

	Manual de Administração do GIT	Página 1/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

Modelo de Uso do GitLab

Versão 2.0

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
--	---	----------------------

	Manual de Administração do GIT	Página 2/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

Histórico de Revisões

Data	Versão	Descrição	Autor
03/08/2016	1.0	Criação do documento	CTMONSI
21/10/2016	1.1	Adicionados itens 1.2, 2 e 3	CTMONSI
09/11/2016	1.2	Adicionado itens 1.2.1 e 1.7	CTMONSI
07/12/2016	1.3	Ajustes Inclusão de informações no capítulo 2 Criação dos capítulos 6 e 8	CTMONSI
16/01/2017	1.4	Ajuste no capítulo “Criação de Grupo” Ajuste no capítulo “Concessão de acesso” Ajuste no capítulo “Concessão de acesso a Grupo” Ajuste no capítulo “Concessão de acesso ao Repositório” Ajuste no capítulo “Integração GIT – Jenkins”	CTMONSI
06/02/2017	1.5	Ajuste no capítulo “Concessão de acesso” Ajuste no capítulo “Criação de Repositório”	CTMONSI
21/02/2017	1.6	Alteração no capítulo “Configuração de JOBS no Jenkins para execução com GIT” Inclusão de informação no capítulo “Perfis de usuário e inclusão em grupos”	CTMARG
24/02/2017	1.7	Melhorias no capítulo “Concessão de acesso”	CTMARG
08/03/2017	1.8	Remoção do Capítulo “Integração GIT – Jenkins”, o qual foi criado manual específico. Melhorias no Capítulo “Criação de Repositório”	CTMARG
29/09/2017	1.9	Inclusão do grupo Cedersj060_verificacao_qualidade nos procedimentos.	CTMARG
26/10/2017	2.0	Alteração para Corporativo	GEARQ CEDES BR RJ e SP SUCTI (PEDES)

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
---	--	---------------

	Manual de Administração do GIT	Página 3/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

Sumário

1	GLOSSÁRIO	4
2	Concessão de acesso	4
2.1	Solicitação	4
2.2	Política	4
2.3	Perfis de usuário e inclusão em grupos	5
2.4	Criação do usuário	5
2.5	Inclusão da chave pública RSA do usuário	5
3	Papéis e responsabilidades nos Grupos de Projetos	9
3.1	<i>Owner</i>	9
3.2	<i>Master</i>	9
3.3	<i>Developer</i>	9
3.4	<i>Reporter</i>	10
3.5	<i>Guest</i>	10
4	Criação de Projetos	10
4.1	Passos para criação do projeto	10
4.2	Concessão de acesso aos Projetos	10
5	Políticas e Procedimentos	10
5.1	Política de Uso de Repositórios Remotos	10
5.2	Política de uso do <i>Branch</i> (Ramos de Desenvolvimento)	11
5.3	Política de uso de <i>Baseline</i> (versão de <i>software</i>)	12
5.4	Política de Expurgo dos “Códigos Transitórios”	13
5.5	Procedimento de Retorno de versão	14
5.6	Procedimento para impedir criação de <i>login</i> através da interface	16
6	Configurações Globais do Repositório	16
6.1	Uso do <i>gitignore</i>	17
6.2	<i>Hooks</i> Padrões	17

	Manual de Administração do GIT	Página 4/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

1 GLOSSÁRIO

- GIT – Ferramenta de controle de configuração distribuída e um sistema de gerenciamento de código fonte com licenciamento de software livre (GNU *General Public License*).
- GITLAB – Gerenciador de repositórios GIT
- GCM – Gerência de Configuração e Mudança
- GC – Gerência de Configuração
- *Branch* (ramo de desenvolvimento) – são utilizados para permitir o desenvolvimento paralelo e a segregação do trabalho. Principal recurso no GIT para separar o trabalho de construção do que efetivamente já está concluído, bem como segregar a correção de algo em produção das melhorias que estão sendo desenvolvidas.
- Configuração: Conjunto de características físicas e funcionais de um produto conforme definidas na documentação técnica e obtidas no produto [ISO10007].
- Controle de configuração: Atividades compreendendo o controle de alterações em itens de configuração após o estabelecimento formal de seus documentos de configuração. [ISO10007].
- *Merge*: Ato de juntar/conciliar dois ramos diferentes de desenvolvimento (*branch*).
- Item de configuração: objeto sob gerência de configuração. Partes únicas que compõe a configuração.
- *Baseline de Software*: Identificação dos elementos/versões que compõe uma versão de *software*.
- Artefato/Objeto: Um artefato é um pedaço de informação que é produzido, modificado ou utilizado no processo de desenvolvimento. Um artefato/objeto pode ser um documento, código fonte, modelo e etc.
- TAG: recurso do GIT para aplicação de *baselines* e/ou marcação de eventos relevantes do projeto/sistema.

2 Concessão de acesso

2.1 Solicitação

O usuário formalizará sua solicitação de concessão de acesso para a Coordenação de Suporte (BR – 061, RJ – 042 e SP - 051, SUCTI05). Na solicitação deve ser informado a Carteira de Negócios ou Fábrica na qual irá trabalhar e as referidas matrículas.

2.2 Política

A seguinte política deve ser cumprida para segurança dos Projetos existentes no GIT:

- I. O grupo "Qualidade_GIT_XX" conterà somente os empregados das Coordenações de Suporte e os prestadores de suporte ao GIT. Os membros deste grupo serão cadastrados como *owner*.
- II. O Grupo "SUPORTE_GIT_XX" conterà somente os empregados que suportam a ferramenta administrativamente. Os membros deste grupo serão cadastrados como *Admin* (por exemplo para as funcionalidades: *Force push to protected branches*, *Remove protected branches*).
 - o <https://docs.gitlab.com/ee/README.html#administrator-documentation>
- III. Os Grupos/Projetos serão criados como PRIVADO. E o projeto criado será

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
---	--	---------------

	Manual de Administração do GIT	Página 5/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

compartilhado com o grupo da qualidade responsável.

2.3 Perfis de usuário e inclusão em grupos

Os membros dos projetos (equipes de desenvolvimento/fábricas) serão cadastrados pelo Coordenador do Projeto por meio do SIGAL (Acesso Lógico).

