

DOCUMENTO DESCRITIVO DE

Controle de Versionamento do Portal PoP-RN – CVCS



Desenvolvido por:

André Gomes – Análise e Desenvolvimento de Sistema

Douglas Braz – Ciência da Computação

Maxwel da Silva Santos – Engenharia Elétrica

Natal/RN

Maio/2017

Sumário

Introdução.....	3
Instalação do Git	3
Passo 1 - Criar Repositório Local.....	4
Passo 2 – Clonar Repositório Remoto (s8-homepage-pop-rn)	4
Passo 3 – Criar Branch	5
Passo 4 - Definir Branch HEAD para Branch de Desenvolvimento	6
Passo 5 – Adicionar e Commitar em um Branch de Desenvolvimento	7
Passo 6 – Realizar Pull no Repositório Remoto.....	9
Passo 7 – Realizar Merge (branch master local <-> branch de desenvolvimento).....	10
Convergências e Conflitos entre Branchs no momento do Merge.....	10
Passo 8 – Realizar Push do Repositório Local com o Repositório Remoto	13
Passo 9 – Deletar Branch de Tarefas após PUSH com Repositório Remoto	13
Fluxograma	15
Conclusão	16

Introdução

O controle de versionamento do nosso projeto será realizado através da ferramenta CVCS Git. Demonstraremos em um diagrama abaixo como se dá o controle, objetivando facilitar a compreensão do fluxo de trabalho adotado por nossa equipe.

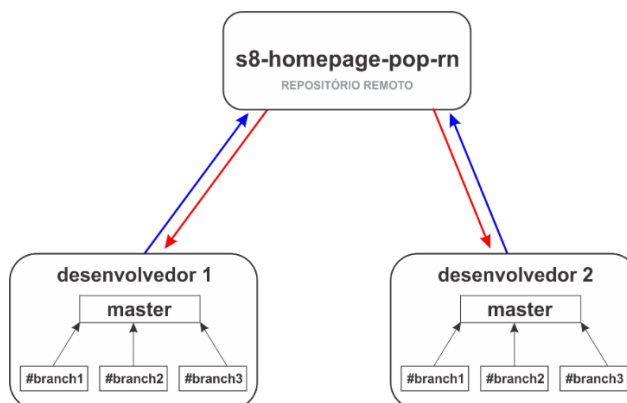


Figura 1 – Fluxo de trabalho de desenvolvimento do projeto com Git.

Como podemos perceber no diagrama, o repositório remoto apenas recebe atualizações (PUSH's) originadas a partir dos terminais de desenvolvimento e fornece às mesmas (PULL's) de forma direta aos desenvolvedores, mantendo o tratamento de conflitos e convergências nas máquinas de desenvolvimento, para que sejam resolvidos em seus repositórios locais, desse modo os commits locais poderão ser atualizados no repositório remoto sem problemas.

Instalação do Git

Caso o Git não esteja instalado no pc do desenvolvedor, abaixo segue link com instruções de instalação e configuração.

Acesse: http://rogerdudler.github.io/git-guide/index.pt_BR.html

QR-CODE:



Listaremos abaixo os passos necessários para versionarmos nosso projeto, partindo desde a criação do repositório local até as rotinas de branches, commits, merges, push e pull.

Passo 1 – Criar Repositório Local

Supondo que ainda não haja um repositório criado na máquina local, iremos através do terminal (acesso como administrador) acessar a pasta pública **www** ou **public_html** do nosso servidor web local (XAMMP, WAMP, etc), e realizarmos a inicialização do git.

