Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

© Pedro R. M. Inácio (inacio@di.ubi.pt), Tiago M. C. Simões (tsimoes@di.ubi.pt)
e Tiago Roxo (tiago.roxo@ubi.pt)

Laboratórios de Programação
Guia para Aula Laboratorial 7
Licenciatura em Engenharia Informática

Programming Laboratories
Guide for Laboratory Class 7
Degree in Computer Science and Engineering

Sumário
Gestão de versões em projetos de programação utilizando o sistema Git.

Summary
Version management in programming projects using the Git system.

Pré-requisitos:
As tarefas propostas a seguir presumem o acesso a um sistema operativo com o *software* Git ou com permissões para a sua instalação. Todas as tarefas foram testadas com sucesso em ambiente Linux mas devem funcionar noutros sistemas, assumindo que as ferramentas necessárias estão instaladas. Alguns exercícios requerem ainda que seja feita uma conta no github.

1 Introdução ao Sistema de Controlo de Versões – Git

Introduction to the Version Control System – Git

O Git, inicialmente desenvolvido por Linus Torvalds, é um sistema de controlo de versões distribuído utilizado para o desenvolvimento de *software*. No entanto, o mesmo pode ser utilizado para manter um registo histórico de qualquer tipo de ficheiros (*e.g.* livros, artigos, *Curriculum Vitae*).

O GitHub é uma plataforma que permite alojar código fonte com suporte ao sistema de controlo de versões Git. Este permite que utilizadores possam contribuir para o desenvolvimento de projetos privados e/ou *open source*.

Tarefa 1 *Task 1*

Dirija-se a `https://github.com/` e crie uma conta gratuita no GitHub. Escreva nos espaços seguintes o nome do utilizador (*username*) e o endereço de *e-mail* que utilizou no registo:

Username: _____

E-mail: _____

O Git pode ser instalado em sistemas operativos Windows, Linux ou macOS. Há instruções detalhadas de instalação em `https://git-scm.com/downloads`. Por exemplo, na distribuição Ubuntu, o Git pode ser instalado utilizando o comando `sudo apt-get install git`.

Tarefa 2 *Task 2*

Verifique o manual do Git utilizando o comando `man git`.

Q1.: De que forma é que o Git é definido no seu manual?
- ☐ Como uma ferramenta de gestão de versões.
- ☐ Como um editor de texto.
- ☐ Como um gestor de conteúdos estúpido.

Tarefa 3 *Task 3*

Crie uma diretoria para um novo projeto chamada `Proj1`, e entre nessa diretoria.

Q2.: Para recordar, quais foram os comandos que utilizou para realizar a tarefa anterior ?
- ☐ `mkdir Proj1` e `cd Proj1`
- ☐ `cd Proj1` e `mkdir Proj1`
- ☐ `cd_to_dir Proj1` e `make_my_dir Proj1`

Tarefa 4 *Task 4*

Inicialize o Git nessa diretoria, utilizando o comando

`git init`, e verifique que foi criada uma nova diretoria escondida nessa mesma pasta com `ls -a .`

Q3.: Para recordar: de que forma é que sistemas Unix ou Unix-*like* simbolizam ficheiros ou diretorias escondidas?

- ☐ Os nomes desses ficheiros ou diretorias aparecem a vermelho, tal como o meu amado Benfica.
- ☐ Os nomes desses ficheiros ou diretorias aparecem a verde, tal como o meu amado Sporting.
- ☐ Os nomes desses ficheiros ou diretorias aparecem a azul, tal como o meu amado Porto.
- ☐ Os nomes desses ficheiros ou diretorias começam com um **ponto**.
- ☐ Os nomes desses ficheiros ou diretorias têm todas as letras capitalizadas.

Tarefa 5 *Task 5*

Antes de prosseguir, e para estabelecer uma linha base, verifique o estado da gestão de conteúdos na sua diretoria com `git status`.

