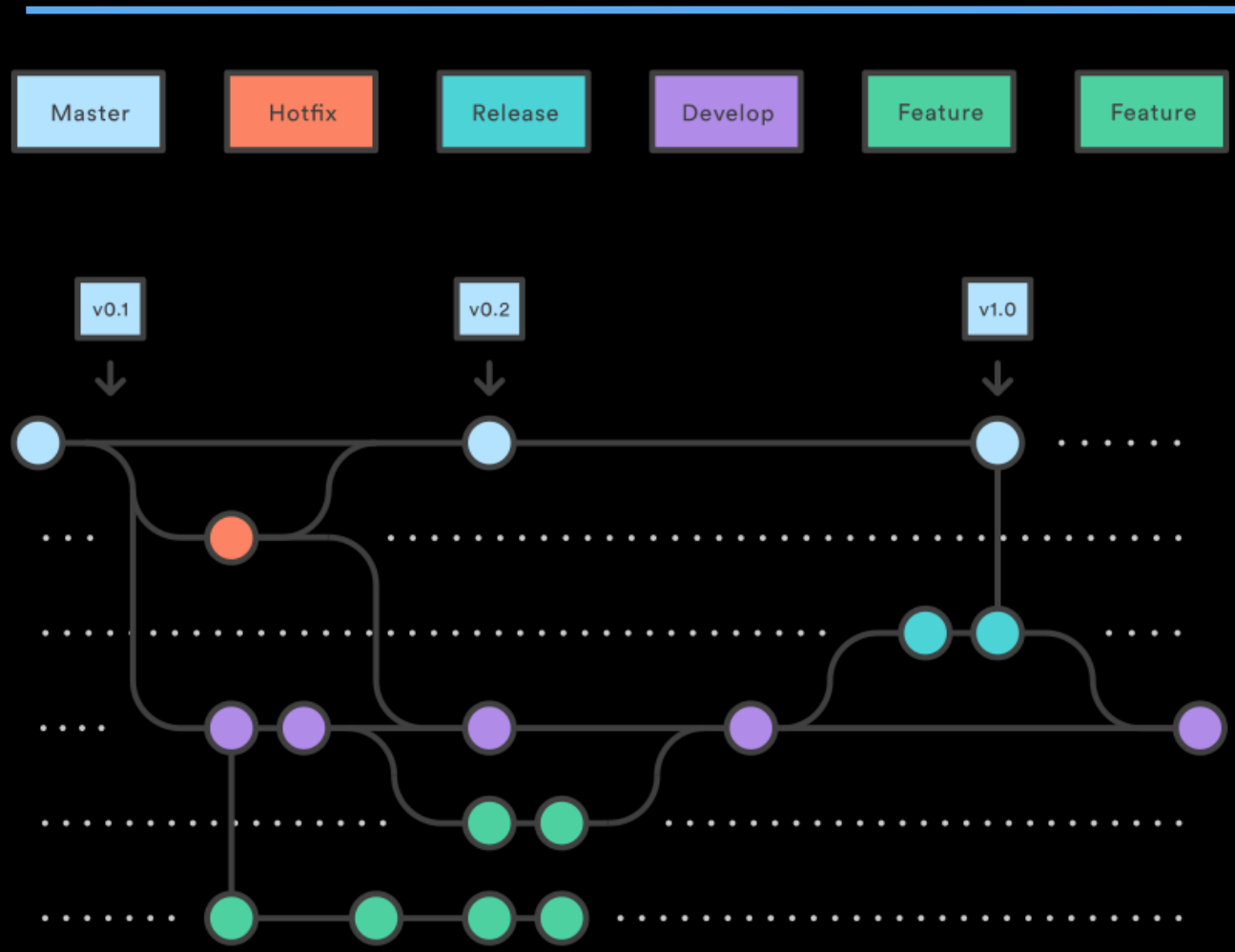
SISTEMAS DE CONTROLO DE VERSÃO



José Ferreira

Miguel Carvalho

CONTROLO DE VERSÃO

ÍNDICE

CONTROLO DE VERSÃO

O QUE É?