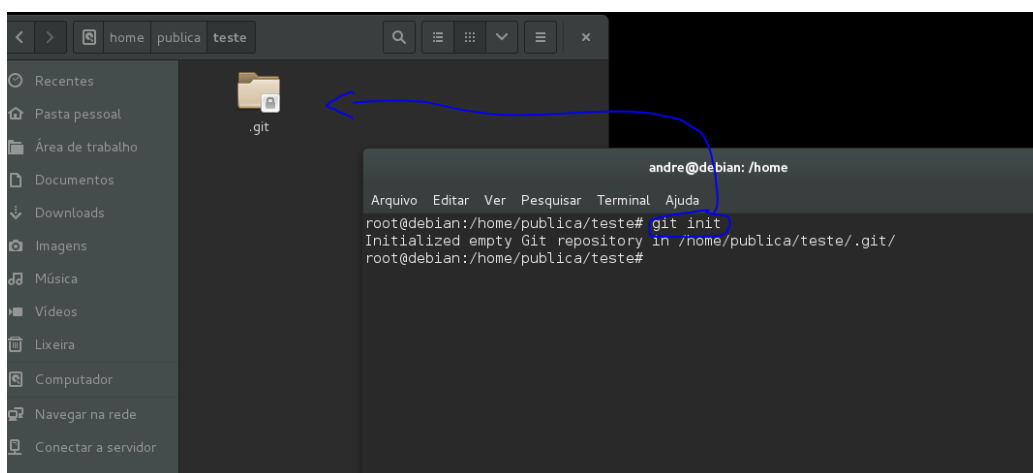


Figura 2 – Repositório sendo criado através do comando **git init** (porém ainda vazio).

Passo 2 – Clonar Repositório Remoto (s8-homepage-pop-rn)

Uma vez que o repositório local tenha sido criado, devemos clonar o projeto do nosso repositório remoto. Este procedimento deve ser executado uma única vez, na ocasião da criação do repositório vazio em um terminal de desenvolvimento. Através do comando: **git clone** <http://ativpop.pop-rn.rnp.br/git/s8-homepage-pop-rn>. O terminal solicitará autenticação, que corresponde ao login de usuário Pop-RN previamente autorizado pela gerência do projeto.

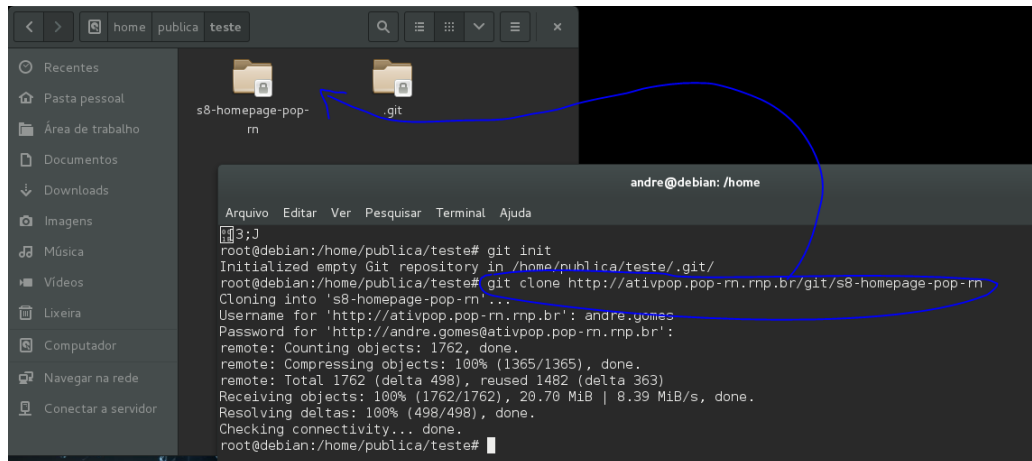


Figura 3 – Repositório sendo clonado do repositório remoto para repositório local.

Passo 3 – Criar Branch

Com o terminal aberto na pasta do projeto, criaremos branches. O branch (ramo) é uma espécie de galho ou desvio do código principal para concentrarmos as tarefas de modo mais específico. Usaremos bastante ao longo do desenvolvimento da aplicação, a fim de isolarmos determinadas tarefas/funcionalidades em desenvolvimento do restante do conteúdo que já esteja em produção ou sendo desenvolvido por um outro membro da equipe. Os nomes dos branches devem ser sugestivos identificando tarefas/funcionalidades em desenvolvimento.

Usamos o comando **git checkout -b "nomeBranchSemAspas"** para criar um novo branch. Observe na figura abaixo que criamos 2 branches (*Tarefa_Departamentos* e *Tarefa_Conteudos*) e em seguida utilizamos o comando **git branch** para listar os branches existentes no projeto.

