Laboratórios de Informática I
2024/2025

Licenciatura em Engenharia Informática

Ficha 7
Sistemas de Controlo de Versões

O desenvolvimento de *software* é cada vez mais complexo, e obriga a que uma equipa de programadores possa desenvolver uma mesma aplicação ao mesmo tempo, sem se preocuparem com os detalhes do que outros membros dessa mesma equipa estejam a fazer. Alterações concorrentes (realizadas por diferentes pessoas ao mesmo tempo) podem provocar conflitos quando várias pessoas editam o mesmo bocado de código.

Além disso, não nos devemos esquecer que algumas alterações a um programa, no sentido de corrigir ou introduzir alguma funcionalidade, podem elas mesmas conter erros, e pode por isso ser necessário repor uma versão prévia da aplicação, anterior a essa alteração.

Para colmatar estes problemas são usados sistemas de controlo de versões.

1 Panorama nos Sistemas de Controlo de Versões

Existe um grande conjunto de sistemas que permitem o desenvolvimento cooperativo de *software*. Todos eles apresentam diferentes funcionalidades mas os seus principais objetivos são exatamente os mesmos.

Habitualmente divide-se este conjunto em dois, um conjunto de sistemas denominados de *centralizados*, e um outro de sistemas *distribuídos*:

- Sistemas de controlo de versões centralizados:
  - Concurrent Versions System (CVS): `http://www.nongnu.org/cvs/`;
  - Subversion (SVN): `https://subversion.apache.org/`;
- Sistemas de controlo de versões distribuídos:
  - Git: `http://git-scm.com/`;
  - Mercurial (hg): `http://mercurial.selenic.com/`;
  - Bazaar (bzr): `http://bazaar.canonical.com/en/`;

Estes são apenas alguns exemplos dos mais usados. A grande diferença entre os centralizados e os distribuídos é que, nos centralizados existe um repositório, denominado de servidor, que armazena, a todo o momento, a versão mais recente do código fonte. Por sua vez, nos distribuídos, cada utilizador tem a sua própria cópia do repositório, que podem divergir, havendo posteriormente métodos para juntar repositórios distintos.

2 Instalação do Git

Na disciplina de Laboratórios de Informática I será utilizado o sistema `Git`. Para o instalar siga as instruções em `https://git-scm.com/downloads`.

3 Configurar o git

Para configurar o `Git` ao nível do sistema deve indicar o nome e o email que ficarão associados às actividades realizadas no repositório.

```
$ git config --global user.name "a999999"
```

Configura o nome que irá ficar associado aos git commits, `a999999` neste exemplo.

```
$ git config --global user.email "a999999@alunos.uminho.pt"
```

Configura o email que irá ficar associado aos git commits, `a999999@alunos.uminho.pt` neste exemplo.

Poderá consultar a configuração actual com:

```
git config --list
```

4 Uso do Git

Os ficheiros de uma directoria de trabalho controlada pelo `Git` podem estar no estado *tracked* (fazem parte do repositório ou estão na área de preparação (*staged area*) para que possam ser incluidos no repositório) ou *untracked* (detectadas pelo `Git`, mas desconhecidos do repositório). Os ficheiros no estado *tracked* podem estar estar *unmodified* (actualizados no repositório), *modified* (modificados desde a última actualização do repositório), or *staged* (marcados como preparados para inclusão no repositório). Para actualizar o repositório, os ficheiros no estado *tracked - staged* terão de ser convertidos em *tracked - unmodified*. Vamos de seguida analisar como o `Git` gere este ciclo de transformações.

4.1 Init

Vamos criar um repositório local chamado `AulasLI1` para experimentar alguns comandos do `Git`.

```
$ git init AulasLI1
```

Verifique que foi criada na directoria actual a subdirectoria `AulasLI1/`.

4.2 Adição de novas directorias e ficheiros

O passo seguinte corresponde a adicionar novos ficheiros ou pastas que queiramos armazenar no repositório. Como exemplo, vamos adicionar um ficheiro `README.md` ao repositório. Crie na directoria `AulasLI1` um ficheiro chamado `README.md`, e escreva no ficheiro o seu nome completo. Este ficheiro ainda não faz parte do repositório (está no estado *untracked*).

Um comando extremamente simples, mas bastante útil, designado por `status`, permite ver o estado atual do repositório local (sem realizar qualquer ligação ao servidor)

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add" to track)
```

Isto indica que o `Git` detecta o ficheiro `README.md`, mas não sabe nada sobre ele.

Para adicionar o ficheiro `README.md` ao repositório terá de começar por executar o comando:

```
$ git add README.md
```

O ficheiro foi adicionado à área de preparação para que possa ser incluído no repositório, mas ainda não faz parte do repositório controlado pelo `Git`. Como veremos, tal só acontecerá quando for executado o comando *commit*.
Se executar:

```
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
```

a mensagem indica que há um novo ficheiro pronto a ser inserido no repositório. Indica também forma de o remover da *staged area*.

Sempre que realizar alterações a um ficheiro e as quiser registar, terá de dar essa informação ao `Git`. O mesmo comando também é usado para adicionar novos ficheiros ao repositório. Assim, depois de terminar as alterações a um ficheiro (novo ou não), deve executar o comando `git add`.

Uma boa prática de desenvolvimento é adquirir o hábito de gerir todo o código que programar para o projeto através do sistema de versões.

Tarefas

1. Crie um novo ficheiro chamado `exemplo.txt` com um qualquer conteúdo.
2. Altere o ficheiro `README.md` adicionando o número de aluno.
3. Execute o comando `git status` e verifique que obtém:

   ```
   $ git status
   On branch master
   No commits yet
   Changes to be committed:
   ```

```
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
        modified:   README.md
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        exemplo.txt
```

Isto indica que o `Git` não sabe nada sobre o ficheiro `exemplo.txt` e que portanto o irá ignorar em qualquer comando executado. É também assinalado que o ficheiro `README.md` foi modificado e que é necessário fazer *add* para actualizar a versão pronta a submeter ao repositório.

4. Faça agora:

```
$ git add README.md
$ git add exemplo.txt
```

5. Volte a executar o comando status devendo obter:

```
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
        new file:   exemplo.txt
```

indicando que ambos os ficheiros estão agora sob controlo do Git.

4.3 Commit

Para registar as alterações e os novos ficheiros ou pastas no repositório local é necessário realizar um processo designado por *commit* [1]. Isto poderá ser feito através do comando `git commit`.

```
$ git commit -m "Adicionados os ficheiros README.md e exemplo.txt"
[master (root-commit) 72e94ad] Adicionados os ficheiros README.md e exemplo.txt
 2 files changed, 4 insertions(+)
 create mode 100644 README.md
 create mode 100644 exemplo.txt
```

No comando *commit* executado foi adicionada uma opção (`-m`) que é usada para incluir uma mensagem explicativa das alterações que foram introduzidas ao repositório. Se escrever apenas `git commit` será enviado para um editor de texto (e.g. Vim) onde terá de escrever a mensagem a associar ao *commit*.

[1] A executar após o add

É boa prática adicionar uma mensagem clara em cada *commit*.

Deve realizar um *commit* sempre que faça alterações ao seu código que em conjunto formem uma modificação coerente. Os seguintes exemplos podem originar novos commits:

- adicionar uma nova função;
- adicionar um nova funcionalidade;
- corrigir um bug;

Note que depois do *commit* o código está no repositório, mas apenas na sua cópia local.

4.4 Diff

O comando `git diff` mostra as diferenças linha a linha dos ficheiros alterados (antes do add).

Altere o ficheiro `README.md` acrescentando o turno de que faz parte e alterando o número de aluno (troque o "a"por "A"ou o inverso).
Execute:

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
        modified:   README.md
no changes added to commit (use "git add" and/or "git commit -a")
```

e pode comprovar que há alterações identificadas.
Execute agora:

```
$ git diff
diff --git a/README.md b/README.md
index 3addcd0..2442e7d 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,3 @@
 Olga Pacheco
-a99999
+A99999
+Turno: PL10
```

para identificar as diferenças entre as versões do ficheiro `README.md` (repare nas cores, no que foi substituido e acrescentado).

Se executar novamente `git add README.md` e de seguida `git diff`, verificará que não há agora diferenças identificadas. Se fizer `git status`, verificará que o ficheiro actualizado `README.md` está pronto a ser submetido ao repositório, o que acontecerá logo que faça `git commit -m "alteração README.md"`.

4.5 Log

O comando `git log` lista o histórico de versões. Depois das alterações acima mencionadas, obtemos:

```
$ git log
commit 68be25ae3ab33f1c4f781ac21042a9253372dd58 (HEAD -> master)
Author: omp <omp@di.uminho.pt>
Date:   Sat Nov 2 11:26:26 2024 +0000

    alteração de README.md

commit 3325b65bbbb38e9eb5621d10dbae763d1321850f
Author: omp <omp@di.uminho.pt>
Date:   Sat Nov 2 11:21:27 2024 +0000

    Adicionados os ficheiros README.md e exemplo.txt
```

Há uma grande variedade de opções para este comando. Para mais detalhes pode consultar: `https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History`.

4.6 Remove

Vamos agora ver como remover ficheiros do repositório. Isto poderá ser feito através do comando `git rm`, seguido do nome do ficheiro. Para remover o ficheiro `exemplo.txt` faz-se:

```
    $ git rm exemplo.txt
rm 'exemplo.txt'
```

Tal como na operação *add*, temos que fazer *commit* para que um ficheiro marcado para ser apagado seja efetivamente apagado no repositório local.

As mensagens do `Git` dão informações detalhadas sobre formas de remover ou reverter alterações a ficheiros, nas diferentes situações.

NOTA: Para esclarecer dúvidas sobre algum <comando> poderá escrever: `git help <comando>`.

Referências

Para informação mais detalhada sugere-se a consulta de documentação do `Git`, nomeadamente:

```
https://git-scm.com/doc
```