

Laboratórios de Informática I

2022/2023

Licenciatura em Engenharia Informática

Ficha 3

Sistemas de Controlo de Versões (Continuação)

Apesar da natureza distribuída do sistema de controlo de versões *git*, a existência de um servidor central como ponto de referência do repositório torna a distribuição do código mais simples. Nesta ficha iremos continuar este assunto, tendo em conta um tal servidor comum.

1 GitLab, GitHub, etc.

Serviços como o GitLab¹, GitHub², SourceHut³, Bitbucket⁴ entre outros, permitem que os utilizadores alojem um repositório *git* de forma pública (aberto a toda a Internet) ou privada (disponível apenas a certos outros utilizadores). Outra grande vantagem destes serviços é permitirem realizar tarefas de manutenção do repositório, tais como:

1. Disponibilizarem uma *Wiki* para fins de documentação;
2. Permitirem navegar nos vários *commits* do repositório, facilitando assim a consulta de como o código evoluiu;
3. Disponibilizarem um gestor de *bugs/issues*, onde podemos documentar problemas no código, discutí-los e, mais tarde, associar a sua resolução a um *commit*;
4. Disponibilizarem um interface de *code review* onde se torna possível estudar o código contribuído por um utilizador antes da sua integração.

1.1 Uso destes serviços

Antes de podermos usufruir de qualquer um destes serviços é necessário criar uma conta de utilizador. Note que o e-mail que utilizar neste registo deve coincidir com o e-mail institucional configurado no *Git* na sessão anterior, por forma a que o registo dos commits seja devidamente acompanhado.

Criada uma conta num destes serviços, podemos agora criar um novo repositório *git* de raiz, clonar um repositório já existente para a nossa máquina, ou criar um novo repositório a partir de outro (*fork*).

¹<https://gitlab.com>

²<https://github.com>

³<https://sr.ht>

⁴<https://bitbucket.org/>

1.2 Clone

Na aula passada vimos como criar um repositório *git* de raiz na nossa máquina. Vamos agora ver como clonar para a nossa máquina um repositório já existente .

O comando para tal é `git clone` seguido do endereço do repositório. Por exemplo, se quiser clonar o repositório individual criado para um aluno número aXXXXXX, deverá executar o seguinte comando (endereço SSH):

```
$ git clone git@gitlab.com:uminho-di/li1/2223/acompanhamento/aXXXXXX.git
```

ou (endereço HTTPS)

```
$ git clone https://gitlab.com/uminho-di/li1/2223/acompanhamento/aXXXXXX.git
```

e inserir a respectiva *password* correctamente.

O resultado da execução deste comando é uma pasta, criada na pasta atual, com o nome do repositório (aXXXXXX). Esta pasta será a raiz do repositório e terá a mesma estrutura do repositório clonado do servidor, que neste caso está vazio.

Este comando `git clone` deverá ser efectuado uma única vez, sendo as alterações futuras ao repositório central geridas através de comandos como, por exemplo, `git push` e `git pull`.

1.3 Push

Para enviar as alterações e os novos ficheiros ou pastas para o servidor central é necessário realizar um processo designado por *push*. Isto poderá ser feito através do comando `git push`. Apenas após a execução deste comando o código estará acessível por todos os utilizadores do repositório.

Tarefas

1. Crie ou use uma conta (com o seu e-mail institucional) no GitLab.
2. Faça clone do repositório aí existente (identificado com o seu número de aluno aXXXXXX).
3. Verifique que foi criada localmente a directoria aXXXXXX.
4. Crie o ficheiro `README.md` na directoria aXXXXXX com o seguinte texto:

```
# Aulas LI1  
Por: <nome> <email-institucional>
```

5. Registe, i.e. faça `commit` desta alteração com a mensagem "`README.md criado`" (depois de fazer `add`, como vimos na aula anterior).
6. Faça *push* para o servidor, escrevendo `git push`.
7. Dirija-se ao repositório aXXXXXX no painel de controlo do GitLab e confirme que as alterações detectadas pelo GitLab estão de acordo com a alteração efectuada localmente.
8. Crie no repositório um ficheiro `exemplo.txt` com um texto à sua escolha. Actualize o repositório local. Actualize o repositório remoto. Verifique no GitLab as alterações efectuadas.

1.4 Pull

Sendo que o *git* é especialmente útil no desenvolvimento cooperativo, vamos ver o que acontece quando um outro utilizador altera o repositório. Suponhamos então que um outro utilizador apagou o ficheiro `exemplo.txt` ou alterou o ficheiro `README.md`.

Um utilizador de *git* deve atualizar a sua cópia local do repositório sempre que possível e no mínimo antes de iniciar uma sessão de trabalho, para que quaisquer alterações que tenham sido incluídas por outros programadores sejam propagadas do repositório central para a sua cópia local.

Este processo é feito através do comando `git pull`.

Tarefas

1. Apague o ficheiro `exemplo.txt` no repositório remoto (através do painel de controlo do GitLab).
2. Verifique que o ficheiro ainda se encontra no repositório local.
3. Faça agora `git pull`. Verifique que o repositório local foi actualizado e já não contém o ficheiro `exemplo.txt`.

Nem sempre os programadores estão a trabalhar em ficheiros distintos. Supondo que dois utilizadores alteraram o mesmo ficheiro em simultâneo, mas em zonas diferentes do ficheiro (por exemplo, cada um modificou apenas o seu nome no ficheiro `README.md`), e que o primeiro já fez *push* da sua alteração. O *git* é capaz de lidar com a alteração concorrente sem problemas, e portanto, poderá ser feito o *push* destas últimas alterações.