Podemos verificar que ao criar um branch com a instrução acima, o Git além de criar o branch, define o **HEAD** (branch ativo) como sendo este novo branch recém-criado. Podemos criar mais de um branches e trabalharmos simultaneamente em mais de um branch de implementando funcionalidades diversas.

```
Administrador: Prompt de Comando

C:\wamp\www\s8-homepage-pop-rn>git checkout -b Tarefa_Departamentos
Switched to a new branch 'Tarefa_Departamentos'

C:\wamp\www\s8-homepage-pop-rn>git checkout -b Tarefa_Conteudos
Switched to a new branch 'Tarefa_Conteudos'

C:\wamp\www\s8-homepage-pop-rn>git branch
* Tarefa_Conteudos
  Tarefa_Departamentos
  master

C:\wamp\www\s8-homepage-pop-rn>
```

Figura 4 – Tela do terminal criando e verificando os branches do projeto.

Passo 4 - Definir Branch HEAD para Branch de Desenvolvimento

É importante frisar que todo update no código da aplicação deve ser feito por meio de branches, e o **HEAD**, é o apontador para o branch ativo do sistema. No momento do desenvolvimento este branch deve ser diferente do branch **master**, a fim de evitar problemas na integração dos repositórios locais com o repositório remoto em **convergências e conflitos**.

Por este motivo, antes de adicionar e commitar, é importante verificar se o **HEAD** (branch ativo) está definido para o branch de desenvolvimento desejado (comando **git branch**). Deste modo, uma vez que os branches estejam criados e o desenvolvimento da aplicação esteja em curso, poderemos adicionar os conteúdos a serem trackeados e em seguida commitados sem problemas.

Para alternar de um branch para outro usamos o comando: **git checkout "nomeBranchSemAspas"** em seguida podemos conferir a mudança com o comando **git branch**, conforme figura abaixo.

```
C:\wamp\www\s8-homepage-pop-rn>git branch
Tarefa_Conteudos
Tarefa_Departamentos
* master

C:\wamp\www\s8-homepage-pop-rn>git checkout Tarefa_Conteudos
Switched to branch 'Tarefa_Conteudos'

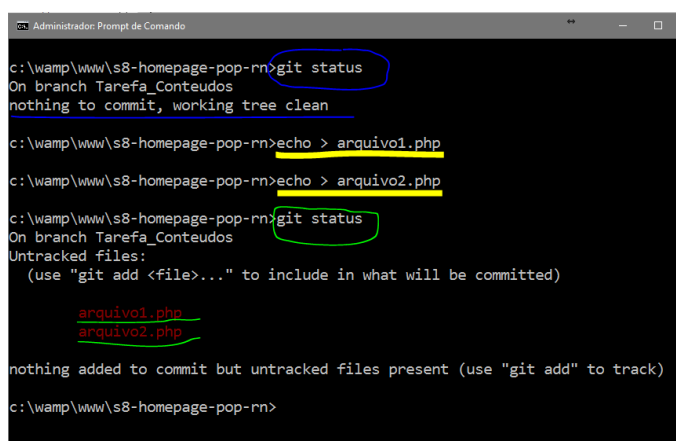
C:\wamp\www\s8-homepage-pop-rn>git branch
* Tarefa_Conteudos
  Tarefa_Departamentos
  master
```

Figura 5 – Alternância entre branches de desenvolvimento.

Observe na figura 5, que usamos o **git branch** para listar os branches existentes em nosso repositório, bem como o branch ativo, no caso da primeira execução do comando, o **máster** (destacado em verde). Em seguida, alteramos o **HEAD** para **Tarefa_Conteudos** através do comando **git checkout Tarefa_Conteudos**, e por fim, listamos novamente e podemos conferir que o **HEAD** alterou de **master** para **Tarefa_Conteudos**.

Passo 5 – Adicionar e Commitar em um Branch de Desenvolvimento

Para listar os arquivos que foram modificados no branch ativo, basta usar o comando **git status**. Observe na figura abaixo, que no primeiro **git status**, não há nada a ser adicionado. Em seguida, criamos dois arquivos de teste, *arquivo1.php* e *arquivo2.php* e repetimos o comando **git status**. Desta vez ele nos mostra que os arquivos precisam ser adicionados para poderem ser comitados.



```
Administrador: Prompt de Comando

c:\wamp\www\s8-homepage-pop-rn>git status
On branch Tarefa_Conteudos
nothing to commit, working tree clean

c:\wamp\www\s8-homepage-pop-rn>echo > arquivo1.php

c:\wamp\www\s8-homepage-pop-rn>echo > arquivo2.php

c:\wamp\www\s8-homepage-pop-rn>git status
On branch Tarefa_Conteudos
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    arquivo1.php
    arquivo2.php

nothing added to commit but untracked files present (use "git add" to track)
c:\wamp\www\s8-homepage-pop-rn>
```

Figura 6 – Consultando status dos arquivos do repositório.