- É uma classe de sistemas que gerem as alterações efetuadas a ficheiros, sejam eles documentos, programação, imagens, etc.

- Estas alterações são guardadas em conjuntos chamados versões e permitem que estas sejam guardadas ou copiadas e controlar o seu acesso.

- Os sistemas também permitem ramificar os fluxos de trabalho, comunicar problemas no código, restaurar versões anteriores e fazer experiencias sem perder o acesso a versões estáveis.

CONTROLO DE VERSÃO

O QUE É?

- A necessidade desta gestão existe á centenas de anos mas tornou se muito importante na era da informática.
- A Engenharia de Software exige sistemas de controlo de versão bastante complexos.

CONTROLO DE VERSÃO

BENEFÍCIOS

VERSÕES

- Cada versão tem uma descrição das modificações efetuadas e o que elas fazem, tal como reparar "bugs" ou adicionar funcionalidades.
- Estas descrições ajudam a equipa a perceber as alterações no código sem terem de analisar linhas ou ficheiros individualmente.
- Também permitem ver ou restaurar versões anteriores a qualquer altura.

CONTROLO DE VERSÃO

BENEFÍCIOS

PROGRAMAÇÃO EM EQUIPA

- Os sistemas sincronizam as versões e garantem que alterações de um utilizador não entram em conflito com as de outro, mesmo que estas sejam criadas ao mesmo tempo.
- Também permitem equipas com outras funções, como, por exemplo, a equipa de testes, estar a trabalhar com uma versão enquanto uma versão nova já está em desenvolvimento.

CONTROLO DE VERSÃO

BENEFÍCIOS

HISTÓRICO

- Os sistemas mantêm um histórico das alterações quando os utilizadores criam novas versões dos ficheiros.
- A equipa pode saber quem, quando, e porque foram feitas as alterações e que alterações são essas.
- Também facilita a retração para uma versão anterior que funcione melhor, permitindo que a equipa possa experimentar sem medo de não poder retroceder.

CONTROLO DE VERSÃO

BENEFÍCIOS

AUTOMATIZAÇÃO DE TAREFAS

- Manualmente o controlo de versão seria dispendioso para as equipas. Automatizando o controlo de versão, os sistemas facilitam a análise, teste e implementação dos projetos entre outras tarefas.

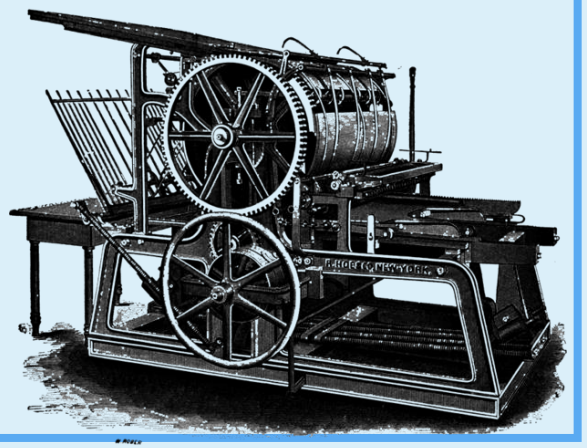
FLUXOS DE TRABALHO

- Os sistemas garantem que as equipas tem os seus processos de desenvolvimento organizados com ferramentas compatíveis e permissões de acesso que controlam quem pode alterar os ficheiros.

CONTROLO DE VERSÃO

BREVE HISTORIA

- A necessidade de organizar e controlar as várias fases do desenvolvimento sempre existiu, mas veio a agravar desde o início da era informática.
- A numeração das edições dos livros é um exemplo do controlo de versões anterior à era informática.
- Os sistemas de controlo de versão utilizados são atualmente desenvolvidos em software e permitem vários membros de uma equipa efetuar alterações.



