Universidade Federal do Ceará
Centro de Ciências/Departamento de Computação
Código da Disciplina: CK0236
Professor: Ismayle de Sousa Santos

Aula 02

Técnica de Programação II

Ambiente de Desenvolvimento Integrado
Versionamento de Código

 qpg4p5x  ismaylesantos@great.ufc.br  @IsmayleSantos

Agenda

- IDE
  - O que é?
  - Vantagens
  - Exemplos
  - Como escolher?
- Versionamento de Código
  - O que é?
  - Tipos
  - Versionamento Semântico
  - GitFlow
  - Git
  - Conventional Commits
  - Change Logs

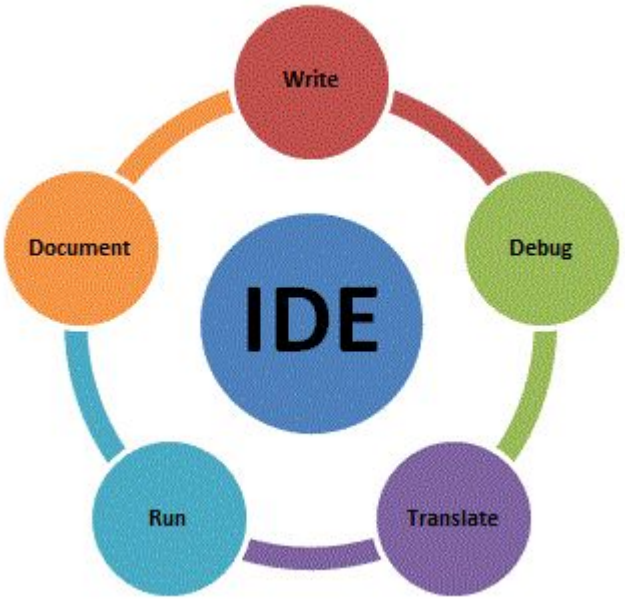
Vamos começar falando sobre ...

Integrated Development Environment

Alguém sabe o que é?

IDE

- Ambiente de desenvolvimento integrado
  - **I**ntegrated **D**evelopment **E**nvironment (IDE)
- Uma IDE contém
  - Editor de código fonte
  - Compilador/interpretador
  - Depurador
  - Outros
    - Sistema de Versionamento
    - Widgets



Exemplos de IDE

Benefícios de usar uma IDE

- Aumenta produtividade
- Facilidades
  - Compilação, Deploy, Depuração
- Feedback
  - Erros de codificação
  - Erros de compilação

Eclipse IDE

- Ambiente de desenvolvimento popular
- IDE de código aberto
- Multilinguagem
- Multiplataforma
- Forte orientação ao desenvolvimento baseado em plug-ins

Principais IDEs - Top Index

Worldwide, Nov 2020 compared to a year ago:

Rank	Change	IDE	Share	Trend
1	↑	Visual Studio	25.31 %	+3.6 %
2	↑	Eclipse	16.31 %	-1.1 %
3	↓↓	Android Studio	11.39 %	-11.3 %
4	↑	Visual Studio Code	8.74 %	+3.4 %
5	↑	pyCharm	7.73 %	+2.5 %
6	↑	IntelliJ	6.13 %	+1.3 %
7	↓↓↓	NetBeans	5.3 %	-0.2 %

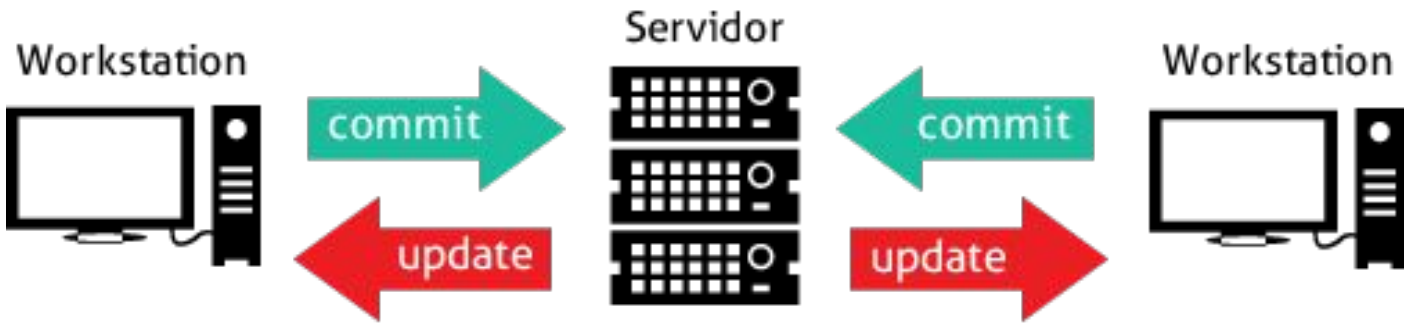
Fonte: https://pypl.github.io/IDE.html

Qual IDE escolher?

- Linguagem de Programação
- Integrações com outras ferramentas e plugins

Versionamento de Código

- O que é Controle de versão?
  - É o registro de alterações em um arquivo ou conjunto de arquivos ao longo do tempo, para que você possa recuperar versões específicas mais tarde



Versionamento de Código

- Sistemas de gerenciamento de Versões
  - Git
  - Subversion (SVN)
  - SourceSafe

