Projetos Remotos

PET Estatística UFPR

Sumário

Capítulo 1

Projetos Remotos

Nos capítulos anteriores descrevemos como instalar o Git e ter projetos versionados. No entanto, o uso do Git até então foi apenas local. Os arquivos eram mantidos na sua máquina de trabalho e disponíveis só para você.

Os recursos do Git, como o desenvolvimento em *branches*, permite que vários segmentos sejam conduzidos de forma independente e no futuro, quando apropriado, reunidos em um único *branch*. Isso é exatamente o que precisamos para trabalhar em equipe, certo? Se cada colaborador pudesse ter um ramo separado do projeto para trabalhar, todos poderiam trabalhar simultâneamente. Quando oportuno, bastaria fazer merges para reunir o trabalho. A questão é: como deixar o projeto

disponível para os colaboradores?

A resposta é simples: mantenha o projeto em um servidor onde os colaboradores tenham acesso. Isso inclusive vai permitir que você acesse o projeto de várias outras máquinas, como o *notebook* de casa e o desktop do *escritório*.

1.1 Repositório remoto pessoal

O repositório remoto serve de centro do seu repositório Git. Como ele está em um servidor que você tem acesso, você pode compartilhar o repositório com outras máquinas, clonado de lá. Ele serve como *backup* do repositório.

Aqui não se trabalha em colaboração mas o processo permite acessar o repositório, transferir arquivos de várias máquinas suas.

```
## Autenticar no servidor (logar).
ssh eu@servidor

## Verificar se a máquina tem o Git, se não instalar.
git --version

## Criar um diretório para conter o projeto.
mkdir -p ~/meusProjetos/meu1repo
cd ~/meusProjetos/meu1repo
```

```
## Começar um projeto Git remoto. Note a opção --bare.
git --bare init
```

Apenas isso precisa ser feito no servidor. Você não cria nenhum arquivo pelo servidor. Inclusive, muitos dos comandos Git, como `git status` não funcionam para repositório iniciados com a opção `git --bare init`.

Caso queira, você também pode usar `git init`. A diferença entre eles é só onde ficam os arquivos do versionamento. Com `git init`, um diretório oculto `.git/` é o repositório Git e os arquivos de trabalho, como o `README.md`, ficam ao lado dele na estrutura de diretório. Com `git --bare init` o conteúdo do repositório Git fica na raíz. Essa última opção é justamente para criar repositórios remotos que vão justamente manter a parte repositório e não os arquivos.

```
git init                  git --bare init
.                         .
|-- .git                  |-- branches
|   |-- branches          |-- config
|   |-- config            |-- description
|   |-- description       |-- HEAD
|   |-- HEAD              |-- hooks
|   |-- hooks             |-- info
|   |-- info              |-- objects
|   |-- objects           +-- refs
|   +-- refs
+-- README.md
```

Uma vez iniciado o repositório no servidor, todo trabalho passa ser local. É a vez de adicionar o endereço do diretório no servidor e transferir arquivos.

```
## Agora na sua máquina local, adicione o endereço do remoto.
git remote add eu@server:~/meusProjetos/meu1repo

## Exibir os endereços remotos.
git remote -v
```

Esse endereço pode ter IP, porque na realidade, todo servidor tem um IP. Por exemplo, o servidor do tem o IP 192.30.252.129. Para saber o IP é só dar um *ping* no endereço.

```
ping github.com
ping gitlab.com
ping google.com
ping cran.r-project.org
```

Normalmente, servidores de escritório não tem um endereço nominal, apenas o IP (numérico). É necessário registrar domínio para ter nominal.

```
## Agora na sua máquina local, adicione o endereço do remoto.
git remote add eu@111.22.333.44:~/meusProjetos/meu1repo
```

Nesse processo, toda transferência de arquivos vai pedir senha do seu usuário no servidor. Para facilitar, pode-se trabalhar com chaves públicas.

