GIT

Apresentado por: André Justi

27 DE MAIO DE 2015

Git é um sistema de controle de versão distribuído com ênfase em velocidade para ser utilizado pela internet. O Git foi inicialmente projetado e desenvolvido por **Linus Torvalds** para o desenvolvimento do kernel **Linux**.

Uma breve história do Git

O kernel (núcleo) do Linux é um projeto de software de código aberto de escopo razoavelmente grande. Durante a maior parte do período de manutenção do kernel do Linux (1991-2002), as mudanças no software eram repassadas como patches e arquivos compactados. Em 2002, o projeto do kernel do Linux começou a usar um sistema DVCS proprietário chamado BitKeeper.

...Uma breve história do Git

Em 2005, o relacionamento entre a comunidade que desenvolvia o kernel e a empresa que desenvolvia comercialmente o BitKeeper se desfez, e o status de isento-de-pagamento da ferramenta foi revogado. Isso levou a comunidade de desenvolvedores do Linux (em particular Linus Torvalds, o criador do Linux) a desenvolver sua própria ferramenta baseada nas lições que eles aprenderam ao usar o BitKeeper.

Alguns dos objetivos do novo sistema eram:

- Velocidade
- Design simples
- Suporte robusto a desenvolvimento não linear (inúmeras de branches paralelas)
- Totalmente distribuído
- Capaz de lidar eficientemente com grandes projetos como o kernel do Linux (velocidade e volume de dados)

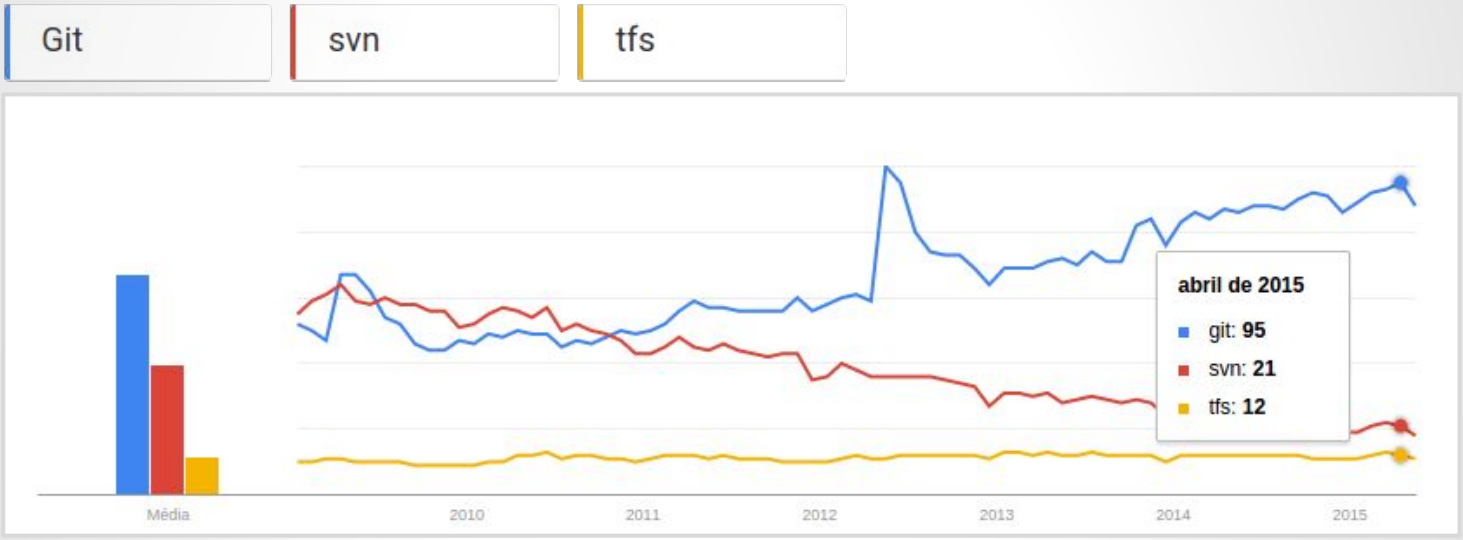
...Uma breve história do Git

Desde sua concepção em 2005, o Git evoluiu e amadureceu a ponto de ser um sistema fácil de usar e ainda mantém as qualidades iniciais. É incrivelmente rápido, bastante eficiente com grandes projetos e possui um sistema impressionante de branching para desenvolvimento não-linear.