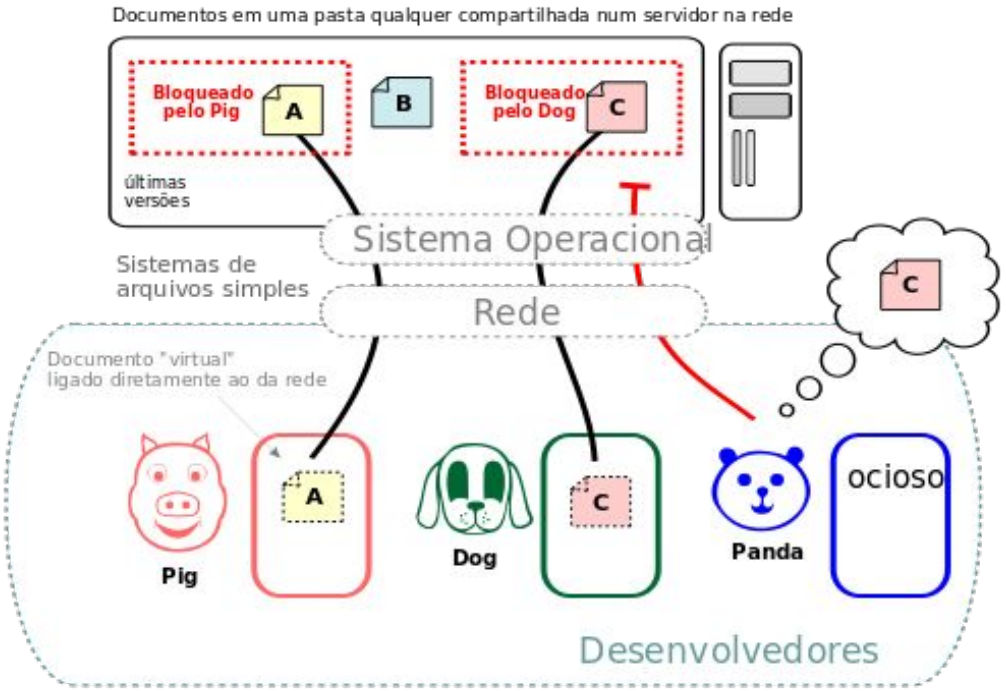
Versionamento de Código

- Benefícios do Controle de versão
  - Histórico de alterações
  - Desenvolvimento Paralelo
  - Desenvolvedores trabalhando no mesmo código
  - Restaurar uma determinada versão

- Sistemas de controle de versão também podem ser usados para armazenar
    - documentação
    - manuais
    - relatórios, etc

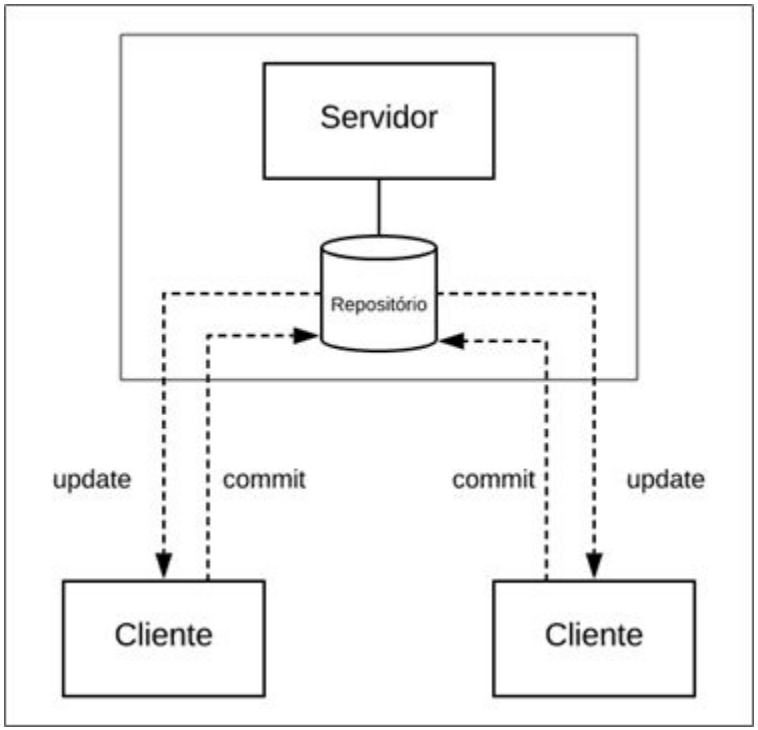
Controle de Versão

- Sem controle de versão
  - Arquivos ficam bloqueados enquanto estão sendo utilizados



Fonte: https://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%B5es

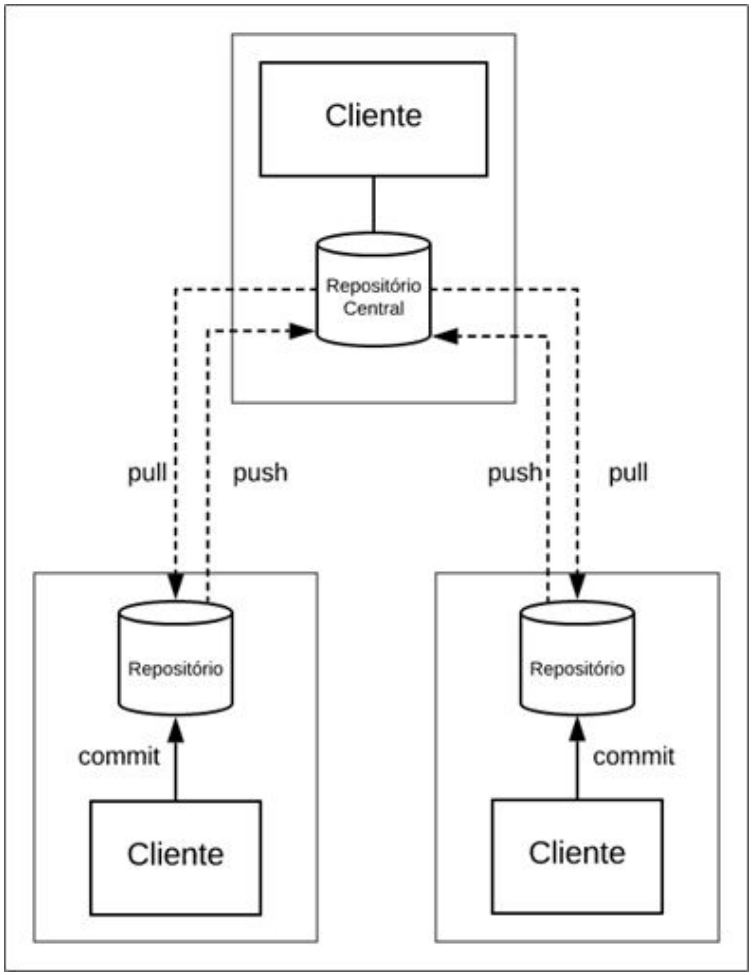
Sistema de Controle de Versão Centralizado (e.g. svn, cvs)