2 Gestão de Ficheiros e Versões

File and Version Management

Tarefa 6 *Task 6*

Crie um novo ficheiro chamado `program.c`, abrindo-o imediatamente para escrita utilizando o comando `nano program.c`. Adicione e guarde o seguinte código no seu ficheiro:

```c
#include<stdio.h>

int main(){
 printf("Hello World!");
 return 0;
}
```

Tarefa 7 *Task 7*

Verifique o estado atual do seu repositório Git utilizando o comando `git status`.

Q4.: Qual foi o *output* do comando realizado na tarefa anterior?

- ☐ O *status* está mau. Isto está complicado e eu tenho de abandonar, porque tenho uma consulta às 5.
- ☐ É indicado que existem *untracked files* e que ainda não existem *commits*.
- ☐ São dadas as indicações do que deve ser feito a seguir.
- ☐ Não obtive qualquer tipo de *output* utilizando o referido comando.

Tarefa 8 *Task 8*

Para **inicializar a monitorização** do ficheiro `program.c` considere emitir o comando `git add program.c`.

Após executar a tarefa anterior, execute novamente o comando `git status`. **Q5.: Qual é agora o seu *output*?**

- ☐ É indicado que existem *untracked files* e que ainda não existem *commits*.
- ☐ É indicado que ainda não existem *commits*, mas já não se queixa de *untracked files*.
- ☐ São dadas as indicações do que deve ser feito a seguir.
- ☐ Não obtive qualquer tipo de *output* utilizando o referido comando. Apenas um silêncio constrangedor...

Tarefa 9 *Task 9*

Se tiver a certeza que o programa está numa forma estável, emita a sua primeira ordem de consolidação, representado por `Commit` ao longo deste guia.

Q6.: Qual foi o comando que utilizou para executar a tarefa anterior?

- ☐ `git commit -m "My initial commit of program.c"`
- ☐ `git_my_program commit -m "My initial commit of program.c"`
- ☐ `commit -m "My initial commit of program.c"`
- ☐ `cc -o program program.c`

Q7.: Qual foi o *output* do comando que executou na tarefa anterior?

- ☐ Não me dei conta que era para usar um comando...
- ☐ Impecável, tudo a funcionar!
- ☐ Foi me dito que tenho que adicionar uma conta de utilizador.

Tarefa 10 *Task 10*

No primeiro `Commit` feito numa máquina, o Git **pede** sempre **informação acerca do utilizador**, nomeadamente o nome de utilizador *e-email*. Se já tiver uma conta no GitHub, utilize os mesmos valores. Para ajustar esta informação, terá de utilizar os seguintes comandos:

```
git config user.email my@email.com
git config user.name us3rn4me
```

Pode, alternativamente, guardar estes valores ao nível do sistema, para que os mesmos possam ser utilizados em utilizações futuras e noutros projetos. Para obter esse efeito, pode adicionar a opção `--global` aos comandos anteriormente referidos, *e.g.*,

```
git config --global user.email my@email.com
git config --global user.name us3rn4me
```

Tarefa 11 *Task 11*

Após realizar a tarefa anterior, execute novamente o comando de consolidação para a efetivar:

```
git commit -m "My initial commit of program.c"
```

.

Verifique o estado do sistema de gestão de versões com `git status`. **Q8.: Qual o efeito obtido?**

- ☐ Escrevi `commit` só com um *m*... e não deu. Mas já vou emendar o erro!
- ☐ Satisfação: o meu o primeiro *commit* foi feito com sucesso.
- ☐ Tristeza: foi me dito outra vez que tenho que adicionar uma conta de utilizador. :(

Tarefa 12 *Task 12*

Experimente compilar o programa em C que acabou de realizar utilizando o comando `cc program.c -o program.exe`.

Irá verificar que, ao executar o programa, a *string* `"Hello World!"` é escrita no ecrã, mas sem quebra de linha para a *prompt*. Será, portanto, necessário fazer mais uma alteração ao programa.