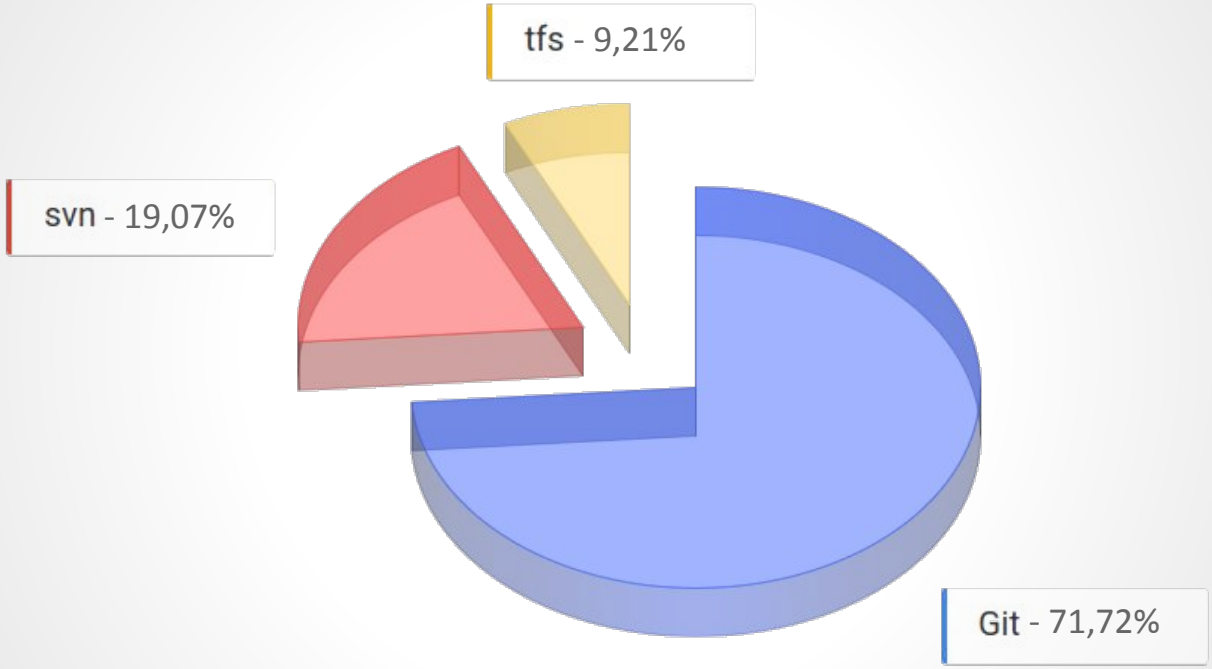
Quem utiliza

...Entre outras milhares de empresas e projetos

Oque estão falando (Busca Google)



Oque estão falando (Tags Stack Overflow)



Instalação

- **https://git-scm.com/download/**

Arquitetura de armazenamento

- Indíces
  - Armazenam informação sobre a versão atual e as mudanças feitas nela
- Banco de Dados de Objetos
  - Blobs (arquivos)
    - Armazenados na pasta .git/objects
    - Indexados por um único hash
    - Todos os arquivos são armazenados em blobs
- Trees (diretórios)
- Commits
  - Cada commit gera um novo objeto
  - Informações do commit: hash do pai, nome do autor, data/hora do commit e o hash da estrutura corrente

Arquivos de controle

- A pasta .git
  - Apenas no diretório raiz do projeto
  - Contem todos os objetos, commits e configurações do projeto
  - .git/config: arquivo com configurações específicas do repositório
- .gitignore
  - Arquivo texto que indica os arquivos que devem ser ignorados
    - Exemplo: *.exe, *.dll, *.o, ~*

Vantagens

- Open Source
- Comunidade muito forte
- Documentação simples
- Funciona offline
- A maioria das ferramentas (build, gestão e etc) já possuem integração nativa
- Foi projetado para ser totalmente distribuído desde o início, permitindo que cada desenvolvedor tenha controle local completo
- Muito mais rápidos que outros controles de versão (SVN e TFS)
- Repositórios são muito menores que outros controles de versão (SVN e TFS)
- Permite uma melhor e completa auditoria do que aconteceu no repositório

Vantagens

- Ao efetuar um merge o histórico dos commits da branch mergeada é levado para o nodo qual o merge foi feito
- O formato dos arquivo de controle do GIT são mais simples, por isso é mais fácil sua recuperação e corromper um arquivo é muito raro
- Algoritmo de merge e comparação extremamente avançado, garante maior assertividade do que foi alterado
- Algoritmos de compressão eficientes que analisam “o todo” reduz o tamanho local, assim como as transferências em operações de push/pull

Desvantagens

- Curva de aprendizado maior que os outros controles de versão (SVN e TFS)
- Aplicações visuais não possuem todos os recursos que o GIT prove

Glossário

Branch

Um ramo paralelo de um repositório, branchs podem ser locais e/ou remotas.

Clone

Um clone é uma cópia de um repositório no computador cliente.