- Arquitetura Cliente-servidor
- Existe um único servidor que armazena o repositório e o sistema de controle de versões

Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

Sistema de Controle de Versão Distribuído (e.g. git)



- Arquitetura peer-to-peer
- Cada desenvolvedor possui em sua máquina um servidor completo de controle de versões
- Uma máquina principal contém o repositório central
  - Armazena a versão de referência do código fonte

Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

Vantagens do Sistema de Controle de Versão Distribuído

- Pode-se fazer commits com mais frequência
  - Implementações não precisam ser integrados imediatamente com o repositório central
- Commits são mais rápidos
  - Realizados no repositório local
- Pode-se trabalhar off-line
  - Commits são realizados primeiro no repositório instalado localmente na máquina do desenvolvedor

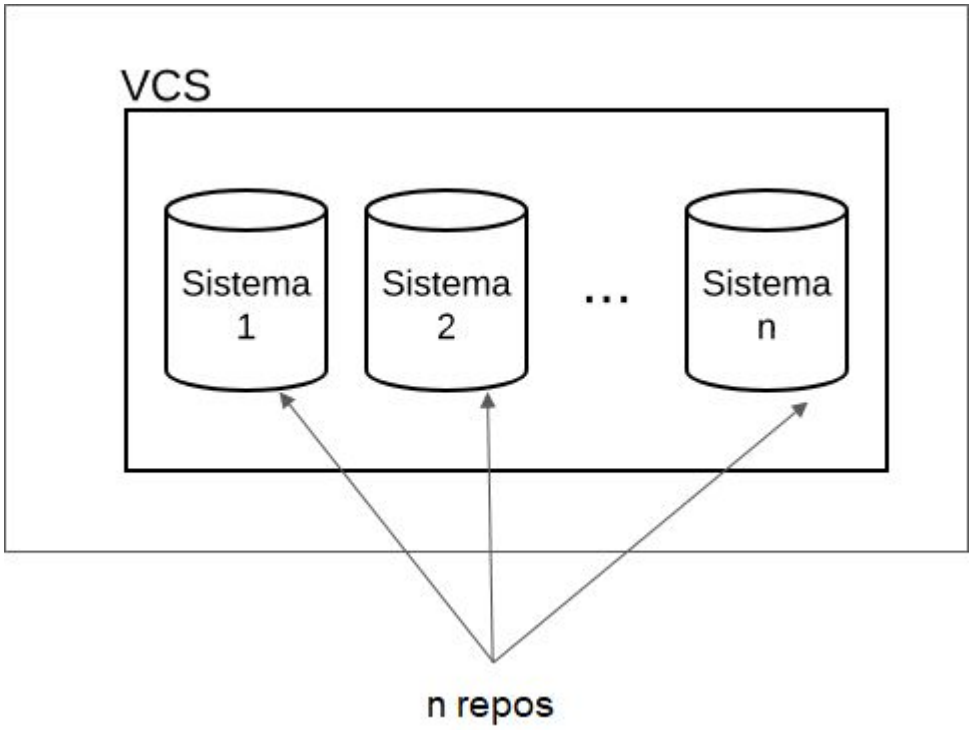
Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

Git e GitHub

- Git
  - Sistema de controle de versões distribuído
- GitHub
  - Serviço de hospedagem de código que usa o sistema Git para prover controle de versões
  - Repositórios públicos e gratuitos
  - Sistemas similares:
    - GitLab
    - BitBucket

Fonte: http://gabsferreira.com/instalando-o-git-e-configurando-github/

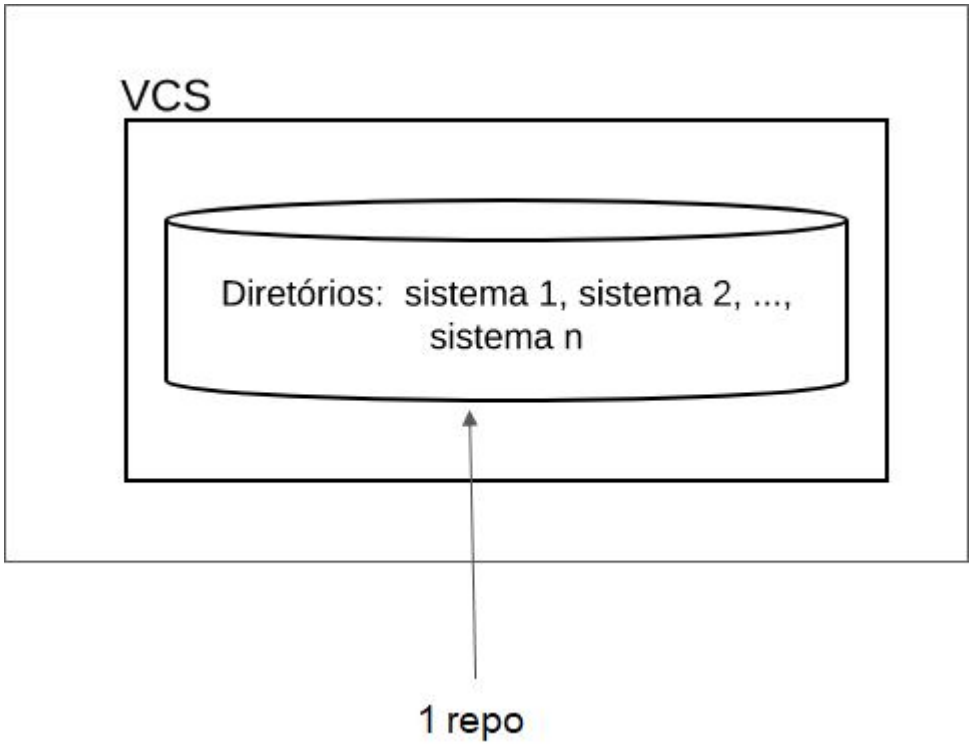
Multirepo (mais comum)



