Sumário

[Introdução ao Gerenciamento de Código Fonte 2](#_Toc520195282)

[Conceitos Importantes 2](#_Toc520195283)

[GIT 2](#_Toc520195284)

[O que é 2](#_Toc520195285)

[Terminologia 2](#_Toc520195286)

[Comandos 3](#_Toc520195287)

[Repositório 4](#_Toc520195288)

[Opções de Client para usar o Git 4](#_Toc520195289)

[Fluxos de Trabalho 5](#_Toc520195290)

[Pull Request 5](#_Toc520195291)

[Linguagem Java 6](#_Toc520195292)

[.Tipos primitivos 6](#_Toc520195293)

[Tipos de Referência (“pointers”) 6](#_Toc520195294)

# Introdução ao Gerenciamento de Código Fonte

*“Controlar cada peça de código”*

* Fontes de Programas
* Arquivos de Configuração
* Binários
* Dependências (bibliotecas, etc.)
* CI (Configuration Items)

## Conceitos Importantes

* Equipes e Paralelismo
* Branches
* Merges
* Conflitos
* Labels (Tags) *(Rótulo de um trecho de código alterado)*

## GIT

### O que é

* O GIT surgiu para gerenciar o código fonte do Linux, criado pelo próprio Linus Torvalds e é atualmente mantido Junio Hamano.
* Rastreia as mudanças em um conjunto de arquivos ao longo do tempo.
* Controle de Versão

### Terminologia

* Branches: “Galhos” das alterações realizadas.
* Commit: Uma árvore (tree) de alterações no projeto.
  + Commiter: Aquele que enviou o Commit.
  + Commit Identification: Idem Autor, aquele que realizou as alterações.
  + Commit Message: A mensagem do Commit.
  + Commit Graph: Exibição de todas as Branchs do projeto e os seus autores.
  + Parent Commit: A lista de referência de outros Commits.
* Harsh: Código embaralhado e compactado gerado a partir de um código.
* Repositório: Pasta no Sistema de Arquivos.
* Merge: Junta versões de código
* Rebase: Resolve conflitos e remove histórico de alterações

### Comandos

* git ➲ Abre o Git no terminal do Windows (cmd, promt).
* git init ➲ Inicia o repositório do Git na pasta selecionada.
* git config ➲ Altera uma configuração.
* git status ➲ Mostra tudo que tem e que foi alterado ou excluído
* git add ➲ Adiciona um arquivo ou mais para o Stage.

***Alteração > Stage Area > Commit > Unstage Area***

* git add . ➲ Adiciona todos os arquivos alterados para o Stage.
* git reset <file> //move from Stage area to Unstage area ➲ Move uma arquivo do Stage para fora do Stage.
* git commit -m “Message commit” ➲ Cria o commit com uma mensagem.
* git log ➲ Mostrar o log
* git checkout <commit id> ➲ Checa o commit com o seguinte ID
* git reset –hard <commit id> ➲ Aborta todas as alterações feitas por tal ID
* git clone <repo adress> ➲ Clona o repositório
* git branch ➲ Exibe a branch atual
* git merge ➲ Junta uma branch com a outra
* git pull ➲ Pega as alterações feitas
* git push ➲ Envia as alterações
* git push -u origin <nome da branch> ➲ Envia as alterações feitas em uma branch para a Master

### Repositório

Pode-se clonar um repositório remoto para o local. (*git clone*)

* **Pull:**  Local *pega do* Remoto
* **Push:** Local *para o*  Remoto

### Opções de Client para usar o Git

* GitHub (O que todo mundo usa)
* BitBucket
* GitKraken
* Git Extension
* SmartGit
* SourceTree (O melhor que tem para Windows)
* TortoiseGit

### Fluxos de Trabalho

#### Workflow Centralized

* Uma única branch Master.
* Rebase usado constantemente para solucionar conflitos.
* Fluxo centralizado.

#### GitFlow

* Foco em desenvolvimento paralelo, com várias branch.
  + Fluxo de hotfix: As branchs de hotfix onde os bugs são corrigidos de forma emergencial. Pode ocorrer várias vezes em paralelo com o fluxo de developer.
  + Fluxo de developer: A branch principal para desenvolvimento da nova versão release.
  + Branch de feature: Quando pronta, se junta a branch de developer.
* Utilizado no CPB.

### Pull Request

Um revisor analisa e decide se aprova ou se requisita a mudança em uma solicitação de alteração de código.

# Linguagem Java

*“Write once, run anywhere”*

Criado para rodar sobre uma Máquina Virtual para reaproveitar o código ao invés de escrever um código diferente para cada máquina. Portanto Java é uma linguagem interpretada, ao invés de compilada como C/C++.

* Orientada a objeto
* Vários recursos poderosos
  + Anotation, Generics (Template), Reflection, lambda, Streams, Multithread, Distribuída, Portável, Perfomática.
* Madura, já possui diversas APIs e Frameworks.
* Recursos avançados para desenvolver vários tipos de aplicação
  + Web, Standalone, API, Rest, GUI, Mobile, etc.
* **Compila o código fonte para ByteCode**, um código intermediário **capaz de ser interpreta pela JVM**
* **Não possui aritmética de ponteiro**, portanto sendo mais simples então.

### .Tipos primitivos

* int : Inteiro 32bits -231 a 231-1
* short : Inteiro 8bits -128 a 127
* Byte : Inteiro 16bits -32768 a 32767
* char : 16bits Unicode Pode representar qualquer valor Unicode com “\u0000” até “\uffff” (65535)
* float: 32bits **Não deve ser usado para alta precisão.**
* double: 64bits **Não deve ser usado para alta precisão.**
* boolean: 1bit True or False

Tanto o byte quanto o short podem economizar memória em grandes arrays no lugar do int.

### Tipos de Referência (“pointers”)

* Object
* String
* User Custom Types (Tipos customizados pelo Programador)