Tarefa 13 *Task 13*

Abra o ficheiro `program.c` para edição (*i.e.* `nano program.c`) e altere a linha do `printf` de:

```
printf("Hello World!");
```

para

```
printf("Hello World!\n");
```

No final, saia e salve. Volte a compilar e a executar, verificando se já obtém o efeito esperado.

Tarefa 14 *Task 14*

Q9.: Se emitir o comando `git status` irá verificar que o Git reporta dois tipos de ficheiros na diretoria. Que tipos são esses?

☐ *not staged* ☐ *not tracked* ☐ *not pretty* ☐ *not easy*

Repare que não esta interessado em monitorizar o ficheiro `.exe`, já que esse ficheiro é apenas o resultado da compilação, mas está sim interessado em fazer `Commit` do ficheiro `program.c`, já que esse ficheiro é importante para o projeto. A próxima tarefa endereça ambas as situações.

Tarefa 15 *Task 15*

Execute os seguintes passos:

1. Crie um ficheiro chamado `.gitignore` e coloque o nome do ficheiro que não quer monitorizar lá dentro. Este procedimento irá evitar que o Git se volte a queixar da existência deste ficheiro;

2. Adicione os dois novos ficheiros modificados (o `program.c` e o `.gitignore` à área de *staging* (encenação) e faça o `Commit` , *i.e.*, execute os comandos seguintes

   ```
   git add program.c .gitignore
   git commit -m "Added newline and a new .gitignore file"
   ```

Verifique se as providências tomadas tiveram efeito emitindo novamente o comando `git status`.

Tarefa 16 *Task 16*

Antes de terminar esta parte introdutória, experimente ainda o comando que lhe permite ver a história dos *commits* que já fez no seu projeto:

```
git log
```

Q10.: Quantos *commits* já fez até aqui?

☐ Resmas! ☐ 0 ☐ 1 ☐ 2 ☐ 3
☐ Cachalotes! ☐ 4

3 Navegação entre Consolidações – *Reset*

Browsing between Commits – Reset

Tarefa 17 *Task 17*

Simule uma alteração que vai querer desfazer *a posteriori*. Para isso, abra o ficheiro `programa.c` e **apague** a linha que declara a função `main`.

Tarefa 18 *Task 18*

Após realizar a tarefa anterior, encene (*add*) o ficheiro novamente utilizando o seguinte comando: `git add program.c`. Não se esqueça de realizar `Commit` após encenar/adicionar o ficheiro (*i.e.*, `git commit -m "I think an error was made."`).

Verifique o *log*, e identifique o valor da consolidação **para onde quer voltar** (este deve ser um valor hexadecimal bastante longo). **Q11.: Copie a parte inicial (*e.g.*, 6 primeiros caracteres) desse *commit* para o espaço em baixo:**

Tarefa 19 *Task 19*

Realize um *reset* do seu projeto para um *commit* anterior à introdução do erro no seu código. Para este efeito utilize o comando semelhante a `git reset --hard <id do commit>`.

Tarefa 20 *Task 20*

Após realizar a tarefa anterior, verifique se o efeito das alterações no ficheiro desapareceu, e se já desapareceu do *log* o *commit* errado.

Q12.: Confirma que a função `main()` está de volta ao `program.c`?

- ☐ *Woo Hoo!* Confirmo!
- ☐ Não confirmo nem desminto.
- ☐ Usei os comandos `cat program.c` e `git log` para obter a confirmação que me está a pedir: confirmo!

4 Navegação entre Consolidações – *Revert*

Browsing between Commits – Revert

O uso do comando `git reset --hard` **é perigoso**. Na verdade, é um dos poucos comandos no Git que causa alterações que não podem ser revertidas mais tarde. Para se voltar atrás, sem efeitos destruidores, pode-se utilizar o comando `git revert <id do commit>`

Q13.: Em que situações é que se deverá utilizar o comando `git reset --hard`?

