
Resolve problemas ligados à implantação.

Tecnologia de virtualização.

Utilizado por mais de 13 milhões de desenvolvedores.

Mais de 13 bilhões de downloads.

Figura 9 – Logomarca Docker



Fonte: Oleg Mishutin/iStock.com.

Arquitetura Docker – Containers

Contêiner é uma unidade padrão de software que empacota o código e todas as suas dependências.

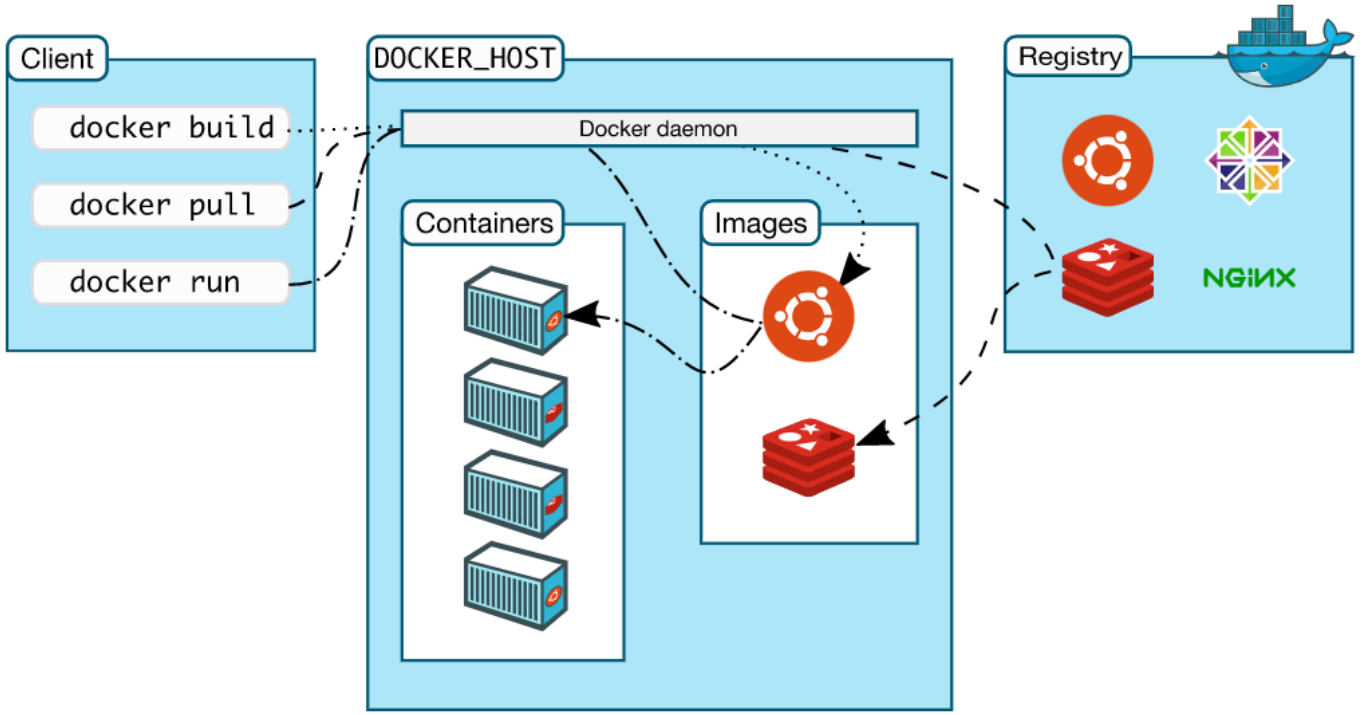
Trata-se de um pacote de software independente e executável que traz o necessário para executar um aplicativo.

Com menos custo de processamento/manutenção e configuração.

Com menor tempo de inicialização/finalização.