CONTROLO DE VERSÃO

BREVE HISTORIA

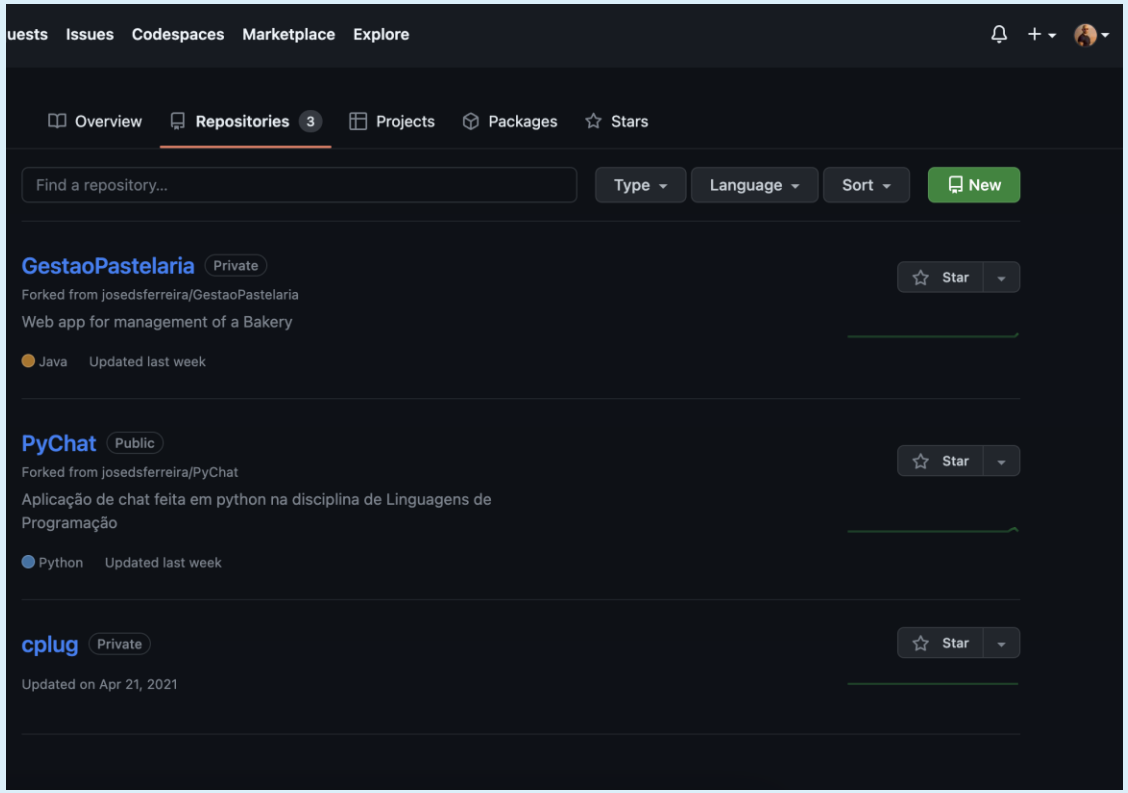
- O controlo de versão já era aplicado nos negócios e na lei. De facto, a definição de contratos é um dos exemplos das primeiras utilizações do controlo de versão.
- As técnicas mais sofisticadas começaram a ser utilizadas para o seguimento eletrónico das alterações aos ficheiros CAD, substituindo a implementação eletrónica "manual" do controlo de versão tradicional.

- Os primeiros softwares de controlo de versão começaram a surgir nos anos 60 e 70 para o sistema operativo OS/360 da IBM.

CONTROLO DE VERSÃO

REPOSITÓRIO

- Um repositório é o local onde são armazenados os dados, incluindo as diferentes versões, todas as alterações efetuadas, entre outros dados.



CONTROLO DE VERSÃO

ESTRUTURA

- As alterações dos dados ao longo do tempo são geridas através de controlo de versão, e estas alterações podem ser organizadas de várias formas.

- Git, como iremos ver mais a frente tem uma maneira diferente de guardar a informação.

- É necessário verificar ou submeter dados modificados ao sistema de controlo de versão.