Para cada sistema, existirão os seguintes perfis no Acesso Lógico para o GIT:

Admin Não Produção: permissão de *master* (grupo)

User Não Produção: permissão de *developer* (grupo)

A automação com o SIGAL ainda não está concluída, e o processo de concessão de acesso deverá ser realizado manualmente por cada CEDES.

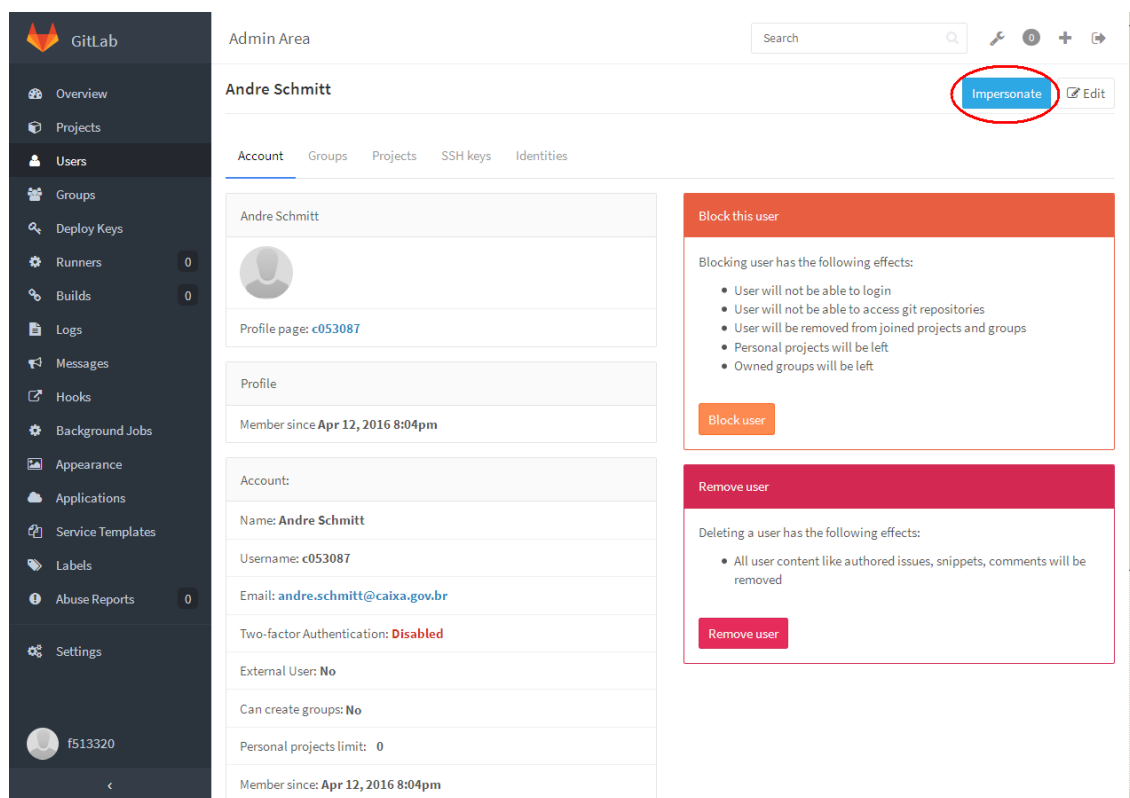
2.4 Criação do usuário

Logar no GITLAB (<http://fontes.des.caixa/>).

2.5 Inclusão da chave pública RSA do usuário

O usuário pode incluir a chave pública diretamente ou, em caso de dúvidas, solicitar para a equipe da Qualidade que irá proceder a inclusão via “*impersonate*”.

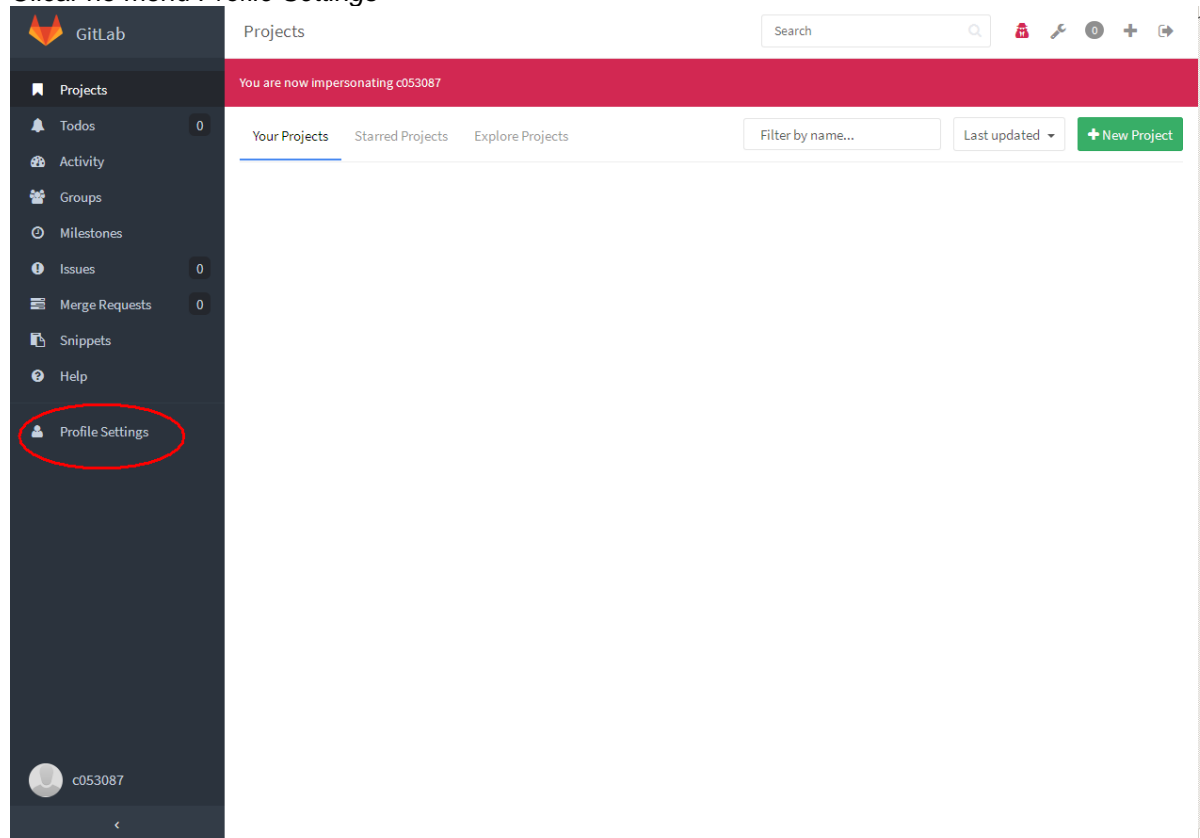
Procedimento via *impersonate*:



