Engenharia de Software – Ficha prática 1

Sistemas de Controlo de Versões: Git e GitHub

Objetivos

- Conhecer conceitos gerais de Sistemas de Controlo de Versões
- Instalar e configurar o Git
- Conhecer a arquitetura do Git e a sua funcionalidade básica
- Criar conta no GitHub
- Conhecer funcionalidade básica do GitHub

Parte I – Instalação e Configuração básica do Git

1. Descarregar de https://git-scm.com/downloads a versão apropriada para o seu sistema operativo
2. Instalar, mantendo todas as opções por omissão, exceto o editor por omissão (recomenda-se escolher o Visual Studio Code). Pode consultar a sequência de ecrãs para um sistema Windows 10 64-bit no Anexo I.

Nota: Embora muitas das funcionalidades do Git estejam integradas em IDEs e editores de código modernos como o IntelliJ IDEA e o Visual Studio Code, nesta ficha será utilizada a predominantemente a linha de comandos. Em Windows, devido as limitações e peculiaridades das linhas de comando disponíveis (cmd e PowerShell), recomenda-se a utilização da linha de comando Git Bash incluída na instalação git-scm.

3. Confirme a instalação abrindo uma linha de comando e executando o comando:

```
$ git --version
```

Deverá resultar em algo semelhante a:

```
git version 2.35.1
```

4. Para completar a configuração, adicione a sua identificação no Git executando os seguintes comandos:

```
$ git config --global user.name "João Silva"
$ git config --global user.email “joao-silva@exemplo.dev”
```

Sem este passo, a funcionalidade será reduzida.
Em alternativa, pode editar o ficheiro de configuração que inclui estas (e outras variáveis) usando o comando para invocar o editor definido na instalação:

```
$ git config --global -e
```

Adicionalmente, pode configurar os seus IDEs (ex.: IntelliJ IDEA) ou editores de código (ex.: Visual Studio Code) para utilizar a Git Bash como linha de comando..

Parte II – Cenário "começar um projeto novo com Git"

1. No IntelliJ um novo Projeto Maven chamado "GitTest".
2. Abra uma linha de comando Git Bash na pasta do projeto e execute o seguinte comando:

```
$ git init
```

Deverá resultar algo similar a:

```
Initialized empty Git repository in /<path>/<to</<project>/.git/
```

3. Confirme que dentro da pasta de projeto foi criado uma pasta `.git`.

Nota: Usualmente esta pasta é considerada "escondida" pelo sistema operativo, pelo que muitas aplicações não a mostram. Em Git Bash basta usar o comando:

```
$ ls -a
```

4. Execute o seguinte comando para verificar o estado do repositório Git criado:

```
$ git status
```

Deverá resultar algo similar a:

```
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .idea/
        GitTest.iml
        pom.xml

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

A secção "No commits yet" indica que o repositório está vazio, sem qualquer estado do projeto armazenado.
A secção "Untracked files" indica que são ficheiros ou pastas que o Git detetou, mas que ainda não sabe o quê fazer com eles.
Dos "Untracked files" apenas o ficheiro `pom.xml` é verdadeiramente fundamental ao projeto. Os restantes são ficheiros auxiliares úteis para o seu ambiente de trabalho pessoal, mas inúteis para um programador quem utilize um ambiente de trabalho diferente.

5. Dentro da pasta de projeto crie um ficheiro com nome `.gitignore` (tenha em atenção a nota do passo 3). Edite o conteúdo deste ficheiro para que o Git ignore pastas e ficheiros não fundamentais para o seu projeto. Nomeadamente, a pasta `.idea` e os ficheiros `*.iml`:

```
.idea/
*.iml
```

Nota: Responda "Cancel" sempre que lhe aparecer uma janela de diálogo "Add Files to Git" do IntelliJ IDEA. Procederemos assim apenas durante esta ficha para poder controlar o Git apenas pela linha de comando.

Sugestões:
O site https://github.com/github/gitignore contém alguns modelos apropriados para certos tipos de projeto.
O site https://gitignore.io permite construir um ficheiro `.gitignore` apropriado para um determinado projeto com base no tipo de projeto e as ferramentas de desenvolvimento utilizadas (no nosso caso: Java, Maven, IntelliJ, Visual Studio Code, etc.).

6. Repita o comando do passo 4. Desta vez, apenas os ficheiros `pom.xml` e `.gitignore` devem aparecer na lista de "untracked files".

7. Coloque todos ficheiros assinalados como "untracked" no "stage" (também conhecido por "index") do Git usando o comando:

```
$ git add .
```

Nota: Se lhe aparecer o aviso "warning: LF will be replaced by CRLF" pode ignorá-lo.

8. Repita o comando `git status` e verifique que `pom.xml` e `.gitignore` passaram a estar na secção "Changes to be commited" (e assinalados a verde). Isto é, foram transferidos para o "stage" (ou "index").

```
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   pom.xml
```

Também pode utilizar o seguinte comando para verificar apenas o conteúdo do "stage":
git

```
$ git ls-files
```

9. Execute o seguinte comando para criar a primeira revisão do projeto armazenada no repositório Git:

```
$ git commit -m "commit inicial"
```

Deverá resultar algo similar a:

```
[main (root-commit) 8a15440] commit inicial
 2 files changed, 108 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 pom.xml
```

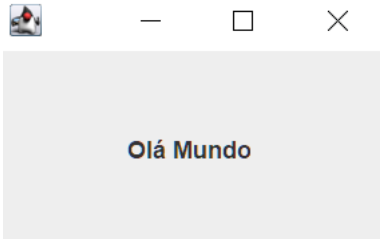
10. Adicione a este projeto uma "GUI Form" (clicar com o botão direito do rato em "java", em src/main e escolher New->Swing UI Designer->GUI Form) com o nome `HelloWorld` contendo uma `JLabel` de nome `lblMensagem` a conter o texto "Olá Mundo" e centrada no painel. O código para essa classe deve ficar como o seguinte:

```
public class HellowWorld extends JFrame {
    private JPanel panel1;

    public HellowWorld(){
        setContentPane(panel1);
        pack();
    }

    public static void main(String[] args) {
        new HellowWorld().setVisible(true);
    }
}
```

11. Execute a aplicação e confirme que resulta em algo similar a:



12. Execute o comando `git status` e verifique que `src/` está em "Untracked Files".
Nota: Se `target/` também estiver em "Untracked Files" é sinal que o ficheiro `.gitignore` que criou no passo 5 não é suficientemente abrangente, pois esta pasta também não precisa de ser guardada no repositório Git. Neste caso, deverá corrigir o ficheiro `.gitignore` antes de continuar.

13. Execute os seguintes comandos para (a) colocar os novos ficheiros no "Stage" e (b) criar uma nova revisão no repositório:

```
git add .
git commit -m "adicionar janela HelloWorld"
```

14. Consulte o histórico das revisões com o comando:

```
$ git log
```

Repare nos detalhes que cada revisão apresenta, nomeadamente: o identificador hexadecimal da revisão, o autor, a data, e a descrição.

15. Consulte o histórico das revisões com o comando:

```
$ git log --oneline
```

Compare a saída deste comando com a do anterior. Repare que o identificador da revisão apenas mostra os primeiros 7 dígitos e não os 40 totais. Aponte os 7 dígitos que identificam a revisão inicial.

16. Execute o seguinte comando para "retroceder no tempo" e repor o seu projeto como estava quando foi feita a revisão inicial. Tenha em atenção que `1234567` deverá ser substituído pelo identificador que apontou no passo anterior.