Para adicionarmos os arquivos no branch atual (Tarefa_Conteudos) usamos o comando **git add ***. Observe na figura abaixo, que após adicionarmos os conteúdos para commit, invocamos o **git status** para conferir o estado de nossos novos arquivos. Podemos perceber que o Git nos mostra que os arquivos estão devidamente adicionados, restando apenas serem comitados.

```
c:\wamp\www\s8-homepage-pop-rn>git add *

c:\wamp\www\s8-homepage-pop-rn>git status
On branch Tarefa_Conteudos
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   arquivo1.php
    new file:   arquivo2.php
```

Figura 7 – Arquivos já adicionados no branch aguardando commit.

Agora podemos realizar o commit do nosso branch através do comando `git commit -m "DescriçãoDoCommitComAspas"`. Esta descrição deve ser sucinta, no entanto sugestiva e objetiva, retratando a tarefa/funcionalidade abrangida pelo branch ativo. Na figura abaixo, realizamos o commit dos arquivos adicionados.

```
c:\wamp\www\s8-homepage-pop-rn>git commit -m "#Tarefa 01 - Gerenciamento de Conteudos"
[Tarefa_Conteudos c998181] #Tarefa 01 - Gerenciamento de Conteudos
2 files changed, 2 insertions(+)
create mode 100644 arquivo1.php
create mode 100644 arquivo2.php
```

Figura 8 – Commit de arquivos do branch ativo do repositório.

```
C:\wamp\www\s8-homepage-pop-rn>git log --oneline --decorate --graph
* c998181 (HEAD -> Tarefa_Conteudos) #Tarefa 01 - Gerenciamento de Conteudos
* ccc9da7 (origin/master, origin/HEAD, master, Tarefa_Departamentos) Inicio da criação do contro
lador e das Views pertencentes ao campo extras
* 1d03b6a correcao de conflito

* 4b80625 commit f6
* | 29d1356 f22 testando
* |
* | 0937cf4 Merge branch 'f5'
* |
* | d359083 commit f5
* | 924907f Merge branch 'f1'
* |
* | 8f0a471 commit f1
* | 0ff3d24 Merge branch 'f4'
* |
* | 93d31fa commit f4
* | d235209 Teste teste
* |
* | 4da790a novo commit
* | ca2a08a commit feature3
* | 203f86f teste teste
```



```

* 3b6981f teste 22
* 7fdc22 comitando feature 2
* 7c079a9 Merge branch 'feature1'
* e712890 eeee
* 989a070 Teste novo
* 46505a9 novos testes
* a88ecb7 Teste funcao cc
* 73f385e funcao zz
* bed7b9d (origin/douglas) Teste Douglas
* 4d0f298 Alterações no arquivo newContentType.php e no controlador da listagem dos usuários
* e3c5a6c Introdução do uso do MVC no sistema
* 78628e2 Acrescimento da função adicionar novo conteúdo
* 55511f9 Preparacao de views do Backend - Tipos de Conteudos e comportamento do menu principal do Backend - 2
* f036c89 Preparacao de views do Backend - Tipos de Conteudos e comportamento do menu principal do Backend
* b37931c alteracao
* 5a732e9 Modificações na Estrutura Front e Backend
* 0d5ae90 Inclusões de pasta para teste - Pasta ./assets/docs
* 4e8426a Acoplagem do Backend no Projeto, divisao de Front e Back visando melhor organizacao
* 7731a91 Organizacao final da estrutura de diretorios
* 1f01a2e Bootstrap Theme Verso
* e52427d Incluído a pasta site
(END)

```

Figura 9 – Log dos commits realizados em forma de árvore, através do comando.

`git log --online --decorate --graph`

Passo 6 – Realizar Pull no Repositório Remoto.