Arquitetura Docker – Containers

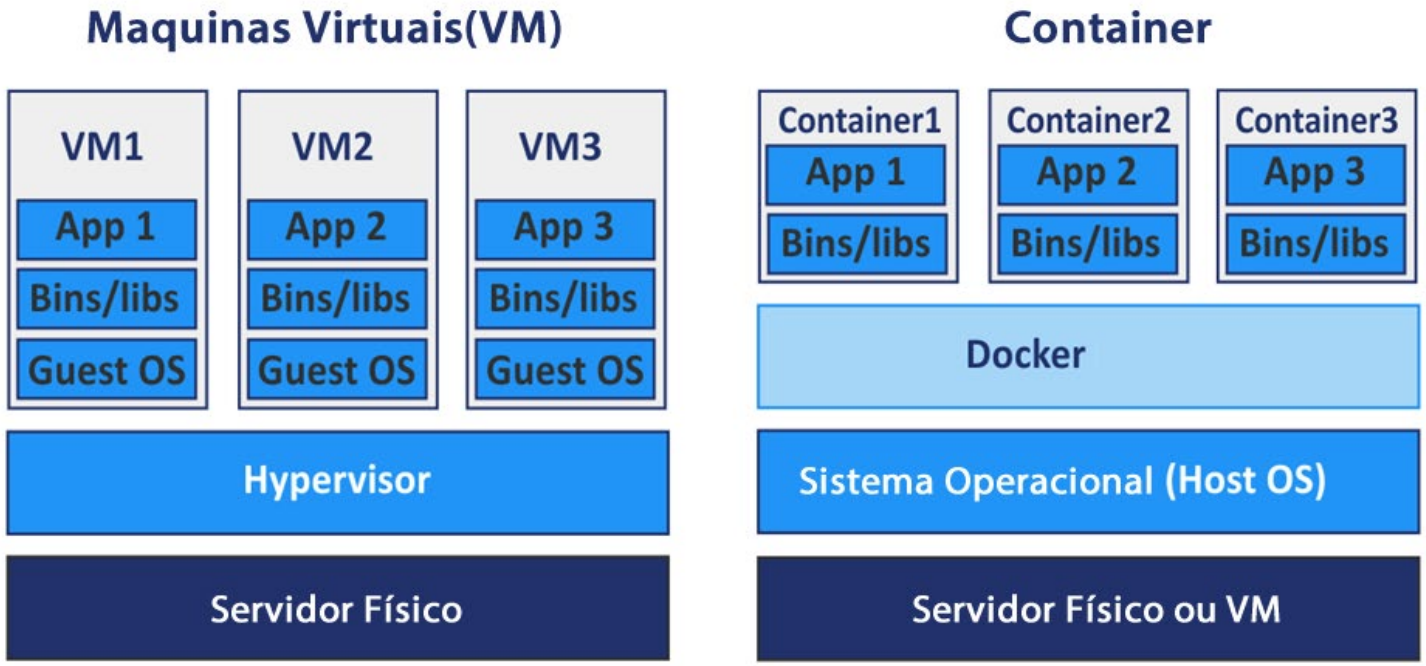
Figura 10 – Arquitetura Docker



Fonte: Docker (2012)b.

Máquina Virtual versus Docker

Figura 11 - Máquina Virtual versus Docker



Fonte: adaptada de Nakivo (2021).

Docker – Vantagens

Economia de recursos:

Maior disponibilidade do sistema.

Compartilhamento.

Facilidade de gerenciamento.

Ambientes similares.

Padronização e replicação.

Menor utilização de memória.

Docker Hub

O Docker Hub é o repositório oficial do Docker para imagens, e funciona de forma semelhante ao git. É possível hospedar, baixar, procurar por imagens, e ainda conta com uma documentação orientando a forma de usá-las.

Selecionando e conhecendo ferramentas do DevOps no desenvolvimento web

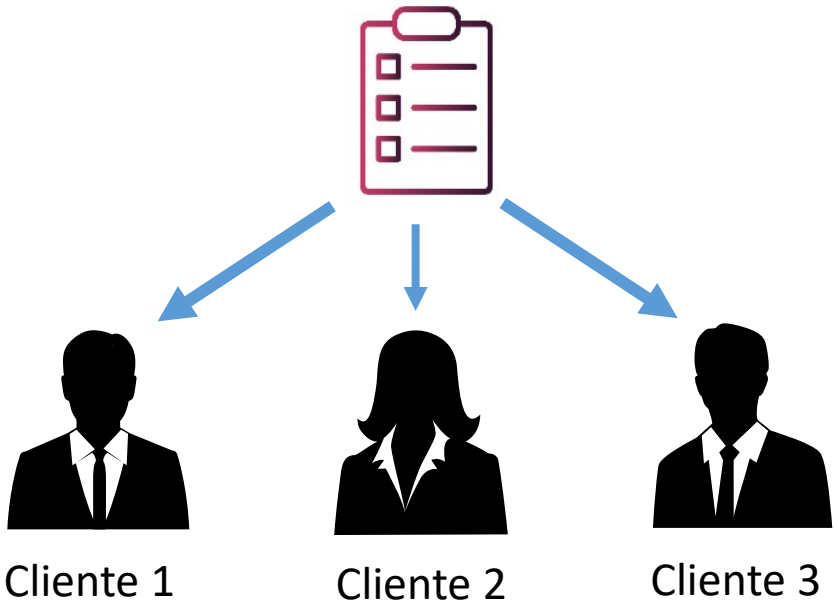
Repositório

Bloco 3

Stella Marys Dornelas Lamounier

Repositórios

Figura 12 – Repositório



Fonte: elaborada pela autora.