The screenshot shows the GitLab Admin Area for user Andre Schmitt. The left sidebar contains navigation links: Overview, Projects, Users, Groups, Deploy Keys, Runners, Builds, Logs, Messages, Hooks, Background Jobs, Appearance, Applications, Service Templates, Labels, Abuse Reports, and Settings. The main content area shows the user's profile with tabs for Account, Groups, Projects, SSH keys, and Identities. The 'Account' tab is active, displaying user details like Name, Username, Email, Two-factor Authentication (Disabled), External User (No), Can create groups (No), Personal projects limit (0), and Member since (Apr 12, 2016 8:04pm). On the right, there are two action panels: 'Block this user' with a list of effects and a 'Block user' button, and 'Remove user' with a list of effects and a 'Remove user' button. The 'Impersonate' button is circled in red in the top right corner of the user profile section.

	Manual de Administração do GIT	Página 6/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

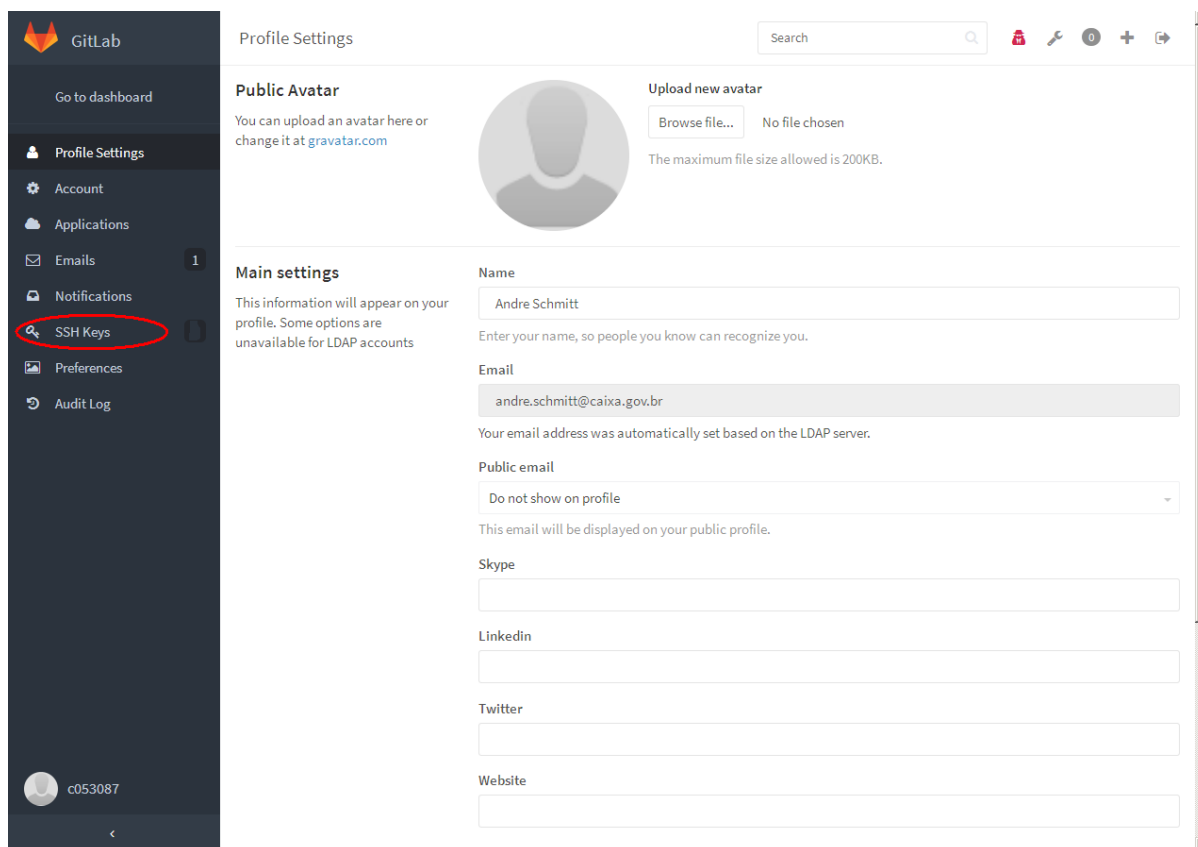
Clicar no menu *Profile Settings*



Clicar no menu *SSH Keys*

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
--	---	----------------------

	Manual de Administração do GIT	Página 7/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		