Agora estamos com nosso branch de desenvolvimento pronto para ser integrado ao repositório de produção, realizaremos os últimos procedimentos necessários. Faremos a troca de branch, ou seja, alternaremos o **HEAD** para o branch master, através do comando **git checkout master**. Em seguida realizamos um pull no repositório remoto através do comando: **git pull** e recebemos a última versão do repositório remoto atualizado.

```

C:\wamp\www\s8-homepage-pop-rn>git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

C:\wamp\www\s8-homepage-pop-rn>git pull
Already up-to-date.

C:\wamp\www\s8-homepage-pop-rn>

```

Figura 10 – Troca do HEAD do repositório de desenvolvimento para o branch master, seguido do comando git pull, trazendo as atualizações do repositório remoto.

Passo 7 – Realizar Merge (branch master local <-> branch de desenvolvimento)

Após receber o conteúdo do nosso repositório remoto através do git pull, utilizaremos o comando merge para fundir as modificações feitas em nosso branch local com o conteúdo recebido através do pull. O Git faz isso muito bem e de maneira muito organizada, ele gerencia as convergências e pontua os conflitos para que possamos de maneira tranquila resolvê-los.

Para fazer o merge, nosso HEAD deve apontar para o branch master, através do comando `git merge "nomeDoBranchSemAspas"`. Nesse momento há uma fusão organizada de conteúdos, arquivos, linhas de códigos e etc. Caso haja convergência o Git gerencia e exibe uma mensagem no fim do merge indicando sucesso. Caso contrário exibirá uma mensagem contendo informações sobre conflitos.

```
C:\wamp\www\s8-homepage-pop-rn>git branch
Tarefa_Conteudos
Tarefa_Departamentos
* master

C:\wamp\www\s8-homepage-pop-rn>git merge Tarefa_Conteudos
Updating ccc9da7..c998181
Fast-forward
 arquivo1.php | 1 +
 arquivo2.php | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 arquivo1.php
 create mode 100644 arquivo2.php

C:\wamp\www\s8-homepage-pop-rn>
```

Figura 11 – Git Merge entre branch master local e o branch Tarefa_Conteudos

Na figura acima, o merge entre o branch master e o branch Tarefa_Conteudos **foi realizado com sucesso**. Quando obtemos sucesso no merge entre branches (ausência de conflitos), nos resta apenas a realização do push (próximo passo) das atualizações para o repositório remoto concluindo o processo de atualização. Caso o merge não apresente conflitos, passamos ao passo 8. O push é o passo final onde toda modificação realizada é sincronizada do branch master para o repositório remoto.

Convergências e Conflitos entre Branchs no momento do Merge.

Contudo, poderão ocorrer eventos adversos (convergências e conflitos) no momento do merge. A ferramenta Git faz a detecção desses eventos e em alguns casos consegue realizar o gerenciamento de forma elegante, tais como: *sincronização de arquivos existentes em repositórios diferentes, equalização de funções distintas implementadas por desenvolvedores diferentes em um mesmo arquivo.*

Essas ocorrências geralmente são gerenciadas pelo Git de forma automática sem que o desenvolvedor perceba. No entanto, há outros tipos de conflitos que o Git consegue identificar, porém não consegue resolvê-los. Para esses casos, ele notifica o arquivo e deixa a critério do programador resolver o conflito. Por exemplo, dois desenvolvedores distintos alteram no mesmo arquivo, uma mesma função com modificações diferentes. Veja no cenário apresentado na figura abaixo, como o Git reporta esse tipo de conflito.

```

1 <?php
2
3 class extrasController{
4
5     public function extrasControllerViewDepartament(){
6         require_once("../views/extras/extraView.php");
7         $dados["page_header"] = "Tipos de Departamentos";
8         $dados["page_panel"] = "Nesta página é possível criar e deletar departamentos.";
9         $table_information = ["id" => "ID", "description" => "Departamento"];
10        $dados["table"] = $table_information;
11        $userView = new extraView();
12        $userView->extraListViewDepartamentos($dados);
13        echo "Linha inserida pelo desenvolvedor 1";
14    }
15 }

```

Figura 12 – Desenvolvedor 1 adiciona uma linha na função extrasControllerViewDepartament no arquivo extrasController.php

```

1 <?php
2
3 class extrasController{
4
5     public function extrasControllerViewDepartament(){
6         require_once("../views/extras/extraView.php");
7         $dados["page_header"] = "Tipos de Departamentos";
8         $dados["page_panel"] = "Nesta página é possível criar e deletar departamentos.";
9         $table_information = ["id" => "ID", "description" => "Departamento"];
10        $dados["table"] = $table_information;
11        $userView = new extraView();
12        $userView->extraListViewDepartamentos($dados);
13        echo "Linha inserida pelo desenvolvedor 2";
14    }
15 }

```

Figura 13 – Desenvolvedor 2 adiciona uma linha na mesma função, no mesmo arquivo do desenvolvedor 1.

```

C:\wamp\www\s8-homepage-pop-rn>git merge testegit
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.

C:\wamp\www\s8-homepage-pop-rn>

```

Figura 14 – Tentamos realizar o comando git merge testegit e recebemos uma mensagem de erro.