CONTROLO DE VERSÃO

ESTRUTURA DE GRAFOS

- As versões ocorrem em sequência ao longo do tempo e podem ser organizadas por ordem, quer por número de versão, quer por registo de data e hora.

- Quando os dados que estão sob controlo de versão são modificados, devem ser registados ou confirmados para refletir as alterações no sistema de controlo de versões.

- Se várias pessoas estiverem a trabalhar num único conjunto de dados ou documento, estão implicitamente a criar ramificações dos dados (nas suas cópias de trabalho), pelo que surgem questões de integração.

CONTROLO DE VERSÃO

BRANCHES (RAMIFICAÇÃO)

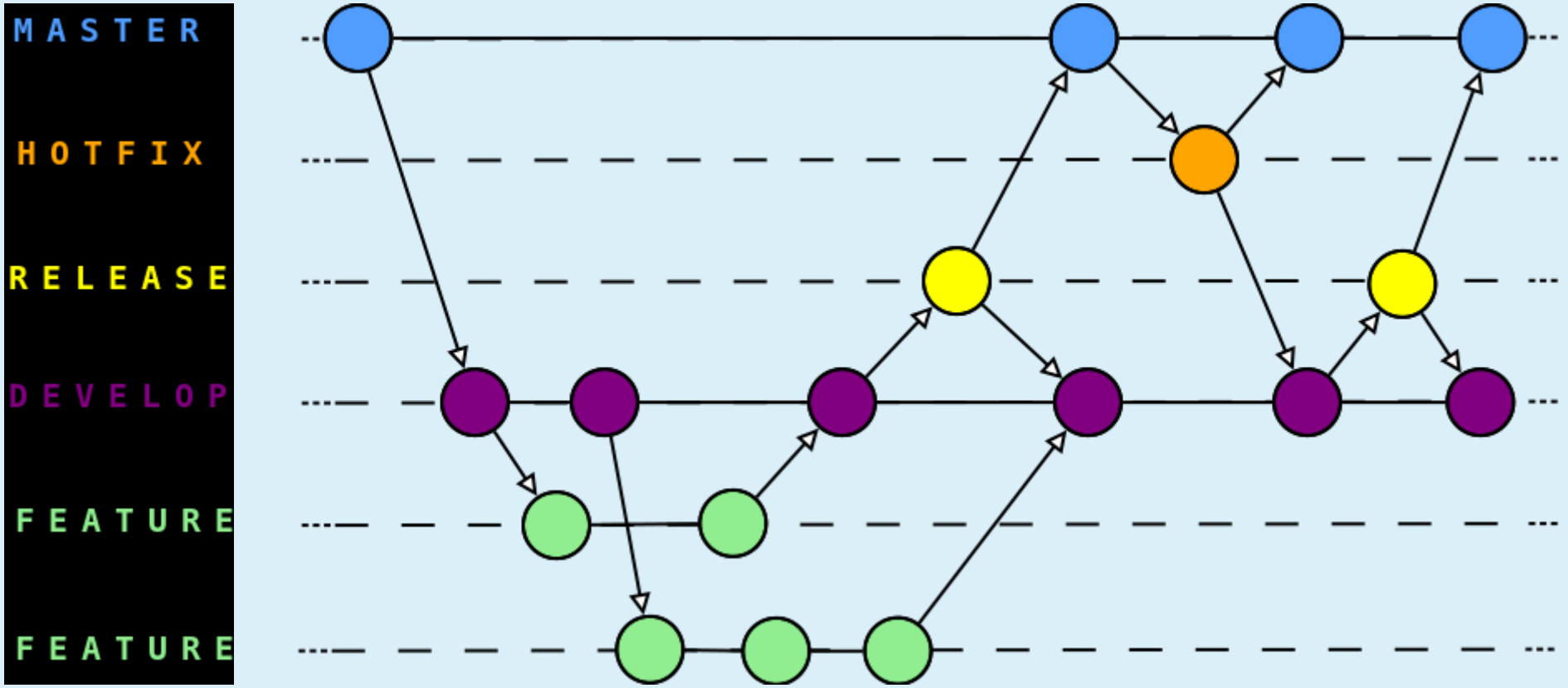
- Branches correspondem a copias do código oficial do projeto, cada branch representa uma modificação diferente ao código-fonte, sem afetar o código oficial.