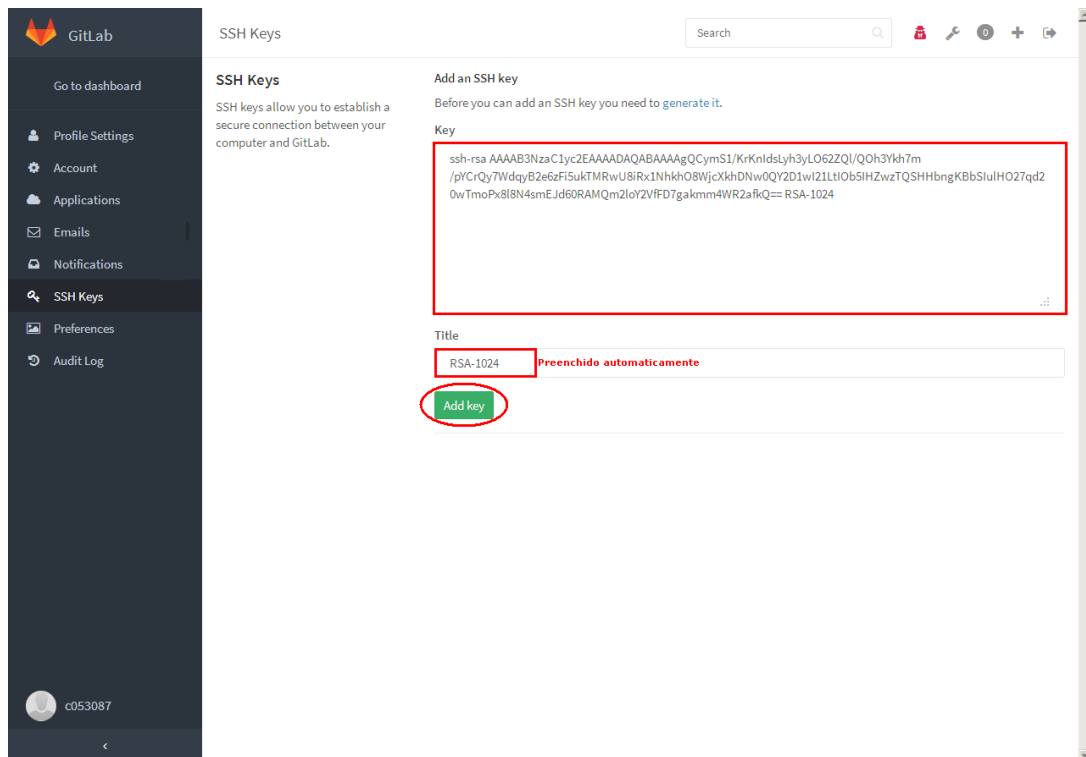
Preencher o campo *key* com o conteúdo do arquivo contendo a chave pública RSA do usuário.

O campo *Title* será preenchido automaticamente.

Clicar no botão Add key.

<p>Elaborado por CEDES BR RJ SP GEARQ SUCTI</p>	<p>Arquivo Modelo de Uso do GitLab.doc</p>	<p>Versão 2.0</p>
--	---	------------------------------

	Manual de Administração do GIT	Página 8/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

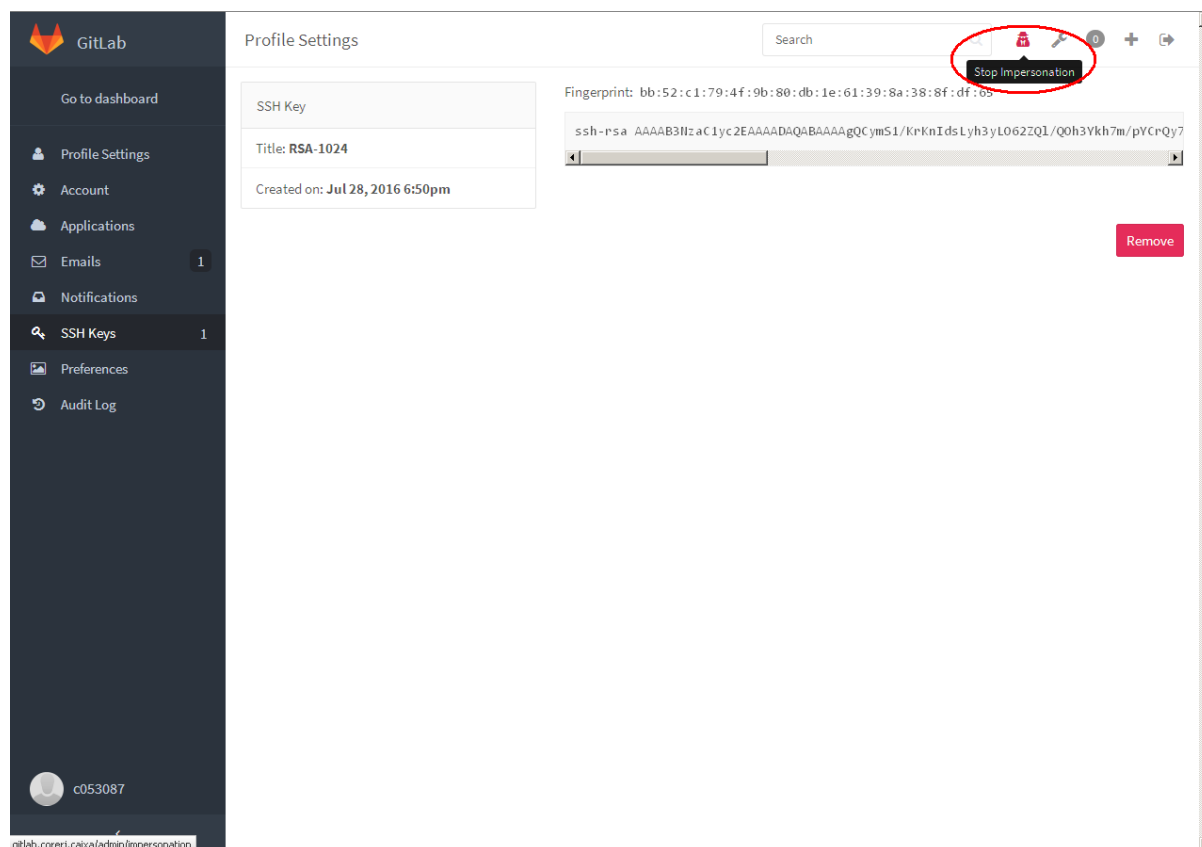


Chave incluída.

Clicar em *Stop Impersonation*

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
---	--	---------------

	Manual de Administração do GIT	Página 9/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		



3 Papéis e responsabilidades nos Grupos de Projetos

O usuário poderá ser classificado no grupo segundo os seguintes perfis no GitLab: *Owner*, *Master*, *Developer*, *Reporter* ou *Guest*.

3.1 Owner

- Papel desempenhado pelas equipes de suporte.
- Utilizado somente para os administradores da ferramenta.
- Deverá criar e administrar Projetos, Grupos e seus usuários e realizar demais atividades administrativas.
- O usuário *Owner* é capaz de desempenhar todos os papéis definidos no GitLab e por isso seu uso deve ser restrito às equipes descritas acima.

3.2 Master

- Permite realizar *merges* para o *branch master* (ou outros *branches* protegidos).
- Permite criar *tags* no *branch master* para marcar a *baseline* do Sistema.
- Permite remover os *branches* remotos que originam os *merges* após realização dos mesmos.