- ☐ Sempre que necessário.
- ☐ Só na história de um repositório local. Nunca na história de um repositório partilhado.
- ☐ Para apagar logo que temos a certeza que não queremos que apareça no histórico de versões.

Tarefa 21 *Task 21*

Mais uma vez, simule uma alteração que vai querer desfazer posteriormente. Para isso, abra o ficheiro `program.c` e **apague** a linha que declara a função `main`. Não se esqueça que deve encenar/adicionar o ficheiro e realizar `Commit`. Para este efeito, execute os comandos seguintes (depois de alterar o ficheiro `program.c`):

```
git add program.c
git commit -m "I think I did it again (by Britney Spears)."
```

Tarefa 22 *Task 22*

Execute todos os passos necessários para reverter o seu *commit*. É importante notar que, neste caso, deve indicar o valor da consolidação que quer ver desfeita, e não o valor do *commit* para onde quer voltar (**ao contrário do reset**).

Q14.: Quais foram os comandos que utilizou para efetuar a tarefa anterior?

Q15.: Como evolui a história das consolidações quando se usa `revert`?

- ☐ Neste caso, os *commits* que estão entre o *commit* mais atual e aquele para onde evoluimos são apagados da história e é criado outro *commit* que substitui o atual.
- ☐ Neste caso, os *commits* que estão entre o *commit* mais atual e aquele para onde evoluimos são mantidos na história e é criado outro *commit* novo na história.
- ☐ Neste caso, o ponteiro é ajustado para o *commit* para onde queremos evoluir, e toda a história futura fica intacta. Se consolidarmos, a história é reescrita.

5 Ramificações – Criação e Navegação

Branches – Creation and Navigation

É muito comum a situação em que se quer trabalhar numa funcionalidade ou alteração sem querer mexer no que já se encontra feito. Para tal, o Git permite a criação de ramos de desenvolvimento (*branches*), onde se podem fazer alterações sem afetar os ficheiros de outros ramos. Quando se cria um ramo

com o Git, este cria automaticamente o equivalente a uma cópia da diretoria de trabalho.

Tarefa 23 ***Task 23***

Verifique quantos ramos tem definidos no seu projeto. Execute e análise o *output* do seguinte comando: `git branch`.

Q16.: Quantos ramos estão definidos?
☐ 0 ☐ 1 ☐ 1,5 ☐ 2 ☐ n

Q17.: Como se chama o ramo onde se encontra atualmente (e que é definido por defeito ao inicializar o Git)?
☐ `main` ☐ `develop` ☐ `master` ☐ `slave` ☐ `git server`

Tarefa 24 ***Task 24***

Utilize o comando `git branch teste-funcao` para criar um novo ramo denominado `teste-funcao`. Verifique se a criação do ramo foi bem sucedida (com `git branch`) e evolua para esse ramo utilizando o comando `git checkout teste-funcao`.

Q18.: Quais dos seguintes comandos permitem ver em que ramo se encontra atualmente?
☐ `git set` ☐ `git branch` ☐ `git status`

Dentro do contexto deste ramo experimental pode fazer quaisquer alterações sem prejuízo para o ramo principal. Para explorar as funcionalidades do Git nesta parte, considere as seguintes tarefas/questões.

Tarefa 25 ***Task 25***

Crie um novo ficheiro chamado `funcao.c` com o seguinte conteúdo:

```
#include<stdio.h>

void imprime(){
 printf("Hello World!");
}
```

Altere também o ficheiro `program.c` de forma a que fique semelhante a:

```
#include<stdio.h>

int main(){
 imprime();
 return 0;
}
```

Após estas alterações, deve compilar e executar o seu programa:

```
cc program.c funcao.c -o program.exe
./ program.exe
```

Tarefa 26 ***Task 26***

Se estiver tudo a funcionar corretamente, convém consolidar (*commit*) as alterações para o *branch* atual. Para esse efeito, execute e analise os seguintes comandos (procure perceber todos os detalhes):

```
git status
git add .
git commit -m "printf was moved to a new function"
```

Q19.: Qual é a finalidade do comando `git add .`?
☐ Adicionar/encenar todos os ficheiros da diretoria atual.
☐ Claramente vai adicionar/encenar os ficheiros da diretoria anterior.