Podemos obter mais detalhes sobre o erro de merge através do comando `git status`, conforme figura abaixo:

```
C:\wamp\www\s8-homepage-pop-rn>git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   adm/controllers/extras/extrasController.php

no changes added to commit (use "git add" and/or "git commit -a")
C:\wamp\www\s8-homepage-pop-rn>
```

Figura 15 – O `git status` retorna haver um problema com o arquivo `extrasController.php`.

O Git possui um outro comando para visualizarmos o merge entre dois branches distintos. Veja na figura abaixo que ele mostra o conflito em nível de função. Temos 2 linhas inseridas por usuários diferentes no mesmo arquivo, mesma função. Comando `git diff <branch_origem> <branch_destino>`.

```
C:\wamp\www\s8-homepage-pop-rn>git diff master testegit
diff --git a/adm/controllers/extras/extrasController.php b/adm/controllers/extras/extrasController.php
index 99d0dd5..088d88c 100644
--- a/adm/controllers/extras/extrasController.php
+++ b/adm/controllers/extras/extrasController.php
@@ -10,6 +10,6 @@ class extrasController{
    $dados["table"] = $table_information;
    $userView = new extraView();
    $userView->extraListViewDepartaments($dados);
-    echo "Linha inserida pelo desenvolvedor 1";
+    echo "Linha inserida pelo desenvolvedor 2";
  }
\ No newline at end of file
C:\wamp\www\s8-homepage-pop-rn>
```

Figura 16 – Com o `git diff` podemos ver quais linhas apresentam conflitos.

A partir da identificação do conflito, abrimos nosso editor de códigos, navegamos até o arquivo em conflito, e podemos ver quais linhas estão em conflito (figura abaixo). Observe que o HEAD aponta para as alterações conflitantes.

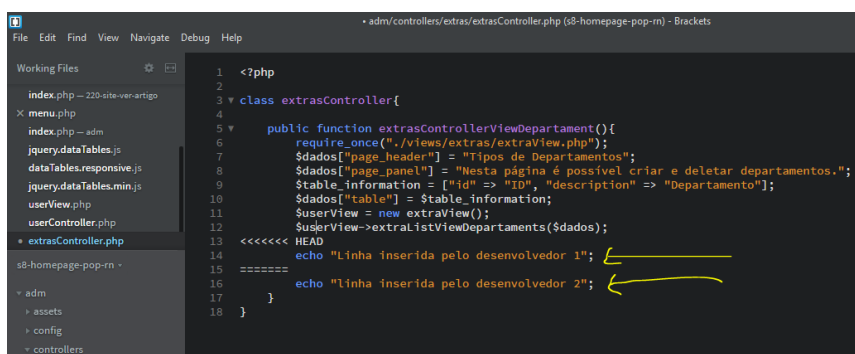
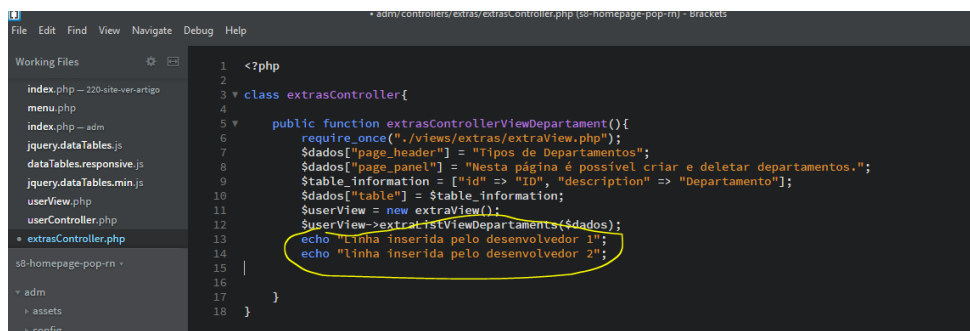


Figura 17 – Editor de código visualizando os conflitos detectados pelo Git.