Em algumas situações poderá acontecer que dois utilizadores tenham editado a mesma zona do ficheiro, e portanto, que o *git* não tenha conseguido juntar as duas versões. Nesta situação, ao realizar o *pull* terá de gerir o conflito manualmente.

Ao editar um ficheiro com conflito aparecerão marcas deste género:

```
codigo haskell muito bom
<<<<<< HEAD
mais código
=====
mais código do outro utilizador
>>>>>> branch-a
```

Isto indica a zona com conflito. Uma versão é o texto entre as marcas `<<<<<<` e `=====` e a outra versão é o texto entre as marcas `=====` e `>>>>>>`. Nesta situação é nosso dever remover as marcas (as linhas com `<<<<<<`, `=====` e `>>>>>>`) e optar por uma das partes (ou então, criar uma nova que resolva o conflito).

Depois de resolver um conflito o utilizador deverá indicar que o conflito foi resolvido:

```
$ git add README.md
$ git commit -m "conflito resolvido"
$ git push
```

1.5 Remove

Vamos agora ver como remover ficheiros do repositório central. Como vimos anteriormente isto poderá ser feito através do comando `git rm`, seguido do nome do ficheiro. Tal como na operação *add*, temos que fazer *commit* e *push* para que um ficheiro marcado para ser apagado seja efetivamente apagado no repositório central.

Note que embora o ficheiro tenha sido removido da directoria de trabalho, ele ficará guardado no servidor. Portanto se tal for necessário é possível reaver o ficheiro.

Note que para que o `git log` apresente a informação actualizada deverá sempre fazer `git pull` antes. Se existirem *commits* locais e existirem *commits* remotos que não existem localmente, devemos fazer `git pull --rebase` para que o nosso *commit* passe a ser o mais recente e não seja criado um *merge commit* desnecessariamente.

2 Boas práticas

2.1 Branches

Quer tenha criado um novo repositório ou clonado para si um já existente, antes de começar a fazer contribuições é boa prática criar uma nova *branch*. Será nessa nova *branch* que irá registar os seus *commits* (contribuições).

O comando para criar uma nova *branch* a partir da actual e ficar posicionado na *branch* criada é:

```
git checkout -b <nome-da-branch>
```

Por exemplo, fazendo

```
$ git checkout -b Carla/Exercicio4
Switched to a new branch 'Carla/Exercicio4'
```

estaremos a criar uma nova *branch* denominada *Carla/Exercicio4*. A partir daqui, deve fazer os seus *commits* como habitualmente faria. Desta forma, quando mais tarde quiser partilhar com os restantes utilizadores as suas contribuições, apenas tem de lhes indicar esta sua *branch*.

Pode consultar as *branches* definidas escrevendo

```
git branch
```

2.2 Pull/Merge Request

Suponha agora que já terminou tudo o que pretendia fazer em relação à *branch* *Carla/Exercicio4*, e quer agora integrar as suas alterações junto do repositório central.

O primeiro passo a fazer é enviar a sua *branch* para o repositório. Lembre-se que o *git* é distribuído, por isso enquanto não fizer este passo a sua *branch* existirá apenas na sua máquina. Para tal, basta-nos invocar o comando *push*. Quando queremos fazer *push* de uma *branch* que ainda não existe remotamente ou quando se cria uma localmente e nunca se fez *push/pull*, é preciso indicar o *remote* e o nome da *branch*. No exemplo acima o comando é:

```
$ git push origin Carla/Exercicio4
```

Também é possível passar uma flag `--set-upstream` para que nos *commits* seguintes seja só fazer *push*. Fazendo:

```
git push --set-upstream origin Carla/Exercicio4
```

os *pushes* seguintes seriam só `git push`.

Se tiver sido feito clone, o origin aponta para o repositório central (no nosso caso o repositório no GitLab). Se ainda não existir nenhum remote, pode ser adicionado fazendo:

```
$ git remote add origin https://gitlab.com/uminho-di/li1/2223/acompanhamento/aXXXXXX.git
```

onde `origin` podia ser qualquer nome.

Agora que a sua *branch* já existe no repositório central, poderá criar um *merge request* (*pull request* em alguns sistemas). Um *merge request* consiste num requerimento para integrar as contribuições de uma *branch* noutra, onde os demais utilizadores têm a oportunidade para inspeccionar o código antes de o aceitarem.

Para criar um *merge request*, deve dirigir-se ao site onde alojou o repositório e procurar por esta funcionalidade no painel de controlo do seu repositório. Ser-lhe-á questionado qual a *branch* (e.g. `Carla/Exercicio4`) de origem e qual a *branch* de destino (e.g. `main`).

Tarefas

1. Crie uma nova branch local, denominada **NovoReadme**.
2. Edite o ficheiro `README.md` por forma a conter o seguinte texto

```
# Aulas LI1
Experiência com branch.
Por: <nome> <mail>
```

3. Registe, i.e. faça commit desta alteração com a mensagem `README.md reescrito` (depois de fazer *add*).
4. Faça *push* da branch **NovoReadme** para o servidor.
5. Dirija-se ao painel de controlo do GitLab para o repositório `aXXXXXX`, e inicie um novo *Pull/Merge request*. Confirme que as alterações detectadas pelo GitLab estão de acordo, introduza uma mensagem descritiva e crie finalmente o *pull/merge request*.
6. Agora que o *pull/merge request* foi criado, inspeccione-o, e aprove-o.
7. No seu repositório local, troque agora para a *branch* principal, fazendo

```
$ git checkout <nome-da-branch-principal>
```

8. Invoque `git fetch` para sincronizar o repositório local com as alterações do servidor, e de seguida invoque `git status` para verificar se pode importar novas alterações.
9. Invoque `git pull` para trazer as alterações mais recentes para o seu repositório local.