Glossário

Commit

Um commit é uma mudança individual e um ou mais arquivos. É como quando você salvar um arquivo, no Git toda vez que você comitar um ou mais arquivos ele cria uma identificação única (também conhecida como o "SHA") que lhe permite manter o registro de quais alterações foram feitas naquele commit. Em simples palavras no GIT um commit é a persistência de um ou mais arquivos no repositório local.

Diff

O diff é a diferença entre dois commits. O diff vai visualmente descrever o que foi adicionado ou removido entre dois commits.

Fetch

Recupera informações mais recentes de um repositório remoto.

Glossário

Pull
Pull é a recuperação das novas mudanças do repositório remoto para o repositório local.

Pull Request
É uma solicitação de alteração em um repositório (normalmente terceiro).

Push
É o envio dos commits local para o repositório remoto.

Remote
É a versão de algo que está hospedado em um servidor (GitHub, Bitbucket e etc).

Glossário

Repository

Um repositório é a pasta raiz de um projeto com todos seus arquivos (e arquivo de controle do GIT). Repositórios podem ter vários usuário e podem ser públicos ou privados.

SSH Key

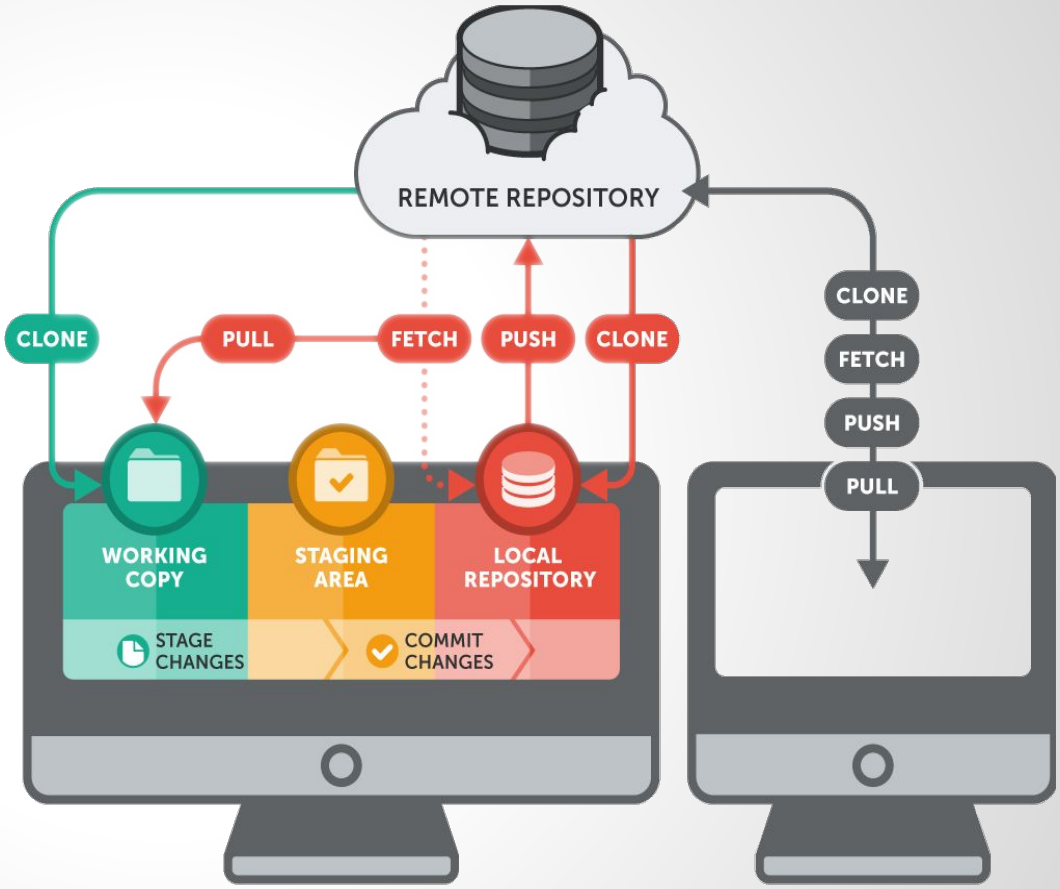
Chaves SSH são uma maneira de identificar-se no acesso a um repositório remoto, é algo como um certificado digital, configurando o SSH Key não é necessário informar usuário e senha do repositório remoto toda vez que vamos fazer alguma interação com ele.

Staging Area

Areá temporária dos arquivos antes do commit.