Agora basta editarmos manualmente o conteúdo do arquivo, salvarmos e fixarmos o problema fazendo um novo **git add *** e **git commit -m "descrição"** conforme figura abaixo.



```

1 <?php
2
3 class extrasController{
4
5     public function extrasControllerViewDepartament(){
6         require_once("../views/extras/extraView.php");
7         $dados["page_header"] = "Tipos de Departamentos";
8         $dados["page_panel"] = "Nesta página é possível criar e deletar departamentos.";
9         $table_information = ["id" => "ID", "description" => "Departamento"];
10        $dados["table"] = $table_information;
11        $userView = new extraView();
12        $userView->extraViewDepartamentos($dados);
13        echo "linha inserida pelo desenvolvedor 1;";
14        echo "linha inserida pelo desenvolvedor 2;";
15    }
16
17 }
18

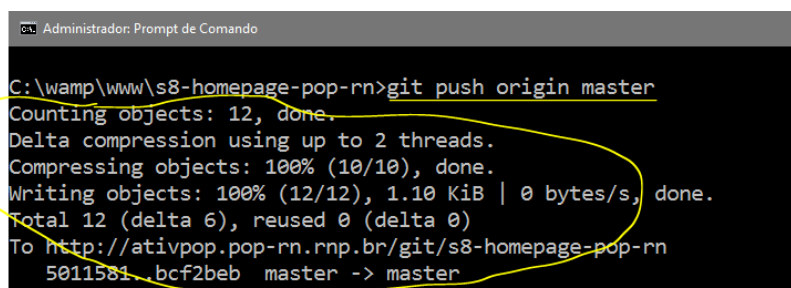
```

Figura 18 – Alteração do código conflitante manualmente.

Um ponto importantíssimo a frisarmos, é que no momento em que o git detecta um conflito no merge, ficamos impedidos de **trocar de branch até fixarmos o problema**, ou seja, realizamos o **git add *** e o **git commit -m "descrição"** dentro do próprio master. Por este motivo não precisamos mais fazer o merge. Apenas o push, nosso próximo passo.

Passo 8 – Realizar Push do Repositório Local com o Repositório Remoto

Nesse momento utilizaremos o comando **git push origin master** para concluirmos o processo de atualização de repositórios.



```

C:\wamp\www\s8-homepage-pop-rn>git push origin master
Counting objects: 12, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 1.10 KiB | 0 bytes/s, done.
Total 12 (delta 6), reused 0 (delta 0)
To http://ativpop.pop-rn.rnp.br/git/s8-homepage-pop-rn
5011581..bcf2beb master -> master

```

Figura 19 – Conclusão de atualizações utilizando o git push origin master.

Passo 9 – Deletar Branch de Tarefas após PUSH com Repositório Remoto

Para mantermos a organização de nosso projeto, recomendamos deletar os branches de desenvolvimento de tarefas/funcionalidades utilizados após o push com o repositório remoto. Fazendo assim, estabelecemos a metodologia de manter nosso branch **master** atualizado e sincronizado com o remoto, e evitamos possíveis confusões com listas de branches existentes no repositório local.

Uma outra razão para deletarmos os branches após suas modificações terem sido fundidas com o repositório remoto é que o branch master local sempre ficará à frente do branch de tarefas/funcionalidades utilizado previamente ao merge. Para deletar um branch utilizamos o comando **git branch -d "nomeDoBranchSemAspas"**. Vale salientar que só poderemos excluir um branch se o HEAD

não estiver apontando para este branch, bem como não commits e/ou conflitos pendentes relacionados a este branch. Veja figura abaixo.