```
$ git checkout 1234567
```

Verifique que os ficheiros `HelloWorld.java` e `HelloWorld.form` desapareceram (não existem na revisão inicial). Repare que a pasta `src/main/java` também foi removida.
Nota: Tenha em atenção que este comando irá falhar se houver alterações que o Git considere que devem ser gravadas, pois ao saltar de revisão as alterações não gravadas são perdidas. Se isto ocorrer, bastará fazer um novo `add` seguido de `commit`, tal como indica no passo 13.

17. Crie a pasta `src/main/java` e depois a classe `Vazia.java` que contem apenas o seguinte código:

```
public class Vazia {
}
```

18. Execute o seguinte comando para "voltar ao presente", isto é, voltar à última revisão registada no repositório:

```
$ git switch -
```

Verifique que os ficheiros `HelloWorld.java` e `HelloWorld.form` reapareceram e que o ficheiro `Vazia.java` permanece inalterado (continua como "untracked").

19. Antes de continuar, investigue e experimente os seguintes comandos:

```
$ git add FICHEIRO1
$ git reset
$ git rm --cached FICHEIRO2
```

Onde FICHEIRO1 será o caminho para um ficheiro "untracked" (ficheiro novo) ou um ficheiro "tracked" mas com modificações; e FICHEIRO2 o caminho para um ficheiro atualmente no "stage" (aparece na secção "Changes to be commited" quando se executa o comando `git status`).
Estas comandos, em combinação com o comando `git add` anteriormente mencionado, são úteis para manipular o conteúdo do "stage".

Sugestão: Todos os comandos Git incluem uma opção `--help` que apresenta uma página com documentação detalhada sobre o comando:

```
$ git add --help
```

A opção `-h` apresenta uma informação mais abreviada (apenas inclui a sintaxe, e a listagem das opções:

```
$ git add -h
```

20. Crie uma nova janela chamada `Experimental`, em tudo idêntica a `HelloWorld` mas com o texto "Experiência" na sua `lblMensagem`. Verifique que estes dois novos ficheiros são reconhecidos como "untracked files".

Sugestão: Crie os ficheiros copiando os ficheiros `.java` e `.form` de `HelloWorld`. Edite a propriedade "bind to class" de `Experimental.form` para referir `Experimental` em vez de `HelloWorld`.

21. Crie um novo "branch" executando o seguinte comando:

```
$ git branch experimental
```

22. Mude para o novo "branch" executando o seguinte comando:

```
$ git switch experimental
```

23. Adicione os ficheiros criados no passo anterior ao "stage" e crie uma nova revisão:

```
$ git add .
$ git commit -m "adicionar janela Experimental"
```

24. Regresse ao “branch” principal “main” executando o seguinte comando:

```
$ git switch main
```

Verifique que os ficheiros `src/java/Experimental.*` desapareceram.

Parte III – Cenário “transferir um projeto para um repositório remoto”

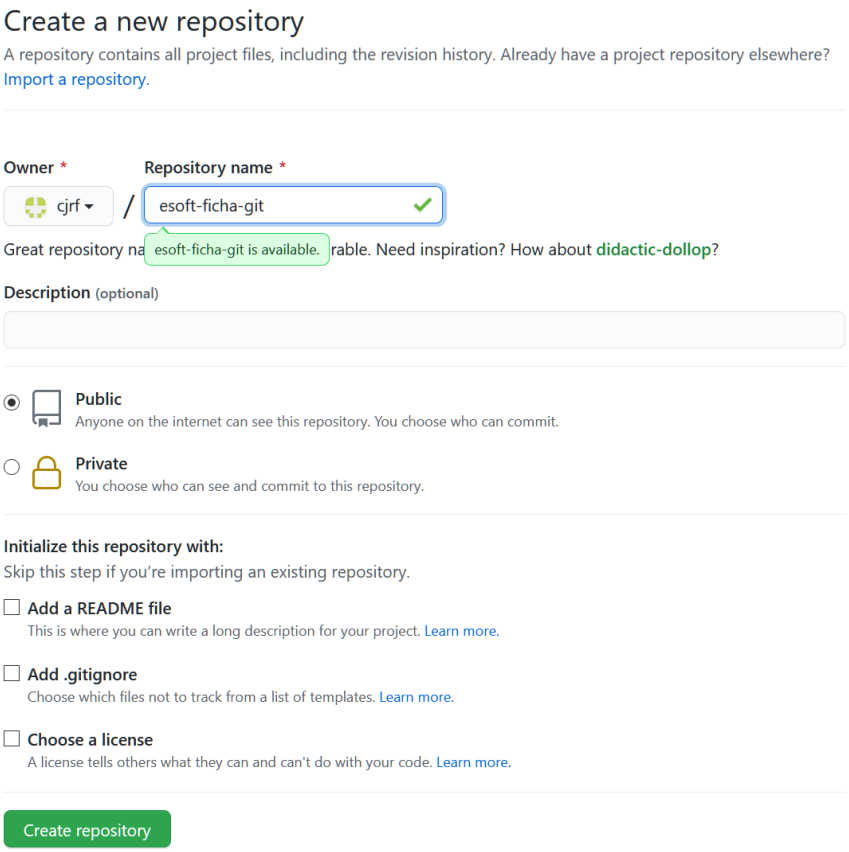
Este cenário é fundamental para permitir a colaboração entre pessoas num mesmo projeto (ex.: realizar um trabalho de grupo), mas também é extremamente útil para projetos com um único autor, pois permite manter uma cópia de segurança do projeto *off-site*, isto é, uma cópia armazenada fisicamente fora do local de trabalho do autor; ou simplesmente divulgar o seu código fonte.

O Git oferece grande flexibilidade para o local onde um repositório remoto pode ser armazenado. Pode ser usado qualquer servidor acessível por HTTPS ou SSH ou até mesmo uma pasta local. Mas geralmente é preferível usar serviços especializados no alojamento de repositórios Git. O GitHub (https://github.com) é o mais popular e aquele que iremos usar nesta UC, mas existem outros. Por exemplo: GitLab (https://gitlab.com), BitBucket (https://bitbucket.org), e SourceForge (https://sourceforge.net). Estes serviços especializados, para além do alojamento do repositório, fornecem múltiplas ferramentas de colaboração e comunicação que ajudam na promoção, suporte e gestão de projetos de software

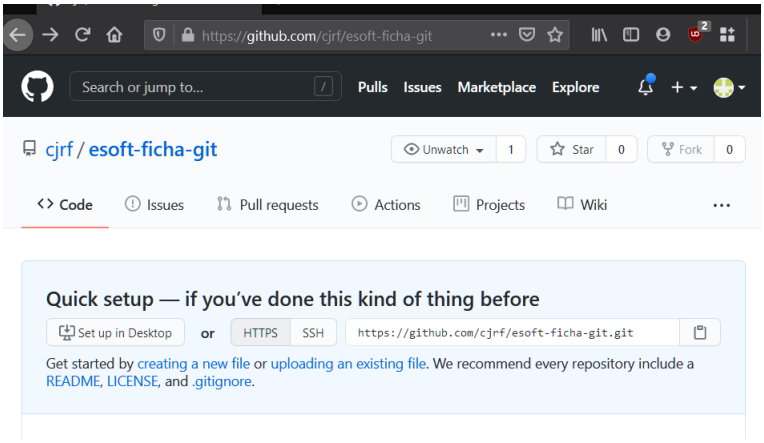
1. Se ainda não o fez, crie uma conta no GitHub (https://github.com/join). Tenha em atenção que o seu *username* será informação pública e que posteriormente poderá adicionar endereços e-mail adicionais.

2. No GitHub, crie um novo repositório **privado** com o nome `esoft-ficha-git`.

Nota: certifique-se que todas as checkboxes na secção "Intialize this repository with:" estão desligadas, para que seja criado um repositório completamente vazio:



Se esta operação for bem sucedida, deverá ver a página Web do seu repositório no GitHub:



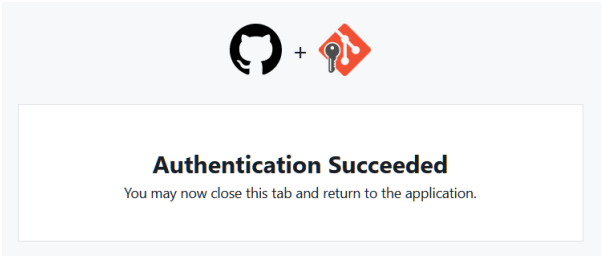
3. Abra uma Git Bash na pasta do seu projeto ou execute o seguinte comando para entrar nessa pasta a partir de uma Git Bash já aberta:

```
cd ~/IdeaProjects/GitTest
```

4. Copie os comandos da secção "…or push an existing repository from the command line" da página Web do seu repositório no GitHub e execute-os na Git Bash aberta no ponto anterior.
   Nota: Em Windows, a combinação de teclas Ctrl-V não serve para "colar" na Git Bash. Em alternativa, utilize o comando "colar" do menu popup (botão secundário do rato) ou clique com o terceiro botão de rato (botão da roda de rato).

5. A primeira vez que executar estes comandos deverá autenticar o seu comando Git com as credencias do GitHub:

   Recomenda-se utilizar a opção "Sign in with your browser". Se necessário, ser-lhe-á pedido que se autentique no GitHub usando o seu browser e, depois disso, o comando Git será autorizado automaticamente:



6. Verifique que na página Web do seu repositório no GitHub se encontra uma cópia do seu projeto.

Parte IV – Cenário "colaborar com outras pessoas usando repositórios remotos"

Uma vez alojado um projeto num repositório remoto (na nuvem), é possível convidar outras pessoas a colaborar nesse projeto. Para isso é necessário atribuir permissões de acesso a esses colaboradores e utilizar os mecanismos de sincronização disponibilizados pelo Git.

1. Convide um colega e o docente da UC para colaborar no recém criado repositório no GitHub. Para tal, na página web do seu repositório no GitHub, clique em "Settings", depois em "Manage access" e depois "Invite a collaborator". Procure o *username* do convidado e clique em "Add XXXX to this repository", onde XXXX será o *username* do convidado.

2. Aceite o convite que um colega lhe terá endereçado para colaborar noutro repositório.

Nota: O convite deverá chegar por e-mail, mas em caso de demora poderá visitar o endereço `https://github.com/XXXX/esoft-ficha-git/invitations`, onde XXXX será o *username* do colega que o convidou.

3. Uma vez recebido e aceite o convite, abra uma Git Bash na pasta `IdeaProjects` (`~/IdeaProjects`) e execute o seguinte comando, substituindo XXXX pelo *username* de quem o convidou:

```
$ git clone https://github.com/XXXX/esoft-ficha-git.git
```

Nota: O comando `git clone` apenas serve para a primeira sincronização do projeto (quando começamos a colaborar).

4. Confirme que:
   (a) a pasta `esoft-ficha-git` contém um repositório Git
   (b) esse repositório está associado ao repositório remoto de quem o convidou
   (c) esse repositório apenas tem "commits" feitos por quem o convidou
   Para tal, ainda na Git Bash, execute os seguintes comandos:

```
$ cd esoft-ficha-git
$ git remote -v
$ git log
```

5. Abra este projeto no IntelliJ e edite a janela `HelloWorld` adicionando programaticamente (editar apenas `HelloWorld.java`) uma `JLabel lblMensagem2` com o texto "O João Silva esteve aqui" (naturalmente usando o seu nome).

6. Verifique que o ficheiro `HelloWorld.java` está como "changes not staged for commit."

7. Adicione todas as mudanças "not staged" ao "stage" e faça "commit" das alterações que fez:

```
$ git commit -a -m "adicionar o meu nome"
```

8. Execute este comando para efetivar as alterações que fez anteriormente no repositório remoto:

```
$ git push origin main
```

9. **Antes de avançar, deverá confirmar que o colega que o convidou anteriormente já chegou a este passo.**

10. No IntelliJ, mude para o seu projeto `GitTeste`. Repita neste projeto as mesmas alterações que fez ao projeto do seu colega (passos 11 a 14).

Repare que o `push` falha e que lhe é pedido que faça um `pull` primeiro.

11. Execute o seguinte comando para transferir as alterações do repositório remoto para o repositório local:

```
$ git pull origin main
```

Repare no aparecimento da palavra "CONFLICT" no resultado deste comando.

12. Resolva o conflito editando o ficheiro `HelloWorld.java`. As linhas começadas por "<<<<<<<", "=======" e ">>>>>>>" são adicionadas pelo Git como forma de indicar as diferenças entre as duas versões do ficheiro em conflito e terão de ser removidas como parte do processo de "merge" (combinar as alterações existentes nos repositórios local e remoto).

Anexo I – Assistente de Instalação do Git

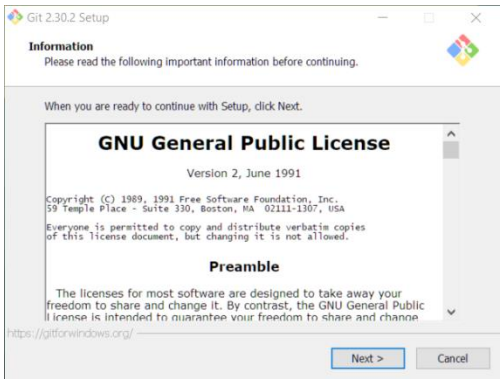


Figura. Licença (Passo 1 de 15)

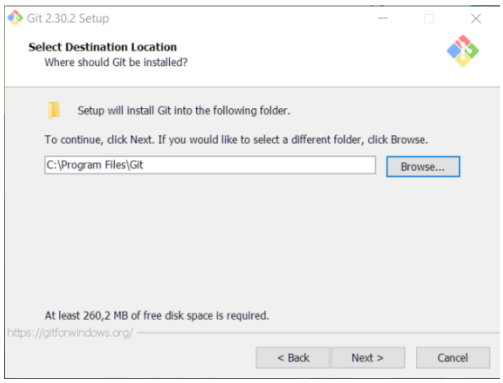


Figura 1. Pasta de instalação (Passo 2 de 15)