Workflow

1) Efetuar o **clone** do repositório remoto
2) Alterar os arquivos locais na **working copy**
3) Adicionar os arquivos na **staging area**
4) Efetuar o **commit** dos arquivos alterados
5) Efetuar **push** dos arquivos comitados para o repositório remoto



git help

Se você precisar de ajuda ao usar Git, existem três maneiras de obter a ajuda para qualquer um dos comandos Git:

```
git help {comando}
git {comando} --help
man git- {comando}
```

git config

A primeira coisa que você deve fazer quando instalar o Git é definir o seu nome de usuário e endereço de e-mail. Isso é importante porque todos os commits no Git utilizam essas informações.

```
git config --global user.name "André Justi"
git config --global user.email "justi.andre@gmail.com"
```

git config

Relembrando, você só precisará fazer isso uma vez caso passe a opção --global, pois o Git sempre usará essa informação para qualquer coisa que você faça nesse sistema. Caso você queira sobrepor estas com um nome ou endereço de e-mail diferentes para projetos específicos, você pode executar o comando sem a opção --global quando estiver no próprio projeto.

```
git config user.name "André Kauling Justi"
git config user.email "andre.justi@softplan.com.br"
```

git config

Para ativar as cores nas respostas de comandos, você pode utilizar o seguinte comando:

```
git config --global color.ui true
```

git init

Você pode obter um projeto Git utilizando duas formas principais. A primeira faz uso de um projeto ou diretório existente e o importa para o Git. A segunda clona um repositório Git existente a partir de outro servidor.

Caso você esteja iniciando o monitoramento de um projeto existente com Git, você precisa ir para o diretório do projeto e digitar.

```
git init
```

Isso cria um novo subdiretório chamado .git que contem todos os arquivos necessários de seu repositório — um esqueleto de repositório Git. Neste ponto, nada em seu projeto é monitorado.

git init

Caso você queira começar a controlar o versionamento dos arquivos existentes (diferente de um diretório vazio), você provavelmente deve começar a monitorar esses arquivos e fazer um commit inicial. Você pode realizar isso com poucos comandos.

```
touch .gitignore
git add .gitignore
git commit -m "Versão inicial do projeto"
```

git clone

Para clonar um repositório com Git, você deve usar o comando git clone {url}. Por exemplo, caso você queria clonar a biblioteca spring-cloud-config, você deve executar o seguinte comando:

```
git clone https://github.com/spring-cloud/spring-cloud-config.git
```

git clone

Se você entrar no novo diretório spring-cloud-config, você verá todos os arquivos do projeto, pronto para serem editados ou utilizados. Caso você queira clonar o repositório em um diretório diferente de spring-cloud-config, é possível especificar esse diretório utilizando a opção abaixo:

git clone https://github.com/spring-cloud/spring-cloud-config.git myspring

```
git clone https://github.com/spring-cloud/spring-cloud-config.git myspring
```

Este comando faz exatamente a mesma coisa que o anterior, mas o diretório alvo será chamado myspring.

git add

Quando um repositório é inicialmente clonado, todos os seus arquivos estarão monitorados e inalterados porque você simplesmente os obteve e ainda não os editou. Conforme você edita esses arquivos, o Git passa a vê-los como modificados, porque você os alterou desde seu último commit. Você seleciona esses arquivos modificados e então faz o commit de todas as alterações selecionadas e o ciclo se repete.
Para passar a monitorar um novo arquivo, use o comando git add. Para monitorar o arquivo README, você pode rodar isso:

```
git add README
```

Se você rodar o comando git status, você pode ver que o seu arquivo README agora está sendo monitorado. Os arquivos monitorados serão os que faram parte do commit.

git status

A principal ferramenta utilizada para determinar quais arquivos estão em quais estados é o comando:

```
git status
```

O comando lhe mostra em qual branch você se encontra. Vamos dizer que você adicione um novo arquivo em seu projeto, um simples arquivo README. Caso o arquivo não exista e você execute git status, você verá o arquivo não monitorado dessa forma:

```
# On branch master
# Untracked files:
# (use "git add {file}..." to include in what will be committed)
#
# README
nothing added to commit but untracked files present (use "git add" to track)
```

git status

Você pode ver que o seu novo arquivo README não está sendo monitorado, pois está listado sob o cabeçalho "Untracked files" na saída do comando status. Não monitorado significa basicamente que o Git está vendo um arquivo que não existia na última captura (commit); o Git não vai incluí-lo nas suas capturas de commit até que você o diga explicitamente que assim o faça. Ele faz isso para que você não inclua acidentalmente arquivos binários gerados, ou outros arquivos que você não têm a intenção de incluir. Digamos, que você queira incluir o arquivo README, portanto vamos começar a monitorar este arquivo.

git diff

Se o comando git status for muito vago — você quer saber exatamente o que você alterou, não apenas quais arquivos foram alterados — você pode utilizar o comando.

```
git diff
```

