FENIX

# GIT – Sistema de Controle de Versões

Versão 1.0

22/02/2019

Número do Documento: 01

Conteúdo

[GIT – Sistema de Controle de Versões 1](#_Toc1991715)

[Lista de Tabelas 2](#_Toc1991716)

[1. Introdução 3](#_Toc1991717)

[2. Propósito do Documento 4](#_Toc1991718)

[3. Metodologia 5](#_Toc1991719)

[4. Referências 6](#_Toc1991720)

[5. Dicas úteis 10](#_Toc1991721)

[6. Fluxo de Trabalho 11](#_Toc1991722)

[7. Git Flow 12](#_Toc1991723)

Lista de Tabelas

[**Tabela 1** - Registro de Mudanças 1](#_Toc781375)

[**Tabela 6** – Historico de Revisão do Modelo 1](#_Toc781380)

## Introdução

Neste documento, será esplanado características técnicas e requisitos para um padrão de código visando a qualidade na entrega dos softwares desenvolvidos pelo IPDA.

## Propósito do Documento

Este documento visa padronizar a forma como são criados novos códigos fonte pelos desenvolvedores do IPDA.

## Metodologia

A metodologia utilizada será o *clean code*, desenvolvida para criar padrões de código fácil de entender, fácil de modificar e fácil de testar.

Também utilizaremos os principais conceitos do SOLID, cinco princípios de um bom código orientado a objetos. Resumidamente os princípios são:

## Referências

|  |  |
| --- | --- |
| **Instalação Linux Debian/Ubuntu:** | **Comentários:** |
| # apt-get install git | Instala o git |
| # git –version | Verifica a versão instalada |

|  |  |
| --- | --- |
| **Novo repositório:** | **Comentários:** |
| # git init | Inicializa o git |
| # git clone nome\_repositório | Clona um repositório em um novo diretório |

|  |  |
| --- | --- |
| **Add e Salvar:** | **Comentários:** |
| # git add .  # git add nome\_arquivo | Add os arquivos na area de stage. O . add todos. |
| # git status | Por padrão, o status git atualizará automaticamente o índice, atualizando as informações de estatística armazenadas em cache da árvore de trabalho e gravando o resultado. |
| # git commit –m “Comentário” | Crie um novo commit contendo o conteúdo atual do índice e a mensagem de log fornecida descrevendo as mudanças. O novo commit é um filho direto de HEAD.  -m “Comentário” |

|  |  |
| --- | --- |
| **Enviando as alterações:** | **Comentários:** |
| # git remote add origin <servidor> | * Se você não clonou um repositório existente e quer conectar seu repositório a um servidor remoto, você deve adicioná-lo com |
| # git push origin master | * Envia as alterações ao seu repositório remoto. |

|  |  |
| --- | --- |
| **Ramificando:** | **Comentários:** |
| * Branches ("ramos") são utilizados para desenvolver funcionalidades isoladas umas das outras. O branch master é o branch "padrão" quando você cria um repositório. Use outros branches para desenvolver e mescle-os (merge) ao branch master após a conclusão.   gir_branches.png | |
| # git checkout -b nome\_nova\_funcionalidade\_x | crie um novo branch "nome\_nova\_funcionalidade\_x" e selecione-o usando –b |
| # git branch | Lista as branches |
| # git branch –v | Lista a branch com seu último commit |
| # git checkout master | retorne para o master |
| # git branch -d nome\_branch | Remover uma branch |
| # git push origin <nome\_branch> | Um branch não está disponível a outros a menos que você envie o branch para seu repositório remoto. |

|  |  |
| --- | --- |
| **Atualizar e Mesclar:** | **Comentários:** |
| # git pull | Para atualizar seu repositório local com a mais nova versão. |
| # git merge nome\_branch | Junte dois ou mais históricos de desenvolvimento juntos |
| git diff <branch origem> <branch destino> | Antes de fazer o merge das alterações, você pode também pré-visualizá-as. |

|  |  |
| --- | --- |
| **Rotulando:** | **Comentários:** |
| # git log | Mostra os log dos commits |
| # git tag 1.0.0 | Identificar a branch |
| git diff <branch origem> <branch destino> | Antes de fazer o merge das alterações, você pode também pré-visualizá-as. |

|  |  |
| --- | --- |
| Sobrescrever alterações locais**:** | **Comentários:** |
| # git checkout -- <arquivo> | * No caso de você ter feito algo errado você pode sobrescrever as alterações locais. * isto substitui as alterações na sua árvore de trabalho com o conteúdo mais recente no HEAD. Alterações já adicionadas ao index, bem como novos arquivos serão mantidos. |
| # git pull  # git fetch | * De modo simplório, o git fetch buscas as diferenças em relação ao ramo atual, mas não altera nada nesse branch. Já git pull faz o git fetch e faz o merge das diferenças. |
| # git reset --hard origin/master | * Se ao invés disso você deseja remover todas as alterações e commits locais, recupere o histórico mais recente do servidor e aponte para seu branch master local. |

## Dicas úteis

* Interface gráfica padrão

# gitk

* usar saídas do git coloridas

# git config color.ui true

* exibir log em apenas uma linha por commit

# git config format.pretty oneline

* fazer inclusões interativas

# git add -i

## Fluxo de Trabalho

Seus repositórios locais consistem em três "árvores" mantidas pelo git. a primeira delas é sua Working Directory que contém os arquivos vigentes. A segunda Index/Stage que funciona como uma área temporária e finalmente a HEAD que aponta para o último commit (confirmação) que você fez.



## Git Flow

Apêndice A: Registro de Alterações

Tabela 1 - Registro de Mudanças

| Número da versão | Data | Autor/Proprietário | Descrição da Alteração |
| --- | --- | --- | --- |
| 1.0 | 22/02/2019 | IPDA | Versão inicial |
|  |  |  |  |

Apêndice C: Histórico de Revisão de Documento

Tabela 6 – Historico de Revisão do Documento

| Numero da Versão | Data | Autor/Proprietário | Descrição da Alteração |
| --- | --- | --- | --- |
| 1.0 | 22/02/2019 | Fagner Lira | Criação do Documento GIT |