O arquivo `id_rsa.pub` é a sua chave pública. O `id_rsa` é o seu par. RSA é o tipo de encriptação. O nome e caminho do arquivo e a encriptação podem ser outros, depende do que você escolher ao gerar o par de chaves. Normalmente usa-se RSA e as chaves são criadas no diretório `~/.ssh`.

```
## Exibir chaves públicas.
cat ~/.ssh/id_rsa.pub

## Caso não tenha, gerar.
ssh-keygen -t rsa -C "eu@dominio.com"
```

No servidor, o arquivo `authorized_keys2` contém as chaves públicas das máquinas com acesso permitido sem senha. Esse arquivo nada mais é que uma coleção de chaves. O conteúdo dele são as chaves das suas máquinas, ou das máquinas de outros usuários, conteúdo do `id_rsa.pub`, uma embaixo da outra, conforme o exemplo abaixo[1].

```
## `authorized_keys` do servidor da Liga da Justiça.
ssh-rsa IyBUrjvUdSMY... flash@justiceleague.org
ssh-rsa AAAAB3NzaC1y... batman@justiceleague.org
ssh-rsa Mdju17IdXhSd... superman@justiceleague.org
```

[1] O Flash foi o primeiro a transferir as chaves para o servidor porque ele é mais rápido

```
## Logar no servidor
ssh eu@111.22.333.44

## Criar o diretório/arquivo para as chaves públicas.
mkdir ~/.ssh
 > authorized_keys2
```

Agora é preciso transferir o conteúdo do seu arquivo local `id_rsa.pub` para o `authorized_keys2` que fica no servidor. O jeito mais simples de fazer isso é com transferência *scp* mas a instrução *ssh* abaixo também resolve.

```
## Transfere arquivo para diretório temporário.
scp ~/.ssh/id_rsa.pub eu@111.22.333.44:/tmp

## Cola o conteúdo do *.pub no final do authorized_keys.
ssh eu@111.22.333.44\
  "cat /tmp/id_rsa.pub ~/.ssh/authorized_keys"

## Faz os dois passos anteriores de uma vez só.
ssh eu@111.22.333.44\
  "cat >> ~/.ssh/authorized_keys2" < ~/.ssh/id_rsa.pub
```

1.2 Repositório remoto coletivo

A única diferença é recomendamos você criar um novo usuário e adicionar as chaves públicas de todos os membros. Evite

adicionar chaves públicas para usuários na sua conta porque isso expõe seus documentos, alguma operação desastrosa por parte de alguém pode comprometê-los. Por isso, crie um usuário, por exemplo `gitusers`, para na conta manter o repositório remoto.

Solicite que os colaboradores gerem as chaves públicas e te enviem o arquivo `id_rsa.pub`. Depois você adiciona cada chave ao `authorized_keys` da conta `gitusers`. Com chaves autorizadas, os colaboradores podem transferir arquivos, podem logar no servidor mas não podem instalar nada, a menos que você passe a senha do usuário `gitusers`. Para crias usuários no servidor, você precisa de privilégios de *admin*.

```
## Logar na servidora.
ssh eu@servidor

## No servidor, criar uma conta para o projeto.
sudo adduser gitusers
```

Vamos assumir que você têm os arquivos `*.pub` dos colaboradores no diretório `/chaves` devidamente identificados pelo nome deles. O comando abaixo acrescenta as chaves deles uma embaixo da outra no `authorized_keys`.

```
## Entra no diretório com os arquivos *.pub.
## Existem várias: angela.pub, jhenifer.pub, ...
cd chaves
```

```
## Juntar as chaves em um único arquivo.
cat *.pub > todos.pub

## Copiar o conteúdo do arquivo pro authorized_keys2.
ssh gitusers@111.22.333.44\
  "cat >> ~/.ssh/authorized_keys2" < todos.pub
```

1.3 Fluxo de trabalho com repositório remoto, do clone ao push

1.3.1 Git clone

Este comando é usado para clonar um repositório do servidor remoto para um servidor local, caso você queira copiar um repositório que já existe para realizar colaborações em um projeto que queira participar. Você terá acesso a todos os arquivos e poderá verificar as diferentes versões destes. Supondo que sua equipe de trabalho possue uma biblioteca Git **Teste Clone**, onde são armazenados todos os arquivos. Você pode clonar estes arquivos para o seu diretório de trabalho e assim modificá-los conforme deseja.

Exemplo:

```
git clone git@gitlab.c3sl.ufpr.br:pet-estatistica/TesteClone.git
```

Desta forma você terá um diretório `TesteClone` em seu computador, onde estarão todos os arquivos do projeto nele.