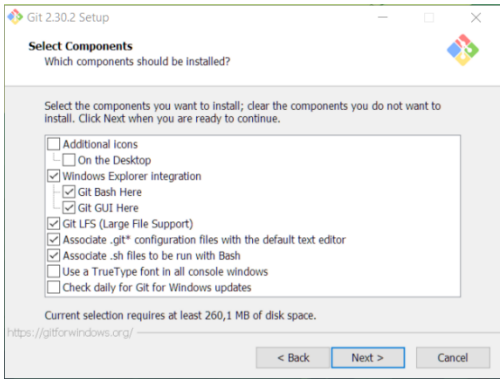


Figura 2. Componentes a instalar (Passo 3 de 15)

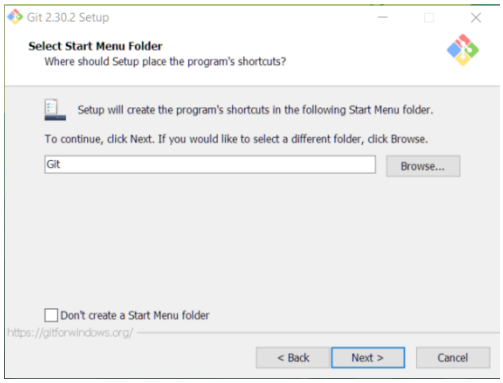


Figura 3. Pasta do Menu de Arranque (passo 4 de 15)

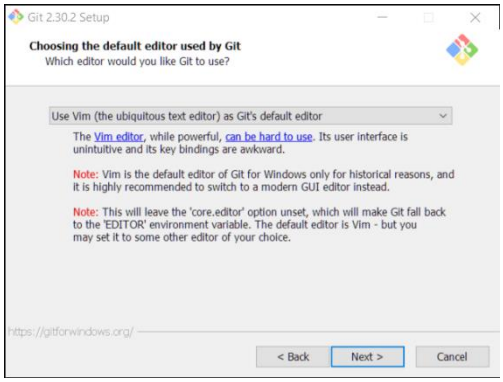


Figura 4. Editor por omissão (passo 5 de 15)

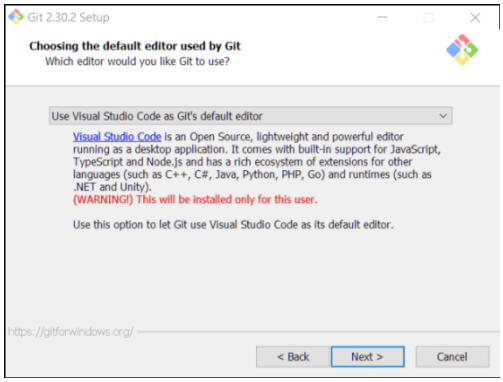


Figura 5. Editor por omissão alterado para Visual Studio Code

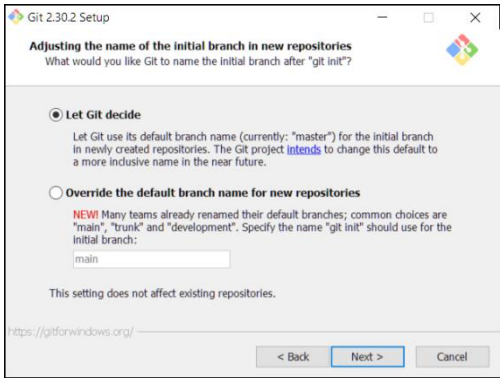


Figura 6. Nome do "branch" por omissão (passo 6 de 15)

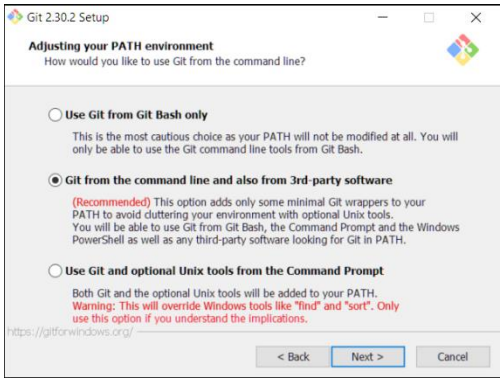


Figura 7. Variável PATH (passo 7 de 15)

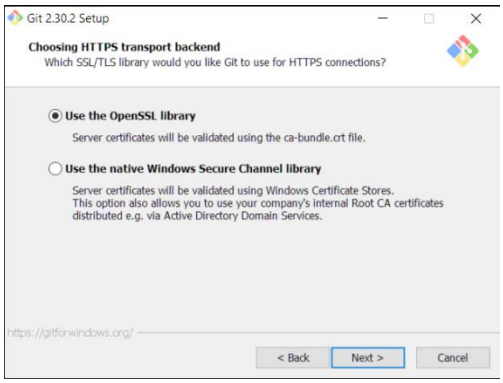


Figura 8. Biblioteca para ligação (passo 8 de 15)