Ainda antes de prosseguir, responda e armazene a resposta à questão seguinte: **Q20.: quantos ficheiros tem na diretoria de trabalho atual?**
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4
☐ Infinitus!

6 Fusão e Eliminação de Ramos

Joining and Removing Branches

Volte ao ramo de desenvolvimento principal através da emissão de um comando semelhante a `git checkout master`.

Q21.: O que significa a palavra *checkout* em Português?
☐ Verificar. ☐ Dar uma vista de olhos. ☐ Golo.
☐ É o contrário do *check-in* que se faz no aeroporto.

Q22.: Já verificou quantos ficheiros tem na diretoria de trabalho atual?
☐ Tenho exatamente os mesmos ficheiros que tinha na tarefa anterior.
☐ Que engraçado, tenho mais ficheiros que na tarefa anterior.
☐ Tenho menos ficheiros que na tarefa anterior. Que género de magia é esta?
☐ Como seria de esperar, tenho menos ficheiros que na tarefa anterior.

Procure explicar o que observou:

Tarefa 27 ***Task 27***

Dado ter implementado e testado com sucesso a funcionalidade nova no ramo `teste-funcao`, pode agorar pensar em **fundir** esse ramo com o principal (`master`). Para fundir um ramo com outro, **deve estar posicionado no ramo que vai receber a fusão** e executar um comando semelhante a `git merge <nome ramo que quer fundir para o atual>`.
Construa o comando adequado à fusão e execute-o. Analise o *output*.

Q23.: Qual foi o comando que executou na tarefa anterior?

- ☐ `git merge teste-funcao`
- ☐ `git merge main-cuidado`
- ☐ `git merge slave`
- ☐ `git merge my-precious`

Depois de emitir o comando anterior, verifique novamente quantos ficheiros tem na diretoria atual. Não se esqueça de verificar também o *log*.

Q24.: Quantos ficheiros tem agora no ramo `master`?

- ☐ Tenho o mesmo número de ficheiros que tinha no *branch* `teste-funcao`, funcionou!
- ☐ Fixe, tenho agora exatamente 30 novos ficheiros.
- ☐ Nada funciona... o que me recomenda?

Tarefa 28 ***Task 28***

Dado já não necessitar do ramo que criou para testar a nova funcionalidade/forma de implementar, pode apagá-lo. Emita o comando `git branch -d teste-funcao` e tome as providências que achar necessárias para se certificar que o ramo já não existe.

Q25.: Como pode verificar se o ramo foi devidamente eliminado?

- ☐ `git branch`
- ☐ `git status`
- ☐ Todas as anteriores.
- ☐ Nenhuma das anteriores, exceto a última que diz *Todas as anteriores*... essa não...

Note que o procedimento ilustrado nas tarefas anteriores tem como objetivo dar uma ideia de como deve desenvolver o seu projeto e integrar novas funcionalidades. De uma forma sucinta, pode dizer-se que, para cada funcionalidade que pretenda adicionar, deve:

1. Criar um ramo de desenvolvimento novo e mudar o foco para esse ramo;
2. Desenvolver e testar a nova funcionalidade nesse ramo, mantendo o ramo anterior intato;
3. Mudar o foco para o ramo de desenvolvimento principal, fundir o ramo da nova funcionalidade e apagá-lo.

7 Repositórios e Sincronização

Repositories and Synchronization

Git é um sistema de controlo de versões distribuído. Neste caso, também significa que cada programador guarda uma versão integral da história de consolidações localmente. A existência de repositórios remotos, contudo, irá permitir guardar uma cópia de segurança (*backup*) do projeto e, mais importante, a colaboração com outros programadores. O GitHub pode ser usado como um desses repositórios.