Versionamento e repositório

Figura 13 – Tipos de sistemas de versionamento

Fonte: Mauro (2017).

Git

Sistema de controle de versão de código livre e aberto, desenvolvido para o desenvolvimento do Kernel do Linux.

Distribuído.

Flexível.

Segurança.

Ramos individuais.

Rápido.

Redução de custos com o servidor.

Método ***Branch.***

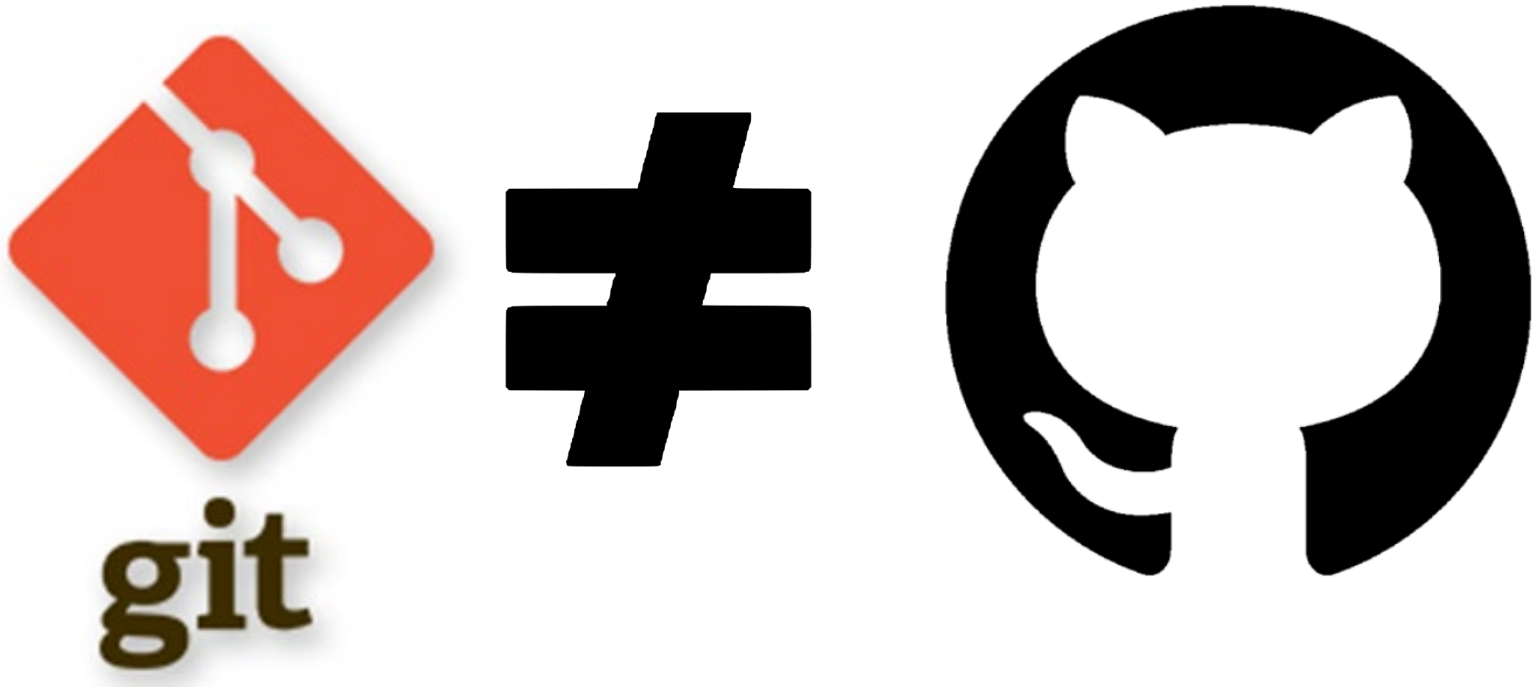
Empresas que utilizam versionamento

Figura 14 – Empresas que utilizam sistemas de versionamento

Fonte: Ribeiro (2020).