- Sistema de Controle de Versão gerencia vários repositórios

Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

Monorepo (um pouco menos comum)



- Sistema de Controle de Versão gerencia um único repositório
  - Projetos são diretórios dentro do repositório

Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

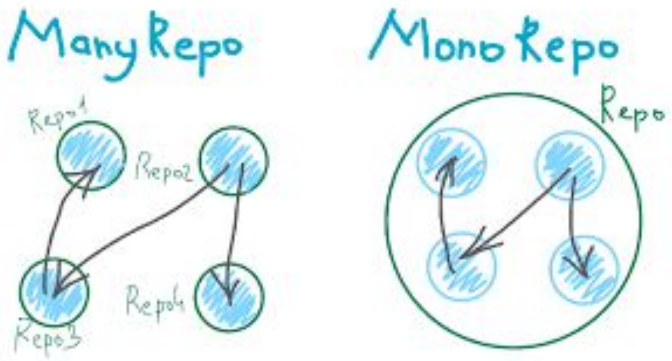
Exemplo

Multirepos	Monorepo
• aserg-ufc/sistema1	• aserg-ufc/sistemas
• aserg-ufc/sistema2	• Diretórios neste repo:
• aserg-ufc/sistema3	○ sistema1
• aserg-ufc/sistema4	○ sistema2
• aserg-ufc/sistema5	○ sistema3

Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

Vantagens de Monorepos

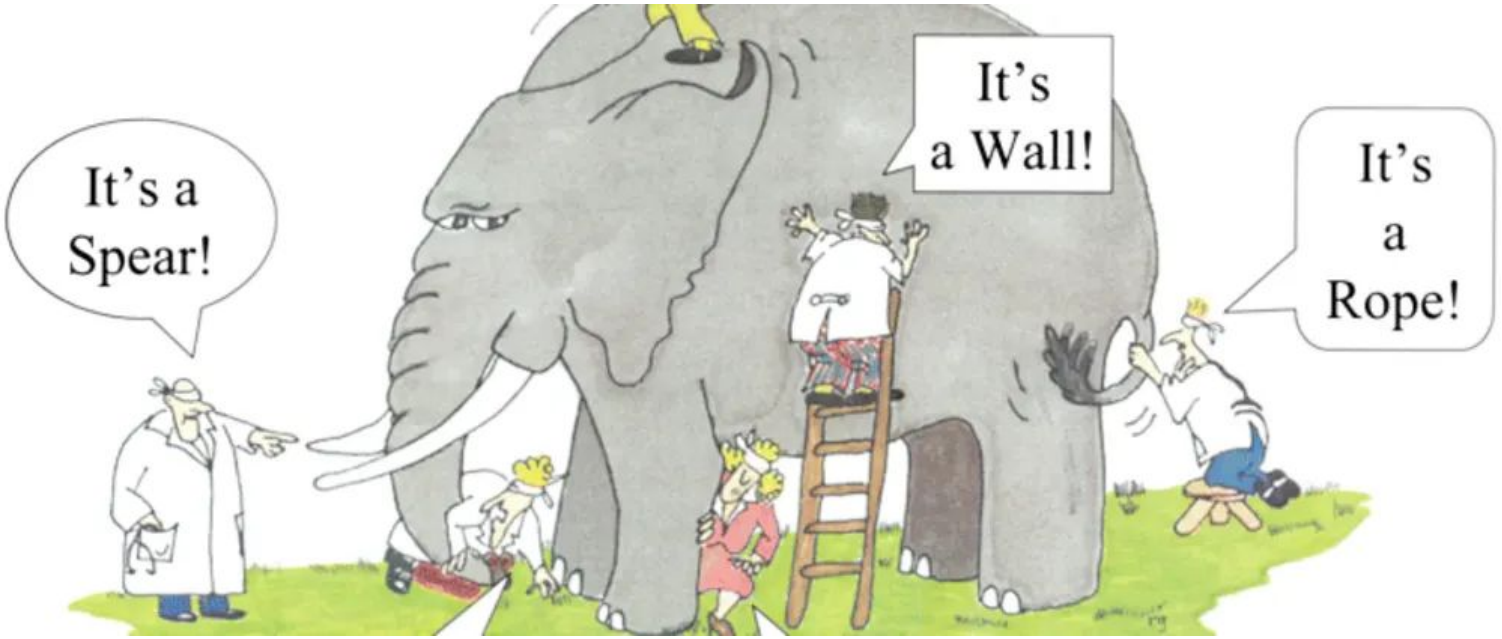
- Uma única fonte de "verdade"
  - Não há dúvida sobre qual repositório possui a versão mais atualizada de um arquivo
- Incentivam reúso de código
  - Acesso rápido a qualquer arquivo, de qualquer sistema
- Mudanças são atômicas
  - 1 commit pode alterar n sistemas
- Facilitam refactorings em larga escala



Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

Desvantagem de Monorepos

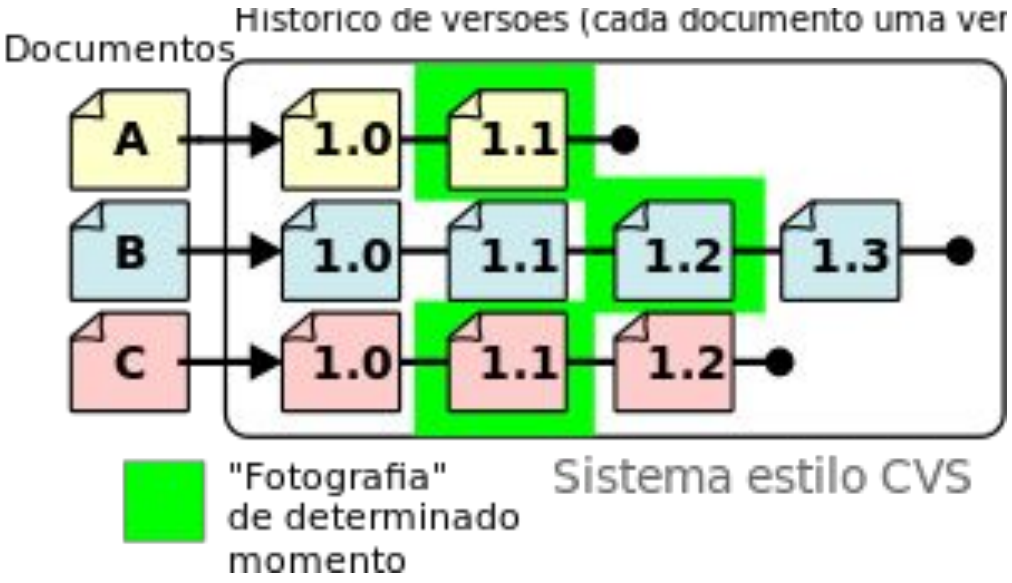
- Podem requerer ferramentas customizadas (IDEs, etc)
- Cada desenvolvedor terá em seu repositório local todos os arquivos de todos os sistemas da organização



Fonte: Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020

Versionamento Semântico

- Conjunto de políticas para gerenciar as diversas versões de um sistema
  - Útil para o gerenciamento de dependências
- Como vocês fazem para definir a versão do software?



Fonte: https://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%B5es

Versionamento Semântico

- Dado um número de versão MAJOR.MINOR.PATCH, incremente a:
  - versão Maior (MAJOR): quando fizer mudanças incompatíveis
  - versão Menor (MINOR): quando adicionar funcionalidades mantendo compatibilidade
  - versão de Correção (PATCH): quando corrigir falhas mantendo compatibilidade

1 . 12 . 7

major
incompatibilidade

minor
novas funcionalidades

patch
correções de bugs

Fonte: https://semver.org/lang/pt-BR/ e http://missaodevops.com.br/docs/scm_efetivo.html

Versionamento Semântico

- Precedência é determinada pelos identificadores da esquerda para direita
  - 1.0.0 < 2.0.0 < 2.1.0 < 2.1.1
- Um número de versão normal DEVE ter o formato de X.Y.Z, onde X, Y, e Z são inteiros não negativos, e NÃO DEVE conter zeros à esquerda
- A versão de Correção DEVE ser redefinida para 0(zero) quando a versão Menor for incrementada
- Versão de Correção e Versão Menor DEVEM ser redefinidas para 0(zero) quando a versão Maior for incrementada
- No início do desenvolvimento, a versão Maior DEVE ser zero (0.y.z). Qualquer coisa PODE mudar a qualquer momento

Fonte: https://semver.org/lang/pt-BR/

Git

- Sistema de controle de versão distribuído
  - https://git-scm.com/

Entendendo o Git

- Pasta .git
  - Diretório raiz do projeto
  - Contém todos os objetos, commits e configurações do projeto
  - .git/config: arquivo com configurações específicas do repositório
- .gitignore
  - Indica os arquivos que devem ser ignorados
    - Exemplo:
      - executáveis
      - arquivos gerados durante a compilação
      - arquivos gerados durante execução dos testes
  - https://gitignore.io/

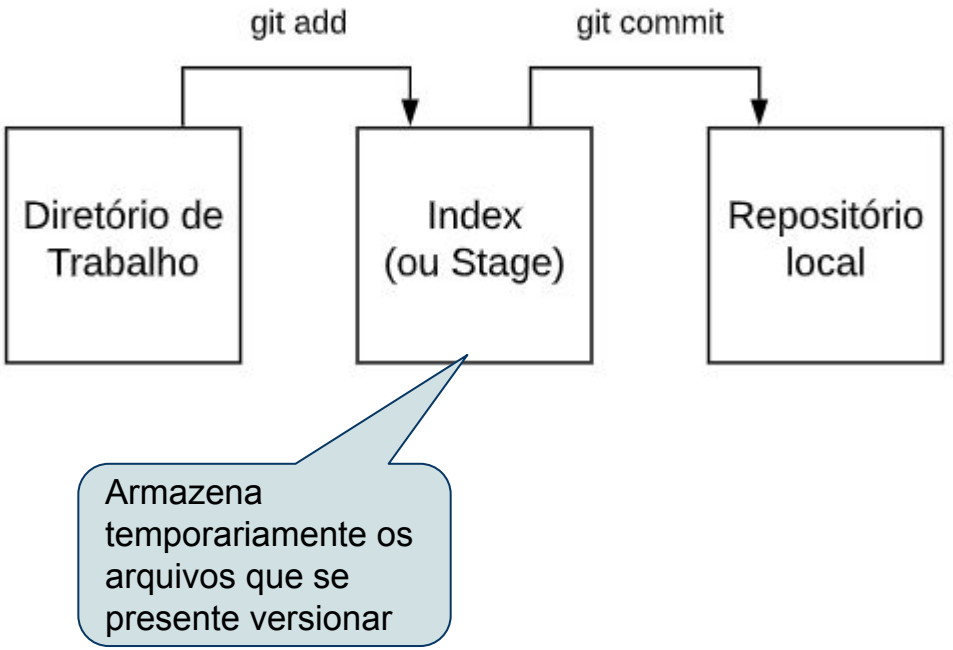
Comandos Básicos do Git

- Criar um repositório local vazio
  - **git init**
- Clonar um repositório
  - **git clone <nome do repositório>**
    - *ele cria o repositório vazio e depois copiar todos os commits do repositório remoto*