3.3 Developer

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
---	--	---------------

	Manual de Administração do GIT	Página 10/17 Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

- Poderá clonar qualquer *branch*.
- Não poderá realizar *merge* sobre o *branch master* (ou outros *branches* protegidos).
- Colaboradores com perfil desenvolvedor deverão realizar *merge requests* para os *branches* protegidos.

3.4 Reporter

- A princípio não será desempenhado por nenhum usuário da Caixa.

3.5 Guest

- Desempenhado por empregados e colaboradores (Auditoria, por exemplo).
- O acesso é limitado a verificação dos arquivos dos projetos através da interface do GitLab.

4 Criação de Projetos

4.1 Passos para criação do projeto

Logar no GITLAB (<http://fontes.des.caixa/>),

Escreva o nome do novo Projeto, em caixa baixa.

Escreva no campo *Description* a descrição do Projeto conforme a descrição existente para o sistema no siapp.caixa.

Marcar *Visibility Level* como *Private*.

Clicar no Botão *Create project*.

4.2 Concessão de acesso aos Projetos

ATENÇÃO: O acesso ao projeto é realizado somente através do compartilhamento de grupos.

Obs.: Em casos excepcionais um usuário Caixa poderá receber perfil *Master*. Para tanto, sua matrícula deverá ser incluída diretamente no Projeto em questão e o perfil *Master* atribuído.

Um usuário de serviço de determinada ferramenta ou aplicação poderá ser criado no GIT para atender ao acesso da mesma. Este usuário deverá ser incluído com perfil *Master* diretamente no projeto do qual faz uso.

5 Políticas e Procedimentos

5.1 Política de Uso de Repositórios Remotos

Para ser capaz de colaborar com qualquer projeto no Git é necessário que o usuário saiba

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
---	--	---------------

	Manual de Administração do GIT	Página 11/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

gerenciar seus repositórios remotos. Repositórios remotos são ambientes de configuração dos seus códigos que estão hospedados além do seu ambiente local, como o GitLab.

Deve-se ressaltar que o GitLab é o repositório oficial para colaboração, onde os códigos deverão estar devidamente armazenados e gerenciados.

Para se obter os códigos que estão no repositório remoto você pode executar o comando `$ git fetch [nome-remoto]`

Esse comando vai até o projeto remoto e captura todos os códigos e demais arquivos que você ainda não tem na sua versão local. Depois de fazer isso, você deve ter referências para todos os *branches* desse remoto, onde você pode fazer o *merge* ou inspecionar a qualquer momento (ver Política de uso do *Branch*).

Se você clonar um repositório, o comando automaticamente adiciona o remoto com o nome *origin*. Então, `git fetch origin` busca qualquer novo trabalho que foi enviado para esse ambiente remoto desde que você o clonou (ou fez a última busca).

É importante notar que o comando *fetch* traz os dados para o seu repositório local — ele não faz o *merge* automaticamente com os seus dados ou modifica o que você está trabalhando atualmente. Você terá que fazer o *merge* manualmente no seu trabalho quando estiver pronto.

Se você tem um *branch* configurado para acompanhar um *branch* remoto, você pode usar o comando `git pull` para automaticamente fazer o *fetch* e o *merge* de um *branch* remoto no seu *branch* atual. Essa pode ser uma maneira mais fácil ou confortável para baixar os seus códigos; e por padrão, o comando `git clone` automaticamente configura seu *branch* local *master* para acompanhar o *branch* remoto *master* do servidor de onde você clonou (desde que o remoto tenha um *branch master*).

Executar `git pull` geralmente busca os dados do servidor de onde você fez o clone originalmente e automaticamente tenta fazer o *merge* dele no código que você está trabalhando atualmente.

Para maiores informações utilize como fonte os documentos oficiais e atualizados do Git e do GitLab.

5.2 Política de uso do *Branch* (Ramos de Desenvolvimento)

Para efeitos didáticos, o processo de codificação do Git produz pelo menos 3 tipos diferentes de versões de código-fonte (Git local):

- Códigos transitórios que ainda não foram concluídos pelo programador e por consequência representam o que ainda está sendo desenvolvido (trabalho temporário);
- Códigos que geraram *software* estável e apto a entrar em produção (trabalho considerado concluído, mas ainda transitório), mas que ainda não foi, ou está sendo, testado e aprovado;
- Códigos que geraram versões de *software* que efetivamente entraram em produção (trabalho efetivamente concluído).

Sob a perspectiva acima, no repositório remoto poderão ser geradas diversas versões de código que poderiam ser descartadas ao longo da finalização de determinados marcos do projeto (ver Política de Expurgo para Códigos Transitórios). Para que possa ocorrer a segregação clara do que é definitivo e do que é transitório, a política de *branch* deve levar em consideração:

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
--	---	----------------------

	Manual de Administração do GIT	Página 12/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

- A *branch MASTER* deve ser utilizada somente para manter versões de trabalho efetivamente concluído (versões de *software* que entraram em produção);
- Qualquer código em desenvolvimento deve ser tratado em *branches* de objetivo específico com temporalidade definida.
- A estrutura de *branch* do projeto deve considerar a forma e a divisão do trabalho na equipe, bem como a segregação em versões de *software*.

Com base nos princípios acima, sugerimos o seguinte modelo de *branch*:

- **Branch MASTER:** somente versões de produção. Após implantação de uma versão em produção, é realizado *merge* com a *branch MASTER*. Permissão: somente integrador.
- **Branch por demanda (RTC):** uma *branch* para cada demanda (DMS/Defeito), criada preferencialmente a partir da última versão em produção do *software*. Alterações no código somente serão realizadas nestas *branches*. Correções também serão construídas neste tipo de *branch*. Permissão: desenvolvedor.
- **Branch de integração (por versão):** uma *branch* para cada versão de planejamento. É utilizada para segregar aquilo que foi transformado em uma versão (empacotamento) e para preparar o que será testado. Nenhuma codificação deve ser realizada nesta *branch*. Permissão: somente integrador.
- **Observações:**
 - O termo “versão de planejamento” se refere à identificação atribuída para as demandas que fazem parte de uma entrega. Considerando a máscara de *baseline* (ver Política de uso de *Baseline*), a versão de planejamento é identificada com os 3 primeiros números da *baseline* (VMa.Vme.FIX).
 - As entregas da equipe de desenvolvimento (interna ou fábrica) serão realizadas nas *branches* por demanda;
 - O integrador (interno ou fábrica) obtém as alterações nas *branches* de demandas e realiza o *merge* de uma ou várias com a *branch* de integração (por versão). Caso seja necessário alterar algum código durante a integração, a alteração será realizada na *branch* da demanda e novamente feito o *merge* na de integração. Este procedimento permitirá que o conteúdo de uma versão possa ser remontado a qualquer instante com as mesmas demandas ou outras. Cada *branch* de integração representa uma data estimada de implantação de *software*.
 - Considerando o conceito de versão candidata, a aplicação de *baseline* de *software* será realizada somente na *branch* de integração.
 - Após implantação de uma versão em produção, o integrador realiza o *merge* da *branch* de integração (por versão) com a *master*.