Você também terá a opção de clonar o repositório `TesteClone` em um diretório diferente do padrão Git, que no próximo exemplo denominaremos de `DirTeste`:

Exemplo:

```
git clone git@gitlab.c3sl.ufpr.br:pet-estatistica/TesteClone.git DirTes
```

1.3.2 Git Push

Após clonar e realizar contribuições ao projeto, você pode enviá-los para o repositório remoto. Estes arquivos, após o `Git push`, estarão prontos para serem integrados ao projeto com o `merge`.
Usado para transferência de arquivos entre repositório local e o servidor remoto. Como o nome já diz, o comando empurra os arquivos para o servidor remoto. No exemplo abaixo enviaremos a ramificação `Branch Master` para o servidor chamado `origin`:

Exemplo:

```
git push origin master
```

É importante ressaltar que se dois usuários clonarem ao mesmo tempo, realizarem modificações e enviarem os arquivos atuali-

zados ao repositório utilizando o `Git push`, as modificações do usuário que realizou o push por último serão desconsideradas.

1.3.3 Git Pull

Para obter todos os arquivos presentes no projeto, você pode importar os arquivos do branch `master`. Toda vez que você utilizar o `Git pull` a última versão de todos os arquivos estarão no seu diretório. Este comando é utilizado para transferência de arquivos. O comando puxa os arquivos do servidor remoto para o repositório local e faz o merge do mesmo, fundindo a última versão com a versão atualizada.

Exemplo:

```
git pull origin master
```

1.3.4 Git fetch

Assim como o comando `Git pull`, o `Git fetch` transfere arquivos do repositório remoto para o local, porém ele não realiza automaticamente o merge dos arquivos transferidos, o usuário deve fazer o merge manualmente.

Exemplo:

```
git fetch origin master
```

Para verificar as modificações realizadas entre versões de um arquivo basta utilizar o comando `git diff`:

Exemplo:

```
git diff master origin/master
```

1.4 Listar branches locais/remotos

O comando `git remote`, este é usado para verificar quais repositórios estão configurados.

Exemplo: para retornar a lista de repositórios:

```
git remote
```

No comando acima é possível visualizar o remoto padrão **origin** (URL SSH para onde será possível enviar os seus arquivos).

Exemplo: para retornar o nome dos repositórios com a URL onde foram armazanados:

```
git remote -v
```

1.5 Adicionar, renomear, deletar remote

1.5.1 Adicionando repositórios remotos

O comando `git remote add` adiciona um repositório remoto. No exemplo a seguir será adicionado um repositório chamado **MeuRepo** ao qual será vinculado a URL `git@gitlab.c3sl.ufpr.br:pet-estatistica/apostila-git.git`. Usaremos como exemplo o projeto Git **Apostila-git**.

Exemplo:

```
git remote add MeuRepo git@gitlab.c3sl.ufpr.br:pet-estatistica/apo
# Quando executamos novamente o comando para obter a lista de repo
git remote -v
```

Pare acessar localmente o branch master do projeto **Apostila-git** será usado `MeuRepo/master`.

1.5.2 Obtendo informações de um Remoto

Você pode acessar as informações de qualquer repositório remoto com o comando `git remote show`, que retornará a URL e os `branches`.

Exemplo:

```
git remote show origin
```

1.5.3 Renomeado Remotos

O comando `git remote rename` pode modificar o nome de um repositório remoto. A seguir o repositório `MeuRepo`será renomeado para `RenameRepo`.

Exemplo:

```
git remote rename MeuRepo RenameRepo
```

1.5.4 Removendo Remotos

Para remover remotos é utilizado o comando `git remote rm`, agora será removido o repositório renomeado anteriormente `RenameRepo`.

Exemplo:

```
 git remote rm RenameRepo
```

1.5.5 Deletar ramos no servidor

Quando houver um branch que não está sendo utilizado ou está concluído, há a opção de excluí-lo do servidor.

Se for necessário apagar branches remotos, podemos utilizar o comando a seguir:

Exemplo:

```
git push origin --delete <branch>
```

1.5.6 Clonar apenas um *branch, commit* ou *tag.*

É possível clonar apenas um branch e não o repositório Git completo. Supondo que no repositório há um branch chamado `MeuBranch` dentro do repositório `MeuRepo`, clonaremos o branch.

Exemplo:

```
git clone -b MeuBranch --single-branch git://sub.domain.com/MeuRep
```

O Git ainda permite clonar um commit ou tag.