Fonte: https://rubygarage.org/blog/most-basic-git-commands-with-examples

Comandos Básicos do Git

- Adicionar um ou mais arquivos
  - **git add <nome-arquivo>**
  - **git add *.extensão**
  - **git add ***



Fonte: https://engsoftmoderna.info/capAp.html

Comandos Básicos do Git

- Confirmar alterações
  - **git commit**
  - **git commit -m “sua mensagem”**
  - **git commit -a -m “sua mensagem”**

```
1   $ git commit -a -m "Do something once more"
```

adding-and-committing-files-to-repository.sh hosted with ❤ by GitHub

Fonte: https://rubygarage.org/blog/most-basic-git-commands-with-examples

Comandos Básicos do Git

- Remover arquivos do repositório
  - **git rm arquivo.extensão**
  - **git commit**

```
1   $ git rm --cached my-file.ts
```

removing-a-file-from-staging-area.sh hosted with ❤ by GitHub

Fonte: https://rubygarage.org/blog/most-basic-git-commands-with-examples

Comandos Básicos do Git

- Verificar alterações
  - **git status**
    - Mostra o estado do diretório de trabalho e do index

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Teste01.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Teste01.txt
```

Fonte: https://rubygarage.org/blog/most-basic-git-commands-with-examples

Comandos Básicos do Git

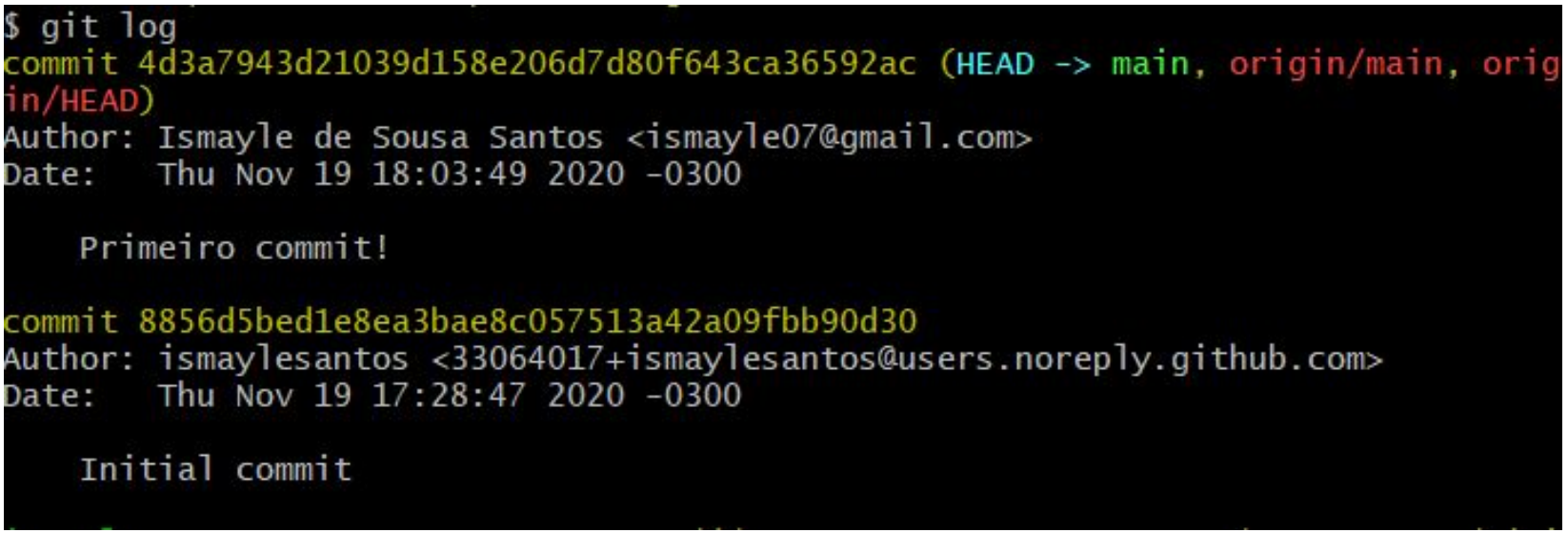
- Verificar alterações
  - **git diff**
    - Destaca as modificações realizadas nos arquivos do diretório de trabalho