5.3 Política de uso de *Baseline* (versão de *software*)

A *baseline* tem o objetivo de identificar uma versão estável de um *software*. O termo estável deve ser enxergado sob a perspectiva de quem desenvolve. Representa o momento que o desenvolvedor considera o *software* pronto para o processo de testes.

Considerando o tipo de desenvolvimento, normalmente este momento ocorre:

- Desenvolvimento terceirizado: quando a fábrica entrega uma versão de *software* para a CAIXA.
- Desenvolvimento interno: quando o integrador conclui o processo de *merge* do trabalho individual dos desenvolvedores e realiza a integração para disponibilizar para a equipe de teste.

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
---	--	---------------

	Manual de Administração do GIT	Página 13/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

Em qualquer um dos casos, antes do *software* ser testado, uma *baseline* precisa ser aplicada. Este procedimento garante que o teste seja realizado em uma versão específica do *software*, e que durante e após a execução dos mesmos os componentes da versão não sofrerão alterações.

A compilação e aplicação do TAG (*baseline* de *software*) só são realizadas em ambiente DES. Para os demais ambientes, deve-se utilizar a mesma compilação realizada em DES (promoção de *baseline*).

No GIT, as *baselines* são aplicadas pelo recurso TAG.

No intuito de manter padrão e conformidade com as versões de *software* já existentes nas demais ferramentas de configuração da CAIXA, a máscara da TAG deve seguir o mesmo padrão do modelo de uso do clearcase:

VMa.Vme.FIX.Build

Onde:

- **VMa:** Versão principal do sistema, representa o escopo maior. Deve ser incrementado quando houver grandes mudanças de escopo ou mudanças significativas na arquitetura do sistema. Não pode ser zerado.
- **Vme:** Versão secundária do sistema que representa o escopo menor. Deve ser incrementado quando houver pequenas e médias mudanças no escopo do sistema, como novas funcionalidades, mas que não alteram significativamente a arquitetura e/ou o próprio sistema. Deve ser zerado quando houver mudança na versão maior.
- **FIX:** Representa correções de *software* em produção. Deve ser incrementado para identificar que uma versão está sendo construída para corrigir um erro que ocorre em produção. É incrementado somente uma vez a cada *fix*, mesmo que o *software* seja compilado várias vezes para fazer a correção. Este número somente será incrementado novamente caso o *software* em produção apresente novos erros, após o *fix* anterior ter sido aplicado em produção. Deve ser zerado quando ocorrerem alterações na VMa/Vme.
- **Build:** Representa a quantidade de compilações de um determinado escopo de *software* (VMa.Vme). Deve ser incrementado quando o *software* é considerado estável por quem desenvolve, após integração, antes do teste. Caso ocorram erros durante o processo de testes e homologação (fase de desenvolvimento), o *software* deverá ser corrigido, nova integração e compilação realizadas e este número será novamente incrementado. Correções do *software* durante o processo de desenvolvimento, geram incremento somente no número da *build*. Deve ser zerado quando ocorrerem alterações na VMa/Vme/FIX.

5.4 Política de Expurgo dos “Códigos Transitórios”

O uso indiscriminado de replicação de conteúdo do repositório ou para o repositório remoto pode obrigar aos seus usuários uma política clara de expurgo do conteúdo não essencial em longo prazo. Já que o descuido em relação a este assunto poderia facilmente degradar sobremaneira procedimentos de replicação (*git clone*) e até mesmo inviabilizar o uso do repositório em estações de trabalho com HDs pequenos.

Devemos tratar como códigos transitórios, todas as versões que foram geradas, mas que não entraram em produção. São códigos que por algum motivo foram considerados inadequados ou inacabados. E na prática possuem pouco ou nenhum valor após implantação da versão em produção.

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
--	---	----------------------

	Manual de Administração do GIT	Página 14/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

No intuito de manter os repositórios limpos e somente com códigos relevantes, após implantação em produção de uma versão de software, sugerimos apagar:

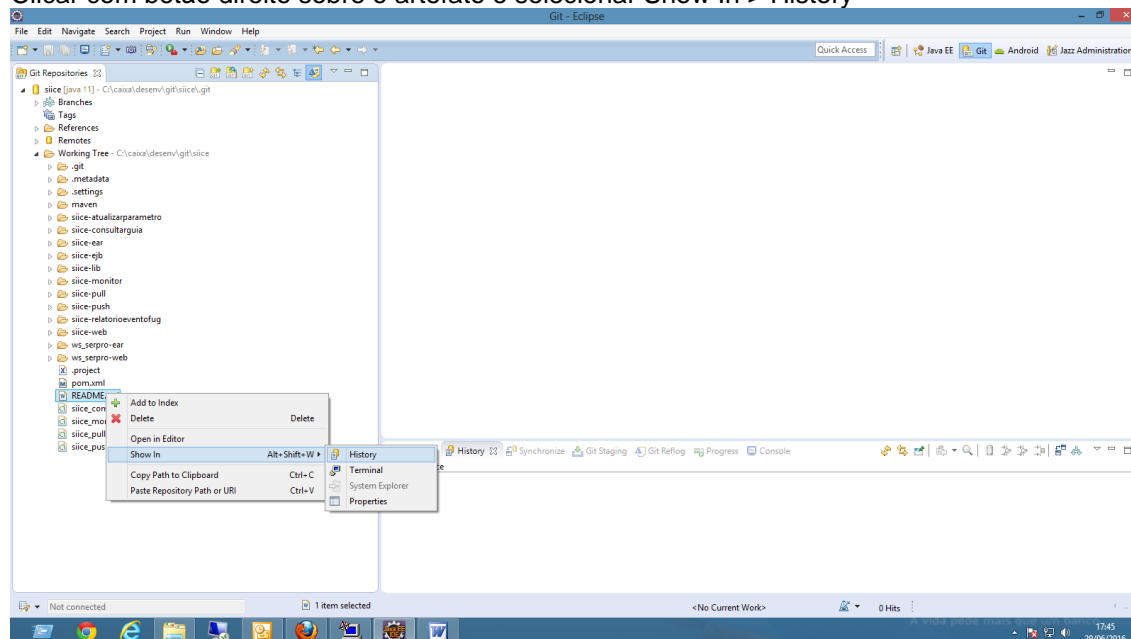
- *Branches* de demandas (melhorias/defeitos) que compõe a versão de *software* implantada;
- *Branch* de integração (por versão) utilizada para compilar e testar a versão de *software* implantada;