Apesar do comando git status responder essas duas perguntas de maneira geral, o git diff mostra as linhas exatas que foram adicionadas e removidas — o patch, por assim dizer.
Se você quer ver o que selecionou que irá no seu próximo commit, pode utilizar:

```
git diff --cached
```

git commit

Armazena o conteúdo atual do índice em um novo commit, juntamente com uma mensagem de registro do usuário que descreve as mudanças.
Se usa o commit depois de já ter feito o git add, para fazer o commit:

```
git commit -m "Mensagem"
```

Para commitar também os arquivos versionados mesmo não estando no Stage basta adicionar o parâmetro -a

```
git commit -a -m "Mensagem"
```

git commit

Refazendo commit quando esquecer de adicionar um arquivo no Stage:

```
git commit -m "Mensagem" --amend
```

O amend é destrutivo e só deve ser utilizado antes do commit ter sido enviado ao servidor remoto.

git reset

Em qualquer fase, você pode querer desfazer alguma coisa. Aqui, veremos algumas ferramentas básicas para desfazer modificações que você fez. Cuidado, porque você não pode desfazer algumas dessas mudanças. Essa é uma das poucas áreas no Git onde você pode perder algum trabalho se fizer errado.

Para voltar ao último commit:

```
git reset --hard HEAD~1
```

git reset

Para voltar ao último commit e mantém os últimos arquivos no Stage:

```
git reset --soft HEAD~1
```

Volta para o commit com a hash XXXXXXXXXXX:

```
git reset --hard XXXXXXXXXXX
```

git reset

Recuperando commit apagado pelo git reset

Para visualizar os hashs

```
git reflog
```

E para aplicar:

```
git merge {hash}
```

git rm

Para remover um arquivo do Git, você tem que removê-lo dos arquivos que estão sendo monitorados (mais precisamente, removê-lo da sua área de seleção) e então fazer o commit. O comando git rm faz isso e também remove o arquivo do seu diretório para você não ver ele como arquivo não monitorado (untracked file) na próxima vez.

```
git rm -f {arquivo}
```

Se você modificou o arquivo e já o adicionou na área de seleção, você deve forçar a remoção com a opção -f. Essa é uma funcionalidade de segurança para prevenir remoções acidentais de dados que ainda não foram gravados em um snapshot e não podem ser recuperados do Git.

git mv

Diferente de muitos sistemas VCS, o Git não monitora explicitamente arquivos movidos.
É um pouco confuso que o Git tenha um comando mv. Se você quiser renomear um arquivo no Git, você pode fazer isso com

```
git mv arquivo_origem arquivo_destino
```

e funciona. De fato, se você fizer algo desse tipo e consultar o status, você verá que o Git considera que o arquivo foi renomeado.

git mv

No entanto, isso é equivalente a rodar algo como:

```
mv README.txt README
git rm README.txt
git add README
```

O Git descobre que o arquivo foi renomeado implicitamente, então ele não se importa se você renomeou por este caminho ou com o comando mv. A única diferença real é que o comando mv é mais conveniente, executa três passos de uma vez. O mais importante, você pode usar qualquer ferramenta para renomear um arquivo, e usar add/rm depois, antes de consolidar com o commit.

git branch

Um branch no Git é simplesmente um leve ponteiro móvel para um dos commits. O nome do branch padrão no Git é master. Como você inicialmente fez commits, você tem um branch principal (master branch) que aponta para o último commit que você fez. Cada vez que você faz um commit ele avança automaticamente.

O que acontece se você criar um novo branch? Bem, isso cria um novo ponteiro para que você possa se mover. Vamos dizer que você crie um novo branch chamado testing. Você faz isso com o comando git branch:

```
git branch testing
```

git branch

Isso cria um novo ponteiro para o mesmo commit em que você está no momento.
Para mudar para um branch existente, você executa o comando git checkout. Vamos mudar para o novo branch testing:

```
git checkout testing
```

Isto move o HEAD para apontar para o branch testing.

git checkout

Com o git checkout você pode mudar de branch.

```
git checkout master
```

Caso a branch ainda não exista você poderá passar o parâmetro -b para criar.

```
git checkout -b {nome_da_branch}
```

git merge

Suponha que você decidiu que o trabalho na tarefa "Cadastro usuário" está completo e pronto para ser feito o merge na branch master. Para fazer isso, você fará o merge da sua branch #cadastro_usuario. Tudo que você tem a fazer é executar o checkout do branch para onde deseja fazer o merge e então rodar o comando git merge:

```
git checkout master
git merge iss53
```

git mergetool

Se você quer usar uma ferramenta gráfica para resolver os merges, você pode executar o seguinte comando que abre uma ferramenta visual de merge adequada e percorre os conflitos.

```
git mergetool
```

git mergetool cria * .orig arquivos de backup ao resolver fusões. Estes são seguros para remover uma vez que um arquivo foi fundida e sua git mergetool sessão foi concluída.