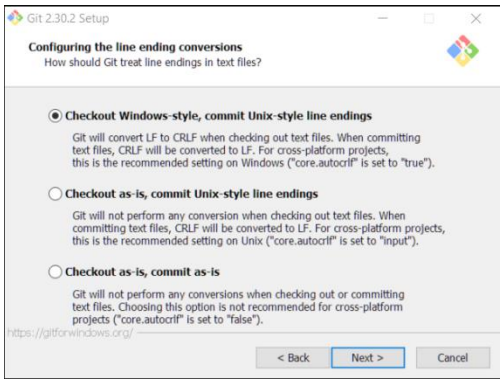


Figura 9. Conversão do fim-de-linha (passo 9 de 15)

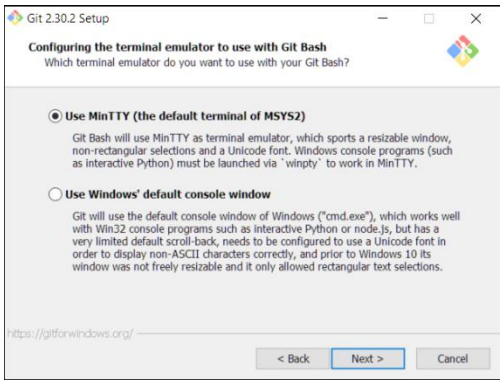


Figura 10. Emulador de Terminal para linha de comando (passo 10 de 15)

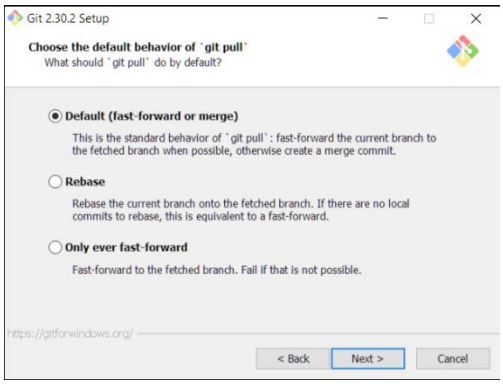


Figura 11. Comportamento do comando "git pull" (passo 11 de 15)

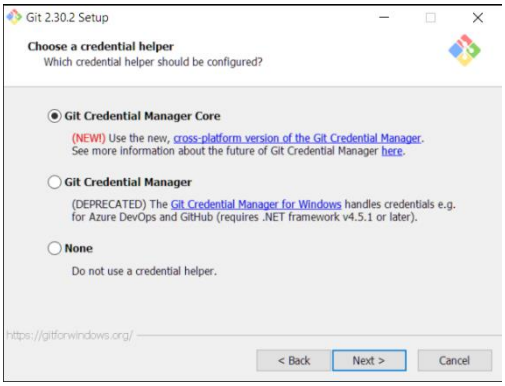


Figura 12. Gestor de credenciais (passo 12 de 15)

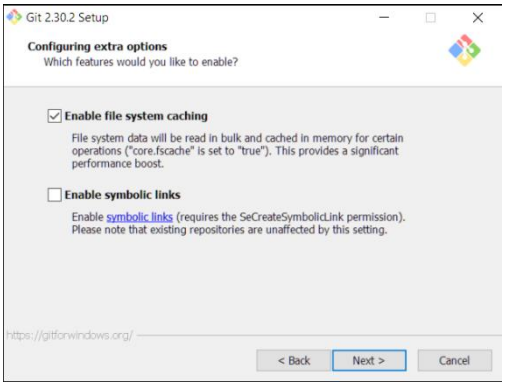


Figura 13. Opções extra (passo 13 de 15)

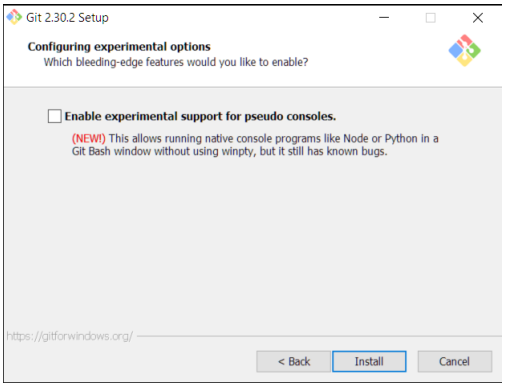


Figura 14. Opções experimentais (passo 14 de 15)

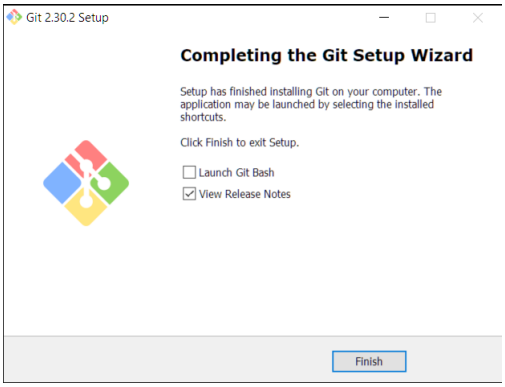


Figura 15. Pós-instalação (passo 15 de 15)