Git verus GitHub

Figura 15 - Git versus GitHub



Fonte: adaptada de https://pt.wikipedia.org/wiki/Git#/media/Ficheiro:Git-logo.svg. Acesso em: 14 jan. 2022.

Git versus CVS/ SVN

Sistema de Versão	Prós	Contra
CVS e SVN	Tecnologia madura. Controle central sobre os projetos. Um repositório e muitos clientes. Repositório central. Segurança de acesso.	Mover ou renomear arquivos não inclui uma atualização de versão. Lento.
Git	Modelo descentralizado Rápido. Pode trabalhar de forma off-line. Cópia funciona como backup. É um repositório com vários repositórios de clientes.	Dificuldade de gerenciamento. Centralizado. Complexidade.

Comandos básicos do Git

A pasta.git: está localizada no diretório raiz do projeto, e contém todas as configurações do projeto.

Git init: primeiro comando a ser utilizado, responsável por criar um repositório vazio local.

Git clone: um comando para baixar o código-fonte existente de um repositório remoto:

- git clone https://url-do-link.

Git status: verifica se os arquivos foram alterados e rastreados.

Git add: comando para adicionar arquivos no repositório local ou que as alterações de um arquivo sejam também gravadas:

- add.
- git add meu_arquivo1.txt .

Comandos básicos do Git

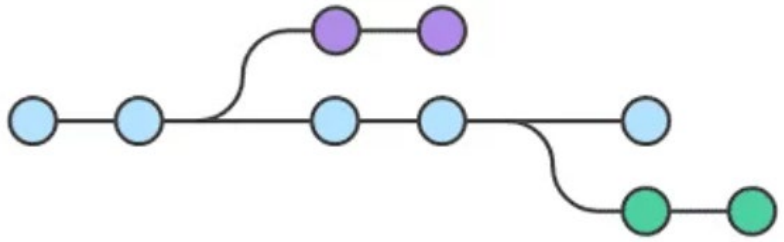
Git commit: mudança individual de um ou mais arquivos.

Utilizado para salvar as mudanças que estavam em stage no seu repositório local.

git commit meu_arquivo1.txt

```
$ git commit -m "Criando arquivos no modelo html"
```

Git branch: lista todas as *branchs* existentes.



Comandos básicos do Git

Git push: envio do *commit* local para o repositório remoto.

Git pull: recuperação das novas mudanças do repositório remoto para o repositório local, mudanças do repositório remoto para seu repositório local.

Comandos básicos do Git

Ls: lista.

Clear: limpa.

git -- version: mostra a versão.

git –help: mostra uma lista de comandos para ajuda.

rm -rf .git: apaga a pasta .git criada no diretório.

git checkout: remove as alterções deixando o arquivo como no último commit.

Git log: mostra o histórico de commits.

Teoria em Prática

Bloco 4

Stella Marys Dornelas Lamounier

Reflita sobre a seguinte situação

- Imagine a seguinte situação: você é desenvolvedor em uma empresa de criação de sistemas computacionais, e está trabalhando em um grande projeto para a criação de um software para uma universidade.
- Você precisa corrigir alguns erros apresentados e também adicionar recursos solicitados ao longo do desenvolvimento pelo cliente.
- Como você faria estas alterações citadas, utilizando o Git, mas sem comprometer a parte estável do código, isto é, sem que haja alteração nas linhas principais?

Norte para a resolução

- O melhor caminho é a utilização de *branch,* pois trata de um poderoso comando dentro de Git que pode adicionar funcionalidades, corrigir erros sem que haja alterações e prejuízos no projeto. Posteriormente, após a conclusão destas alterações no código (ramificações), o desenvolvedor poderá mesclar a ramificação com a principal, geralmente chamada de *master*.

Dica do(a) Professor(a)

Bloco 5

Stella Marys Dornelas Lamounier

Dicas

- Conhecer o ambiente Azure DevOps.

Referências

BOSE, Michael. Kubernetes vs Docker – What Is the Difference? **NAKIVO**, [s.l.], 13 de maio de 2019. Disponível em: https://www.nakivo.com/blog/docker-vs-kubernetes/. Acesso em: 12 set. 2021.

MONTEIRO, E. R. et al. DevOps. Porto Alegre: **Grupo A**, 2021. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786556901725/. Acesso em: 20 set. 2021.

MUNIZ, A. et al. **Jornada DevOps:** unindo cultura ágil, Lean e tecnologia para entrega de software de qualidade. Rio de Janeiro: Brasport, 2019.

Bons estudos!