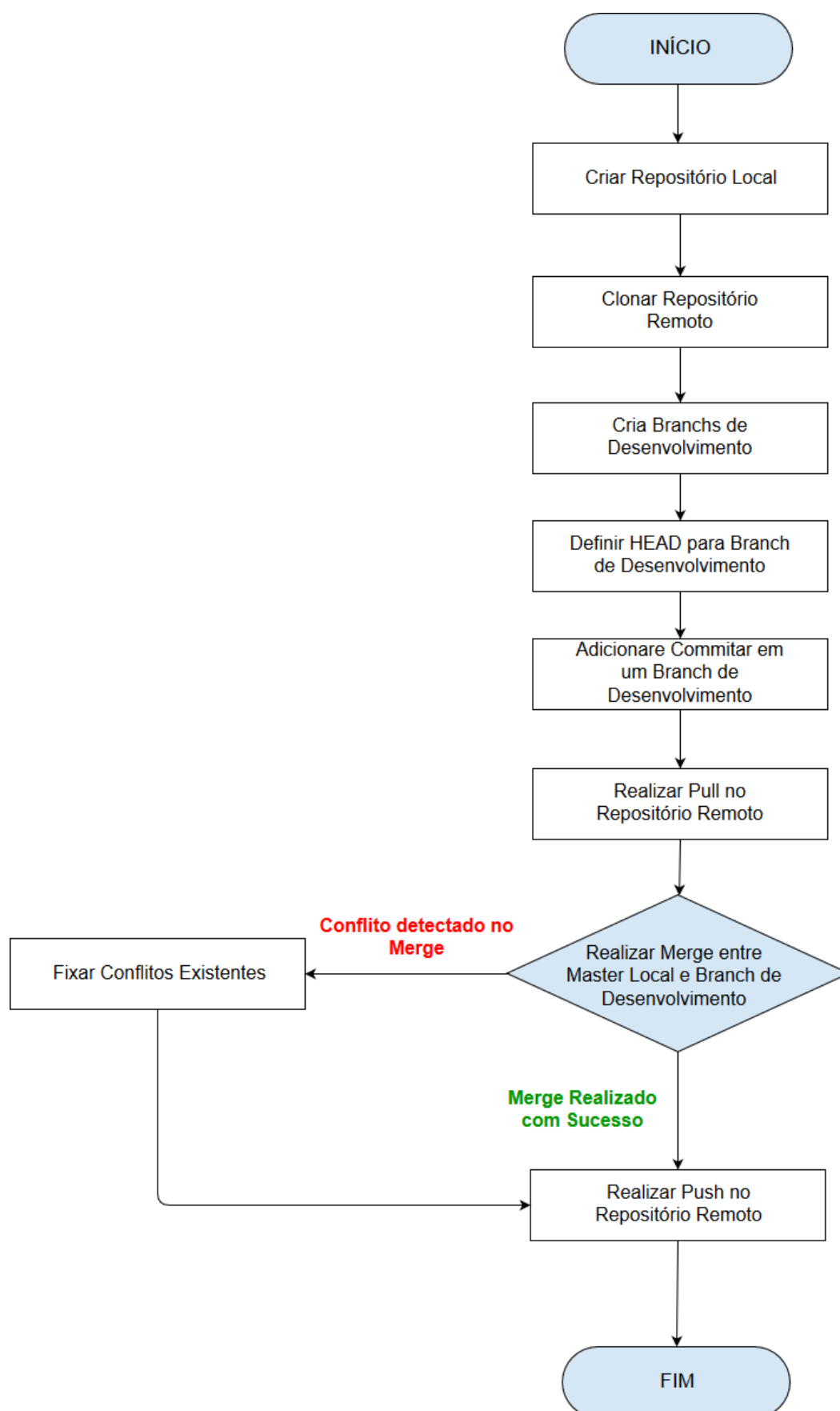
```
C:\wamp\www\s8-homepage-pop-rn>git branch -d tarefa1
error: Cannot delete branch 'tarefa1' checked out at 'C:/wamp/www/s8-homepage-pop-rn'

C:\wamp\www\s8-homepage-pop-rn>git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

C:\wamp\www\s8-homepage-pop-rn>git branch -d tarefa1
Deleted branch tarefa1 (was 15244c1).
```

Figura 20 – Deletando um branch com o git branch -d "nomeDoBranchSemAspas".

Fluxograma



Conclusão

Estabelecemos um modelo de trabalho utilizando a ferramenta Git para gerenciamento do desenvolvimento distribuído do Projeto do novo portal PoP-RN. Abordamos os conceitos e comandos mais utilizados em nossa implementação. A ferramenta Git é bem mais robusta e dispõe de dezenas de outros comandos importantes e interessantes que podem e devem ser pesquisados e conhecidos. Contudo, concentramos o foco nos comandos que estaremos utilizando com maior frequência no desenvolvimento de nossa aplicação.