```
$ git diff
diff --git a/Teste01.txt b/Teste01.txt
index e69de29..8f0746a 100644
--- a/Teste01.txt
+++ b/Teste01.txt
@@ -0,0 +1 @@
+Teste 2
\ No newline at end of file
```

Fonte: https://rubygarage.org/blog/most-basic-git-commands-with-examples

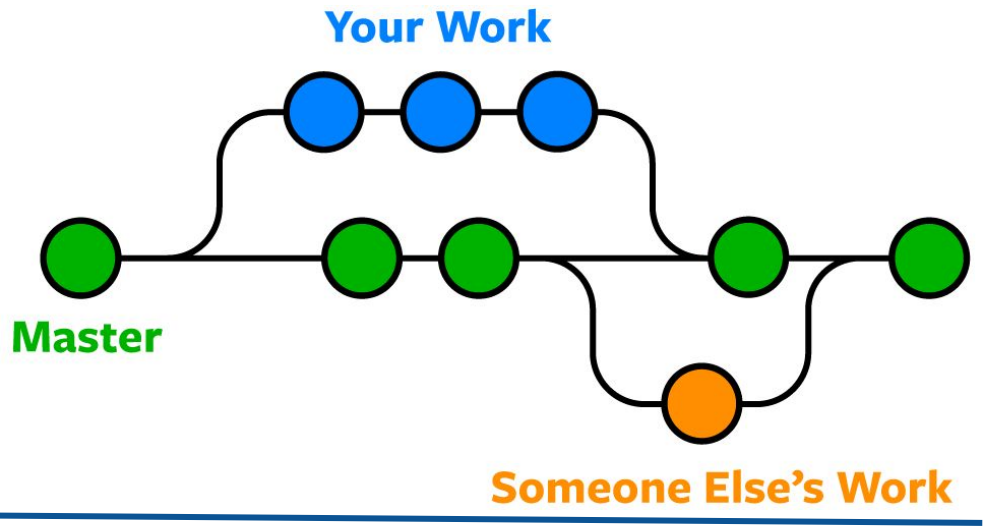
Comandos Básicos do Git

- Histórico de commits
  - **git log**
    - Lista informações sobre últimos commits



Comandos Básicos do Git

- Todo repositório tem uma branch default (master)
- Criando uma nova ramificação (branch)
  - **git branch <nome>**
- Alterando ramificação
  - **git checkout <nome>**
- Visualizando ramificações
  - **git branch**



Fonte: https://www.nobledesktop.com/learn/git/git-branches

Comandos Básicos do Git

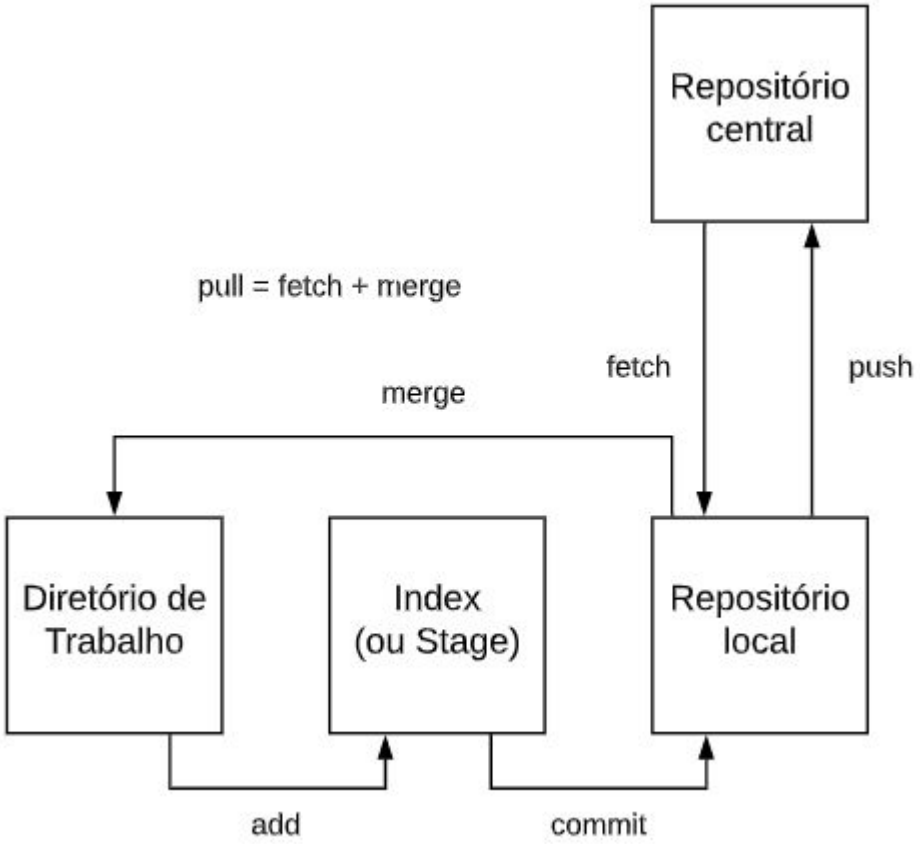
- Mesclar ramificação com a master
  - **git checkout master**
  - **git merge <branchName>**
    - a master vai receber as modificações da branch **branchName**

Conflitos de merge tem que ser resolvidos manualmente

Comandos Básicos do Git

Push & Pull

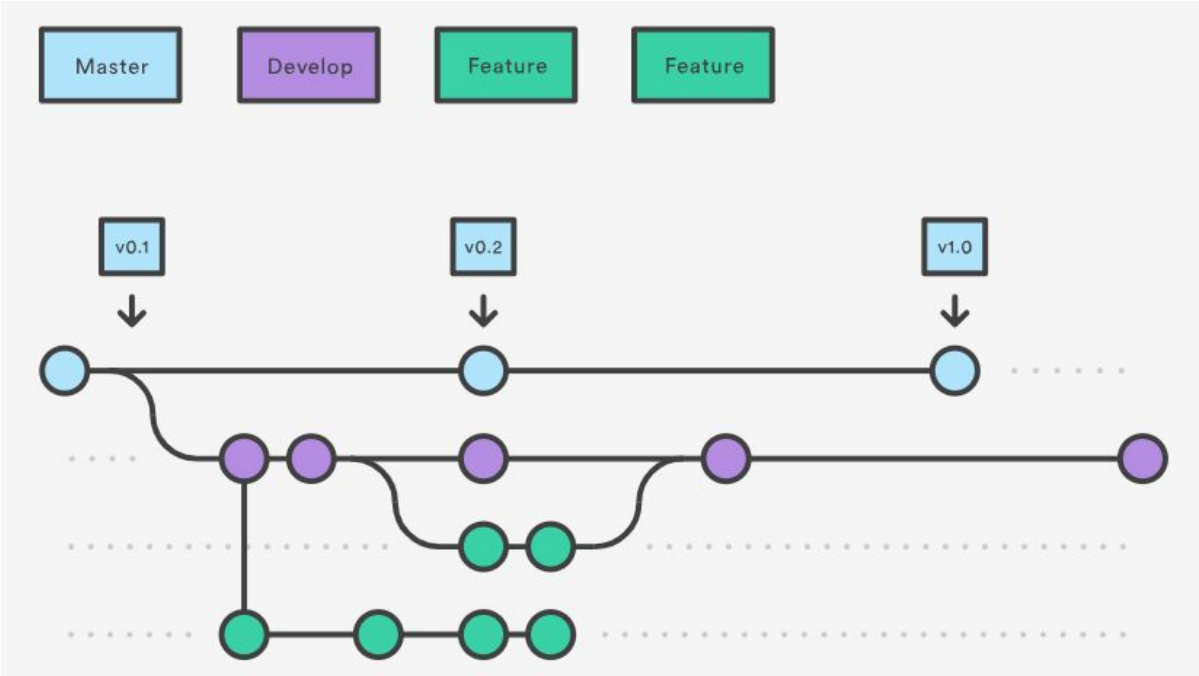
- Push
  - copia os commits mais recentes do repositório local para o repositório remoto
- Pull
  - copia os arquivos do repositório central para o repositório local (fetch)
  - atualiza os arquivos do diretório de trabalho (merge)



Fonte: https://engsoftmoderna.info/capAp.html

GitFlow

- É um design de fluxo de trabalho Git
- Ideal para projetos que têm um ciclo de lançamento agendado



Fonte: https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow

Bizu do Git

Git: configurations

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email@email-provider.com"
$ git config --global color.ui true
$ git config --list
```