Exemplo:

```
# Para uma tag:
git -e: //git.myproject.org/MyProject.git@v1.0#egg=MyProject
# Para um commit:
git -e: //git.myproject.org/MyProject.git@da39a3ee5e6b4b0d3255bfe
```

1.6 Criando um Repositório Git

Primeiramente é necessário ter acesso a um servidor Linux com chave SSH, no qual você poderá ter seus repositórios. É definido um diretório no qual será armazenado o repositório remoto. No próximo exemplo é preciso criar um repositório remoto chamado `MeuRepo` e o armazenar em um diretório `~/git`:

Exemplo:

```
# Para criar um diretório git na sua home:
mkdir ~/git
# Para criar um repositório git:
mkdir MeuRepo.git
# Para definir MeuRepo como um repositório remoto:
git --bare init
```

As configurações do servidor estão completas. A partir você pode realizar os primeiros comandos para iniciar o repositório criado.

1.7 Git no servidor

Primeiramente, para configurar o Git no Servidor e configurar os protocolos, clonaremos o repositório existente em um repo-

sitório limpo. **Observação:** você poderá colocar um repositório no Servidor se este não contém um diretório de trabalho.

Exemplo:

```
git clone --bare MeuRepo MeuRepo.git
```

Acima foi criado um repositório limpo `MeuRepo.git`, no qual está armazenada a cópia de todos os arquivos do diretorio Git.

Após este primeiro passo o repositório limpo será colocado no Servidor e configurado os protolocos. No exemplo abaixo, supondo que você tem configurado um servidor `git.servidor.com`, e um diretório `/dir/git`no qual você quer armazenar seus repositórios. Ao copiar o seu repositório limpo, você pode configurar seu novo repositório.

Exemplo:

```
scp -r MeuRepo.git usuario@git.example.com:/dir/git
```

Agora o repositório pode ser clonado por outros usuários, que podem ter acesso de escrita e de envio de arquivos `push` no diretório.

```
git clone usuario@git.example.com:/dir/git/MeuRepo.git
```

1.8 Configuração de Conexão SSH com Servidor

O Git possibilita ao usuário realizar uma chave SSH que fará uma conexão segura da sua máquina com o servidor. Para isso começamos com o seguinte comando no terminal:

Exemplo:

```
## Gerando uma chave ssh
ssh-keygen -t rsa -C "usuario@email.com"
```

A partir deste comando, será possível alterar o diretório onde será salva a chave SSH. O usuário tem a opção de permanecer com o diretório padrão, para isso basta apertar Enter. A partir disso, são criados dois arquivos no diretório, o `id_rsa` e o `id_rsa.pub`. Após escolher o diretório onde serão salvos os arquivos, você terá a opção de digitar uma senha ou deixar o espaço em branco.

Para visualizar a chave basta digitar o seguinte comando:

Exemplo:

```
cat ~/.ssh/id_rsa.pub
```

A chave está no arquivo `id_rsa.pub`. O usuário deve copiar o texto deste arquivo na íntegra. Para gerar a conexão ssh com o servidor, deve visitar o site https://gitlab.c3sl.ufpr.br/profile/

keys e clicar em Add SSH Key. É necessário escrever um título para a sua nova chave, no campo `key` colar o texto copiado do arquivo `id_rsa.pub` e adicionar sua nova chave.

Para checar a configuração da sua máquina com o sevidor basta realizar o seguinte comando:

Exemplo:

```
ssh -T git@gitlab.c3sl.ufpr.br
```

Configurando o servidor

Agora será abordado como configurar o acesso SSH do ponto de vista do servidor. Você precisa criar um usuário Git e um diretório `.ssh` para este usuário.

Exemplo: criar usuário e diretório.

```
sudo adduser git
su git
cd
mkdir .ssh
```

Agora, você terá um arquivo chamado `authorized_keys` onde será adicionado uma chave pública de algum desenvolvedor. Após obter chaves de alguns usuários, você pode salvá-las no arquivo `authorized_keys`, como no exemplo a seguir.

Exemplo:

```
# chave do primeiro usuário
cat /tmp/id_rsa1.pub >> ~/.ssh/authorized_keys
# chave do segundo usuário
cat /tmp/id_rsa2.pub >> ~/.ssh/authorized_keys
...
```

Depois de armazenar as chaves dos usuários, basta criar um repositório limpo (sem um diretório de trabalho) para eles. Como visto anteriormente:

Exemplo:

```
cd/dir/git
mkdir NovoProjeto.git
cd NovoProjeto.git
git -bare init
```

Agora os usuários, cujas chaves foram salvas no arquivo `authorized_keys` podem compartilhar arquivos no repositório com os comando `git init`, `git add`, `git commit`, `git remote add` e `git push origin master`.