git log

Depois que você tiver criado vários commits, ou se clonou um repositório com um histórico de commits existente, você provavelmente vai querer ver o que aconteceu. A ferramente mais básica para fazer isso é o comando:

```
git log
```

git stash

Muitas vezes, quando você está trabalhando em uma parte do seu projeto, as coisas estão em um estado confuso e você quer mudar de branch por um tempo para trabalhar em outra coisa. O problema é, você não quer fazer o commit de um trabalho incompleto somente para voltar a ele mais tarde. A resposta para esse problema é o comando git stash.
Você quer mudar de branch, mas não quer fazer o commit do que você ainda está trabalhando; você irá fazer o stash das modificações. Para fazer um novo stash na sua pilha, execute:

```
git stash
```

Seu diretório de trabalho estará limpo.

git stash

Neste momento, você pode facilmente mudar de branch e trabalhar em outra coisa; suas alterações estão armazenadas na sua pilha. Para ver as stashes que você guardou, você pode usar

```
git stash list
```

Você pode aplicar aquele que acabou de fazer o stash com o comando mostrado na saída de ajuda do comando stash original: git stash apply. Se você quer aplicar um dos stashes mais antigos, você pode especificá-lo, assim: git stash apply stash@{2}. Se você não especificar um stash, Git assume que é o stash mais recente e tenta aplicá-lo.

git tag

Git tem a habilidade de criar tags em pontos específicos na história do código como pontos importantes. Geralmente as pessoas usam esta funcionalidade para marcar pontos de release (v1.0, e por aí vai). Nesta seção, você aprenderá como listar as tags disponíveis, como criar novas tags, e quais são os tipos diferentes de tags.
Para listar as tags execute:

```
git tag
```

Para criar uma tag basta executar o seguinte comando, caso não queira criar a tag anotada, somente retire os parâmetros -a e -m.

```
git tag -a v1.0.0-0 -m 'Minha versão 1.0.0-0'
```

git fetch

Para pegar dados dos seus projetos remotos, você pode executar:

```
git fetch origin
```

Esse comando vai até o projeto remoto e pega todos os dados que você ainda não tem. 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.

git pull

Incorpora as alterações de um repositório remoto no branch atual. Em seu modo padrão, git pull é uma abreviação para git fetch seguido de git merge. Por exemplo, se eu estiver em uma branch chamada develop e quiser atualizar caso haja atualizações remotamente:

```
git pull origin develop
```

git push

Envia as alterações locais do repositório local para o servidor remoto, para fazer isso basta executarmos o comando:

```
git push origin {branch}
```

git remote

Para saber qual é o servidor remoto configurado, podemos executar:

```
git remote -v
```

Configurando git mergetool

Para configurar que o merge seja feito a partir de uma ferramenta visual, podemos executar o seguinte comando:

```
git config --global merge.tool {ferramenta}
```

Exemplo:

```
git config --global merge.tool meld
```

Isso ira configurar a ferramenta Meld como a ferramenta padrão para efetuar merges

Configurando chave ssh

Gerar uma nova chave de SSH (diretório ~/.ssh/)

```
ssh-keygen -t rsa -b 4096 -C "{email_cadastro_servidor}"
```

Após a geração podemos ver a chave gerada com o seguinte comando:

```
cat {nome_da_chave}.pub
```

Exemplo de um conteúdo de chave:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDYyiUiZLNA5/UTUSvmXoePBsp03W0ynoZxVdOFnx+wmYI36TEGtr4q
H0e2xzR5HvFjHmrSL+aaknyd6UqJZvgH/O+JY8gakkEVJZmWV6AA5qOCz+O5DRAXlgjCIJFEY7mUpyzGEL5o
+dYQdS8u8f12bzDM57eWXPjzFszEZAKKAXiklZrdsKZJe4MWpDvPtF/z2zfLE0qo2o1mblrgHhTN9tEyCo7E
FoKzZ4SCOtJAD3eswGhA111bRfFa94+4C8d10r3gpNpZ+rPe/w3klcADzVX1w==
andre.justi@softplan.com.br
```

Configurando chave ssh

Adicionar a chave para o ssh-agent

Verificando se ossh-agente está habilitado:

```
eval "$(ssh-agent -s)"
```

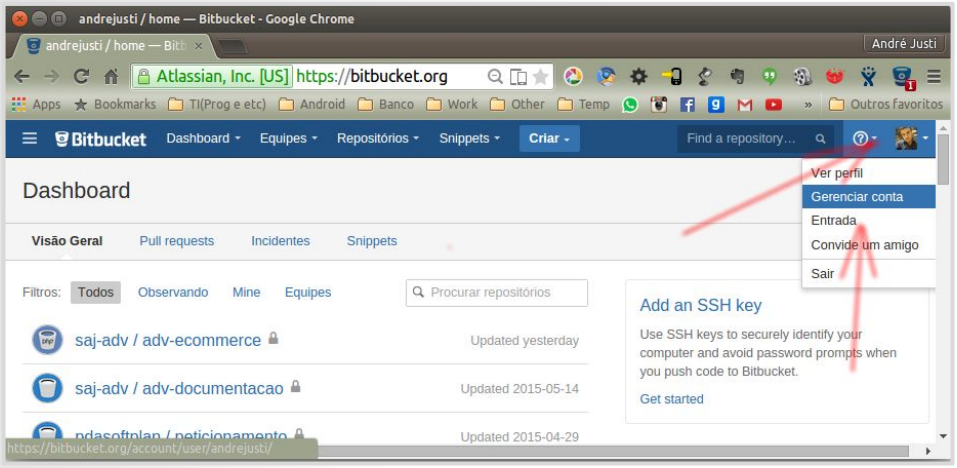
Adicionar a chave ssh gerada para o ssh-agent:

```
ssh-add {nome_da_chave}
```

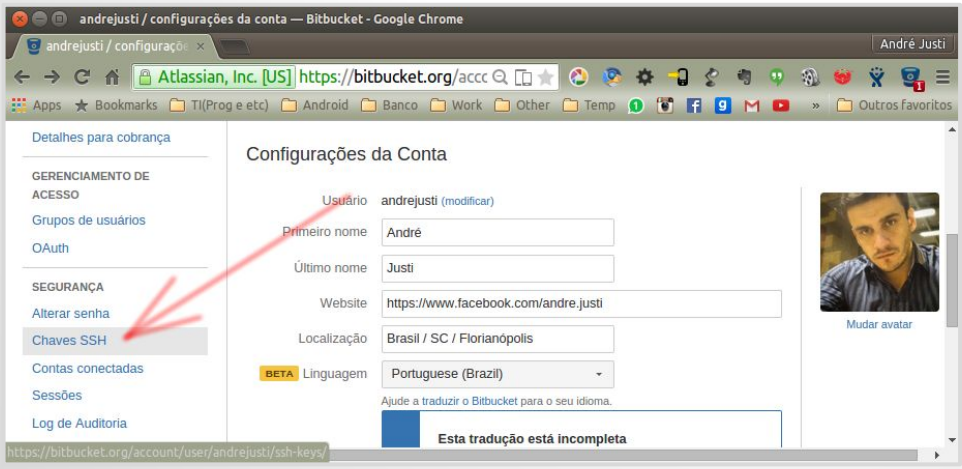
Configurando chave ssh

Adicionando a chave no servidor (BitBucket)

1.



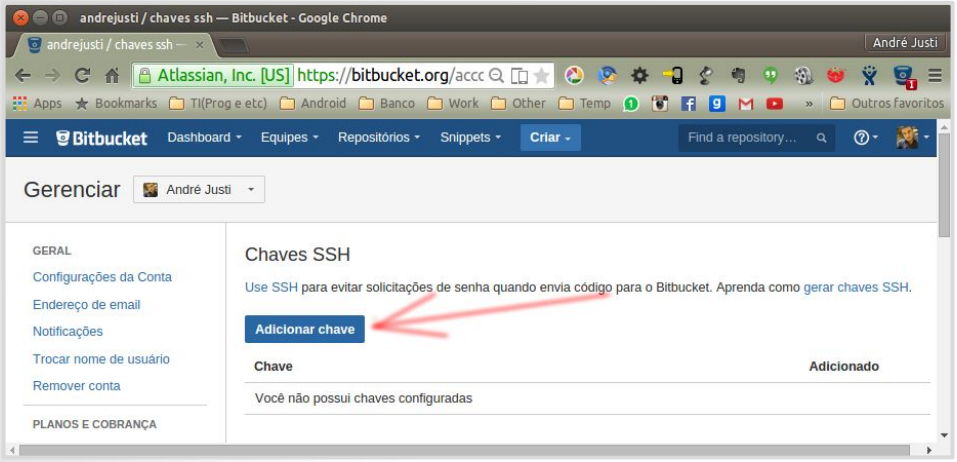
2.



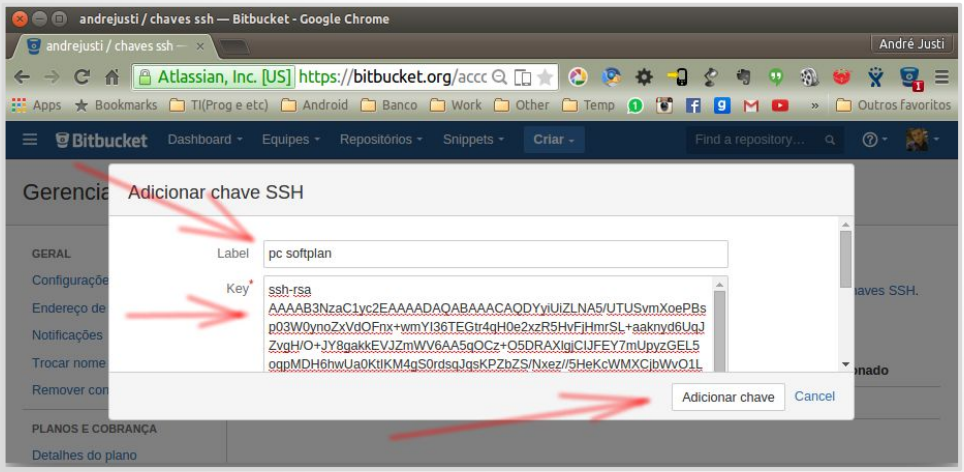
Configurando chave ssh

Adicionando a chave no servidor (BitBucket)

3.



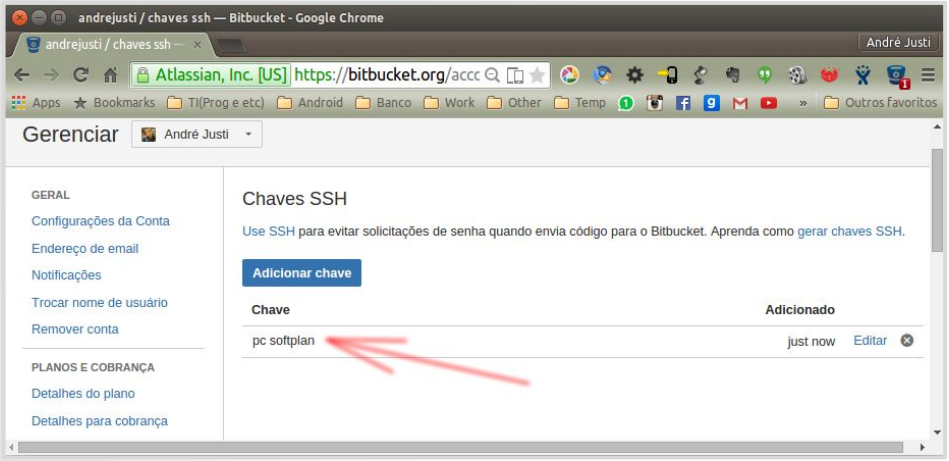
4.



Configurando chave ssh

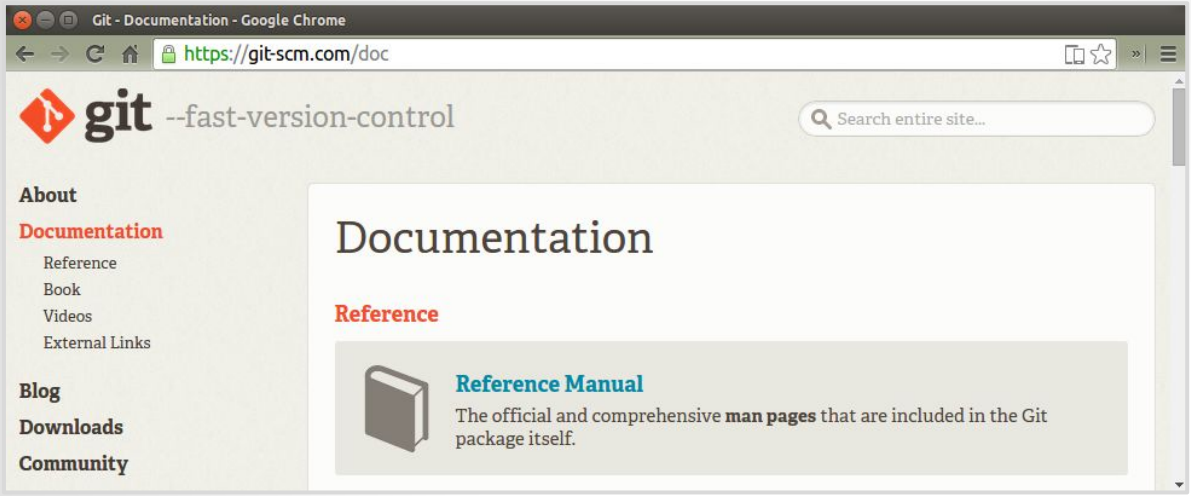
Adicionando a chave no servidor (BitBucket)

5.



Referência

Site oficial: ***https://git-scm.com/doc***



Obrigado!

André Justi

andre.justi@softplan.com.br 48 3027 8000

SOFTPLAN