Git: starting a repository

```
$ git init
$ git status
```

Git: staging files

```
$ git add <file-name>
$ git add <file-name> <another-file-name> <yet-another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

Git: committing to a repository

```
$ git commit -m "Add three files"
$ git reset --soft HEAD^
$ git commit --amend -m <enter your message>
```

Git: pulling and pushing from and to repositories

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

Git: branching

```
$ git branch
$ git branch <branch-name>
$ git checkout <branch-name>
$ git merge <branch-name>
$ git checkout -b <branch-name>
```

Fonte: https://rubygarage.org/blog/most-basic-git-commands-with-examples

Licenças

Qual das seguintes alternativas melhor descreve sua situação?

Eu quero uma licença simples e permissiva

A **Licença MIT** é uma licença permissiva que é concisa e vai direto ao ponto. Ela permite que as pessoas façam o que quiserem com seu código, desde que forneçam uma atribuição de volta para você e não lhe responsabilize.

jQuery e **Rails** usam a Licença MIT.

Eu estou preocupado com patentes

A **Licença Apache** é uma licença permissiva, similar à Licença MIT, mas que também provê uma concessão expressa de direitos de patente dos contribuintes para os usuários.

Apache, **SVN**, e **NuGet** usam a Licença Apache.

Eu me preocupo em compartilhar melhorias

A GPL (**V2** ou **V3**) é uma licença "copyleft" que exige que quem distribui o seu código ou uma obra derivada deve disponibilizar o fonte sob os mesmos termos.

Linux, **Git**, e **WordPress** usam GPL.

Fonte: http://escolhaumalicenca.com.br/

README.md

- É um arquivo de texto que introduz e explica um projeto
  - Nome, instalação, Suporte, Contribuição, Autores, etc
- Exemplo:

Foobar

Foobar is a Python library for dealing with word pluralization.

Installation

Use the package manager pip to install foobar.

```
pip install foobar
```

Usage

```
import foobar

foobar.pluralize('word') # returns 'words'
foobar.pluralize('goose') # returns 'geese'
foobar.singularize('phenomena') # returns 'phenomenon'
```

Fonte: https://www.makeareadme.com/

Conventional Commits

- Uma especificação para dar significado legível às mensagens do commit
  - Criação automatizada de CHANGELOGs
  - Melhora a legibilidade do histórico de commits
- Estrutura de uma mensagem de commit

```
<tipo>[escopo opcional]: <descrição>

[corpo opcional]

[rodapé(s) opcional(is)]
```

Fonte: https://www.conventionalcommits.org/pt-br/v1.0.0/

Conventional Commits

- Tipos
  - feat
    - nova feature para usuário
  - fix
    - correção de bug
  - refactor, test, docs ...
- Rodapé
  - BREAKING CHANGE
    - modificação que quebra a compatibilidade da API

```
feat: permitir que o objeto de configuração fornecido estenda outras configurações

BREAKING CHANGE: a chave `extends`, no arquivo de configuração, agora é utilizada para estender outro arquivo de configuraçã
```

Fonte: https://www.conventionalcommits.org/pt-br/v1.0.0/

ChangeLog

- Changelog
  - É um arquivo que contém uma lista selecionada, ordenada cronologicamente, de mudanças significativas para cada versão do projeto

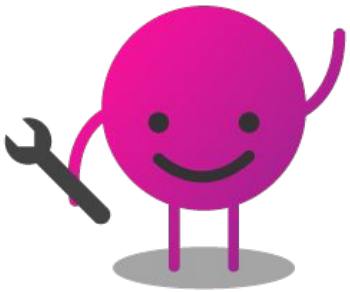
```
## [0.3.0] - 2015-12-03
### Added
- RU translation from [@aishek](https://github.com/aishek).
- pt-BR translation from [@tallesl](https://github.com/tallesl).
- es-ES translation from [@ZeliosAriex](https://github.com/ZeliosAriex).

## [0.2.0] - 2015-10-06
### Changed
- Remove exclusionary mentions of "open source" since this project can
benefit both "open" and "closed" source projects equally.
```

Fonte: https://keepachangelog.com/pt-BR/1.0.0/

ChangeLog

- Princípios Fundamentais
  - Deve haver uma entrada para cada versão
  - Alterações do mesmo tipo devem ser agrupadas
  - Versão mais recente vem em primeiro lugar
  - Data de lançamento de cada versão é exibida
  - Mencione se você segue o versionamento semântico



changelog

Fonte: https://olivia-ai.org/changelog

ChangeLog

- Tipos de Mudanças (* *sugestão*)
  - **Added** (Adicionado)
    - novos recursos
  - **Changed** (Modificado)
    - alterações em recursos existentes
  - **Deprecated** (Obsoleto)
    - recursos que serão removidos nas próximas versões
  - **Removed** (Removido)
    - recursos removidos nesta versão
  - **Fixed** (Corrigido)
    - correção de bugs
  - **Security** (Segurança)
    - em caso de vulnerabilidades

Fonte: https://keepachangelog.com/pt-BR/1.0.0/

Obrigado!

Por hoje é só pessoal...

Dúvidas?

qpg4p5x

ismaylesantos@great.ufc.br

@IsmayleSantos