Tarefa 29 ***Task 29***

Dirija-se ao GitHub, entre na plataforma com as suas credenciais e escolha a opção `New`.

Tarefa 30 ***Task 30***

Vai reparar que, após criar o novo repositório, o GitHub lhe faz uma série de sugestões. Por exemplo, é-lhe dito que é uma boa prática que todos os projetos tenham um ficheiro `README.md` e `LICENSE.md`. Sugere-se que colmate estas falhas, selecionando as opções `Add a README file` e `Choose a license`.

Tarefa 31 ***Task 31***

Crie uma nova diretoria (*e.g.*, chamada `teste-github`) e, colocando nela o contexto de trabalho, criar uma cópia local do repositório remoto utilizando um comando semelhante ao seguinte:

```
git clone https://github.com/<your-github-username/<your-repository-name>
```

Deve alterar o comando, primeiro adicionando o seu nome de utilizador(a) no GitHub e, em segundo, o

nome do repositório que acabou de criar. Após executar este comando, serão pedidas as credencias de acesso ao GitHub (*i.e.*, *username* e *password*).

Tarefa 32 ***Task 32***

Adicione o seu `program.c` e `funcao.c` ao seu repositório Git local (`teste-github`). Execute todos os comandos ou tome todas as providências que achar necessárias para concluir esta tarefa com sucesso.

Q26.: Quais foram os comandos que utilizou para adicionar os ficheiros `program.c` **e** `funcao.c`**?**

- ☐ `git config user.email my@email.com`
- ☐ `git config user.name us3rn4me`
- ☐ `git git add .`
- ☐ `git commit -m "My first commit to a remote repository"`
- ☐ Todas as opções anteriores.
- ☐ Só utilizei os comandos descritos na segunda e terceira opção.

Tarefa 33 ***Task 33***

O comando `push` é utilizado para transferir ou enviar um *commit*, que foi realizado no *branch* local, para um repositório remoto (*i.e.*, para o GitHub). Experimente agora enviar a consolidação mais recente do seu projeto para o repositório remoto emitindo um comando semelhante a:

```
git push origin master
```

Deverá conseguir verificar, através de num navegador de Internet, que os novos ficheiros foram agora copiados para o repositório remoto.

8 Colaboração em Projetos

Collaboration in Projects

Da execução das tarefas na secção anterior deve ter ficado com uma ideia de como se configura um repositório remoto e de como se enviam os ficheiros para esse repositório. Os comandos que permitem a colaboração de vários programadores no mesmo projeto foram já quase todos abordados, faltando apenas parte daqueles que permitem obter o projeto do repositório remoto, nomeadamente o `git fetch` e o `git pull`. Em baixo apenas se aborda o comando `git pull`, mas fica a indicação de que o comando

```
git clone https://github.com/<username>/
<repositorio>.git
```

também faz parte deste conjunto, e deve ser usado para obter o conteúdo **e simultaneamente inicializar** um projeto, incluindo a sua história de versões, ramos e consolidações Git. Este é o **comando usado para inicializar um projeto na sua máquina local quando este ainda não existe.**

Tarefa 34 ***Task 34***

Quando se usa o Git para trabalho colaborativo é necessário ganhar um conjunto de boas práticas. É preciso pensar sempre que outros programadores podem estar a trabalhar no projeto e a publicar as alterações a qualquer momento. **Q27.: Em que momentos considera ser indicado verificar se há alterações no repositório remoto?**

- ☐ Antes de fazer uma consolidação *commit*.
- ☐ Sempre que fizer uma alteração.
- ☐ Quando se prepara para começar a trabalhar no projeto.
- ☐ Mesmo antes submeter alterações para o repositório remoto.

Finalize o guia, experimentando o comando seguinte, só para ficar com uma ideia de como funciona:

```
git pull origin master
```