Todas as *baselines* que possuem a mesma versão de planejamento (VMa.Vme.FIX) da versão de *software* implantada em produção, mas que não foram para a produção por terem sido consideradas inadequadas ou incompletas.

5.5 Procedimento de Retorno de versão

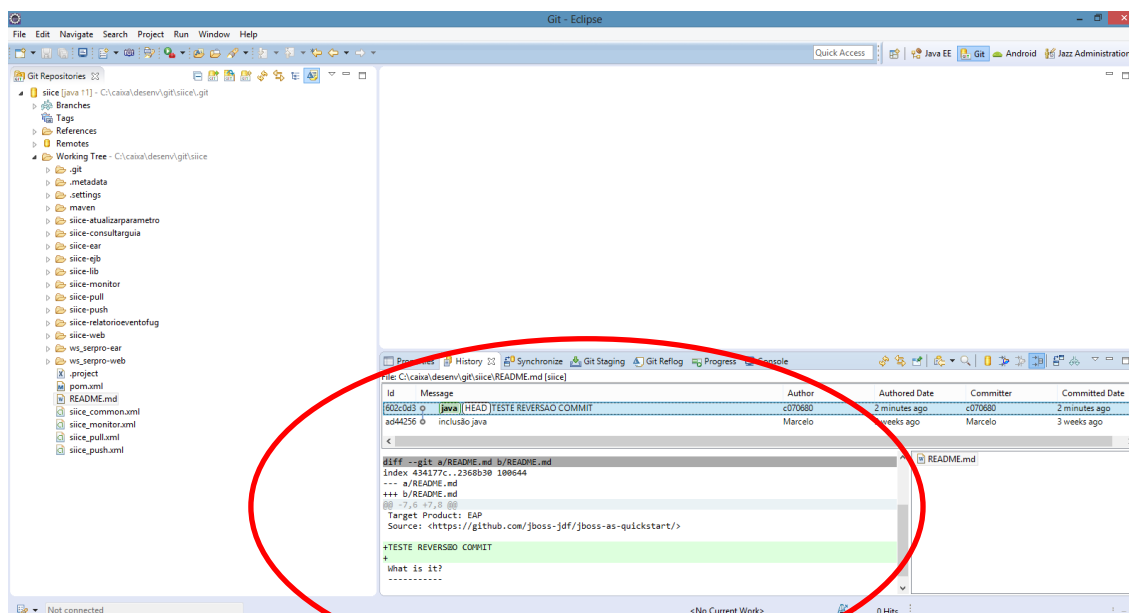
Procedimento para retornar à versão anterior de um artefato:

Clicar com botão direito sobre o artefato e selecionar Show In > History

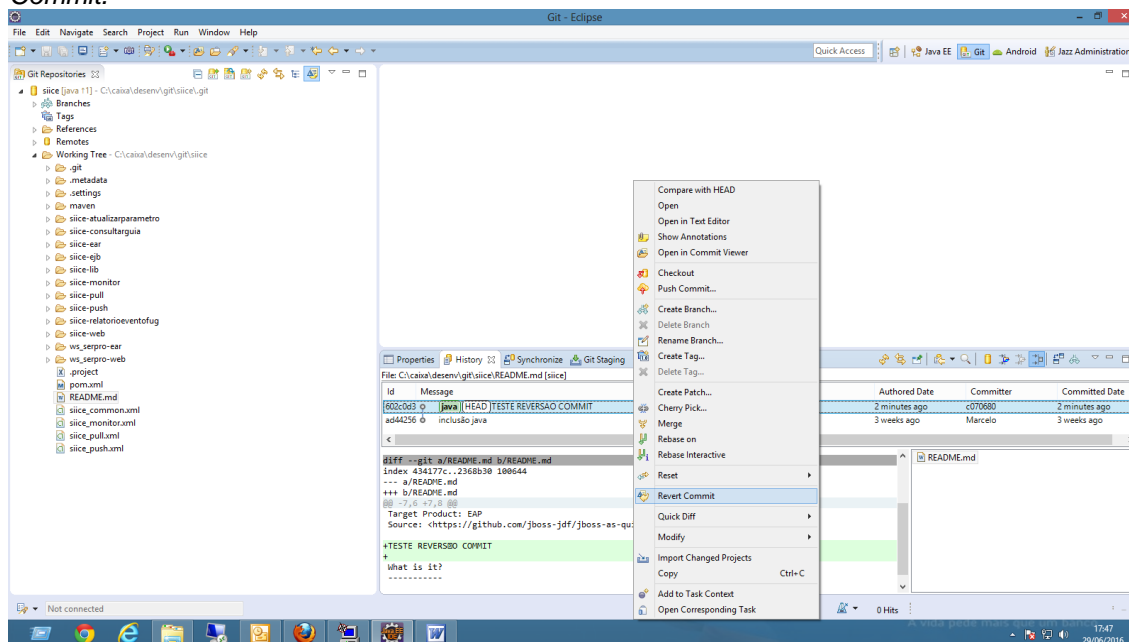


Será exibido o histórico do artefato

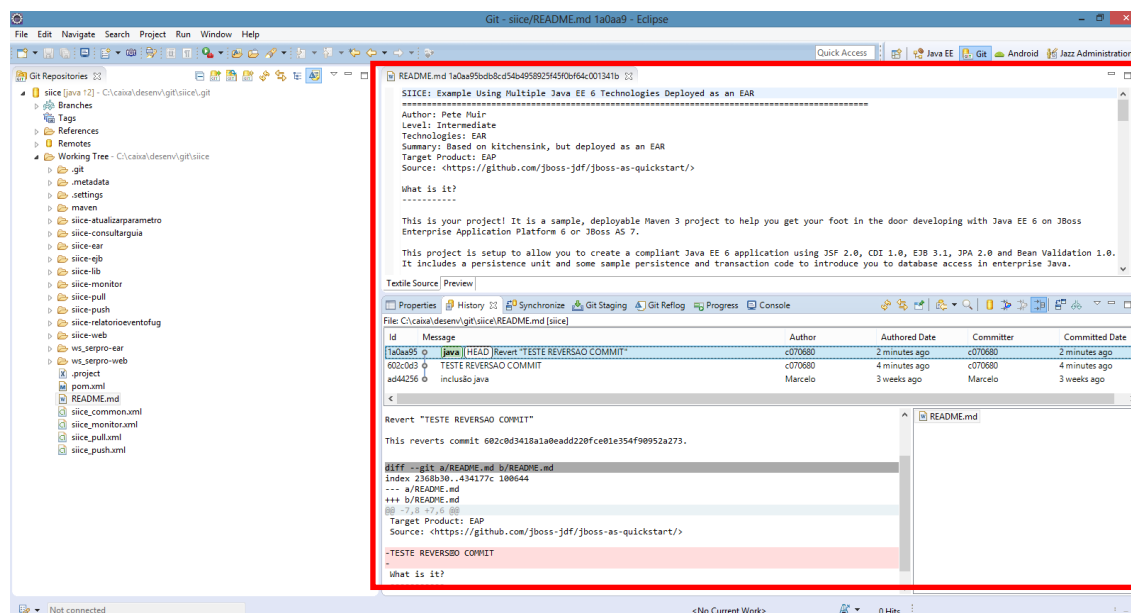
<p>Elaborado por CEDES BR RJ SP GEARQ SUCTI</p>	<p>Arquivo Modelo de Uso do GitLab.doc</p>	<p>Versão 2.0</p>
--	---	------------------------------



Clicar com botão direito na árvore sobre a versão que se deseja retornar e selecionar *Revert Commit*.

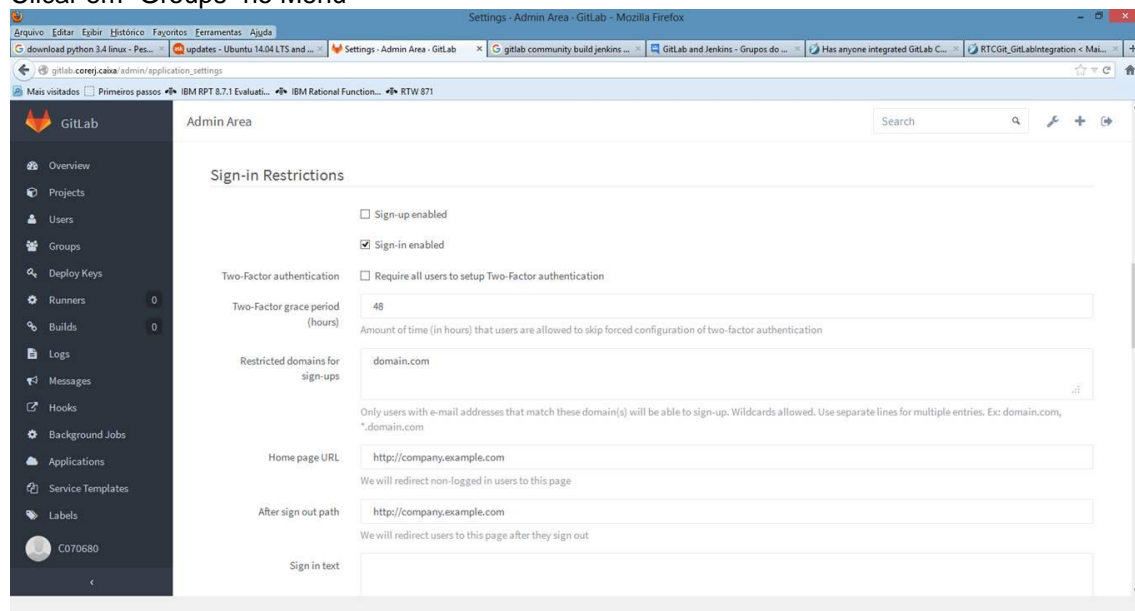


Resultado do retorno da versão. Será incluído o comentário “*Revert*” ao lado do artefato revertido.



5.6 Procedimento para impedir criação de login através da interface

Clicar em “Groups” no Menu



6 Configurações Globais do Repositório

Configurações globais se aplicam a todos os usuários e projetos

	Manual de Administração do GIT	Página 17/17
		Data Emissão 30/10/2017
GCM – Gerência de Configuração e Mudanças		

6.1 Uso do *gitignore*

O recurso *.gitignore* deve ser utilizado para restringir o conteúdo que de fato deve ser armazenado e versionado no GIT.

A configuração é realizada em um arquivo que contém uma lista de pastas e/ou padrões de arquivo (em especial extensões) que devem ser ignoradas no *commit*.

O *.gitignore* deve ser restringir no mínimo:

- Arquivos de configuração de ferramentas e interfaces de desenvolvimento (verificar com SAI). Exceto os necessários para a abertura do projeto em qualquer estação de trabalho;
- Arquivos compilados;
- Documentos diversos (doc; xls; ppt...);
- PSTs;
- Arquivos compactados (zip, rar...).

Quaisquer outras necessidades de configurações globais devem ser discutidas previamente com a GEARQ e as equipes de qualidade.

6.2 Hooks Padrões

Hooks permitem obrigar políticas e regras de Gerência de Configuração em repositórios GIT.

Inicialmente os repositórios serão criados sem uso de *Hooks*, e quaisquer necessidades devem ser discutidas previamente com a GEARQ e as equipes de qualidade.

Elaborado por CEDES BR RJ SP GEARQ SUCTI	Arquivo Modelo de Uso do GitLab.doc	Versão 2.0
---	--	---------------