- São utilizadas para que vários membros de uma equipa façam modificações sem interferir no trabalho uma das outras.

- Depois das alterações serem revistas e aceites, podem ser integradas no projeto oficial.
- O ato de criar um branch chama-se fork.
- O merge é a integração entre dois branches ou mais.

CONTROLO DE VERSÃO

BRANCHES (RAMIFICAÇÃO)



CONTROLO DE VERSÃO

TIPOS

- Existem 3 tipos de sistemas, sendo a grande diferença entre eles o local onde os ficheiros são guardados:

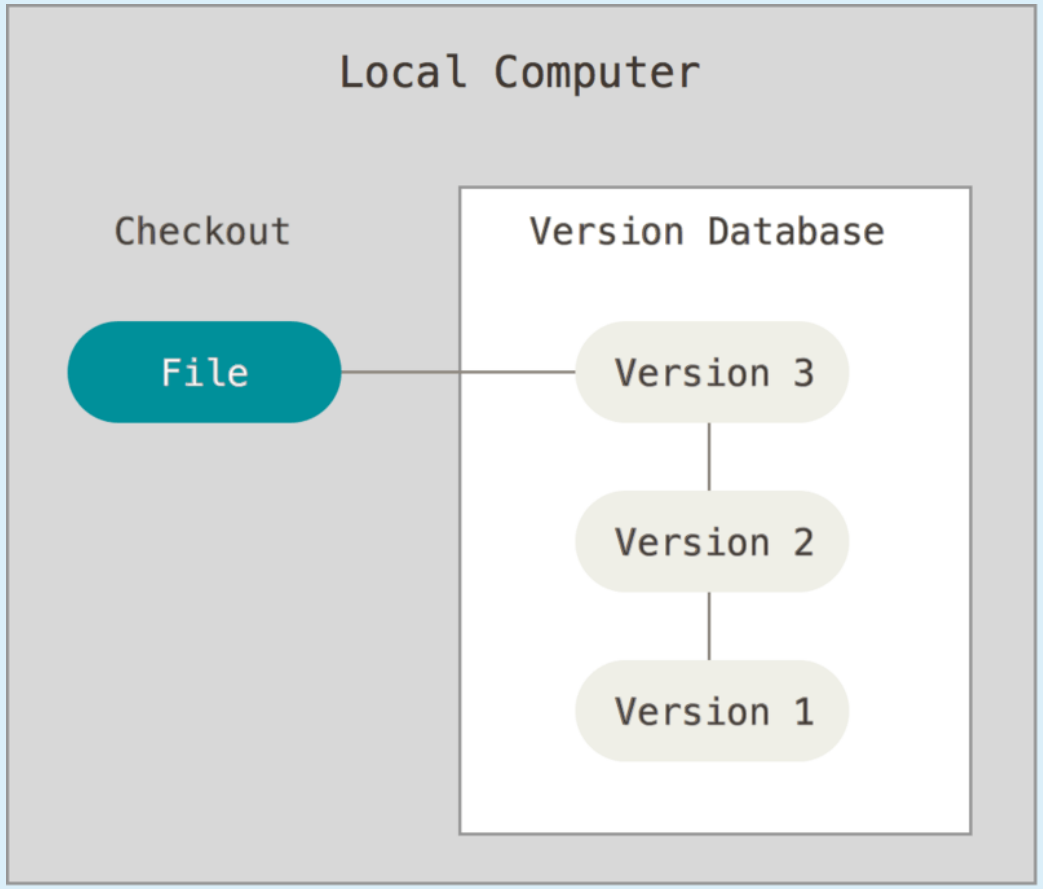
SISTEMAS LOCAIS

SISTEMAS CENTRALIZADOS

SISTEMAS DISTRIBUIDOS

CONTROLO DE VERSÃO

TIPOS – SISTEMAS LOCAIS



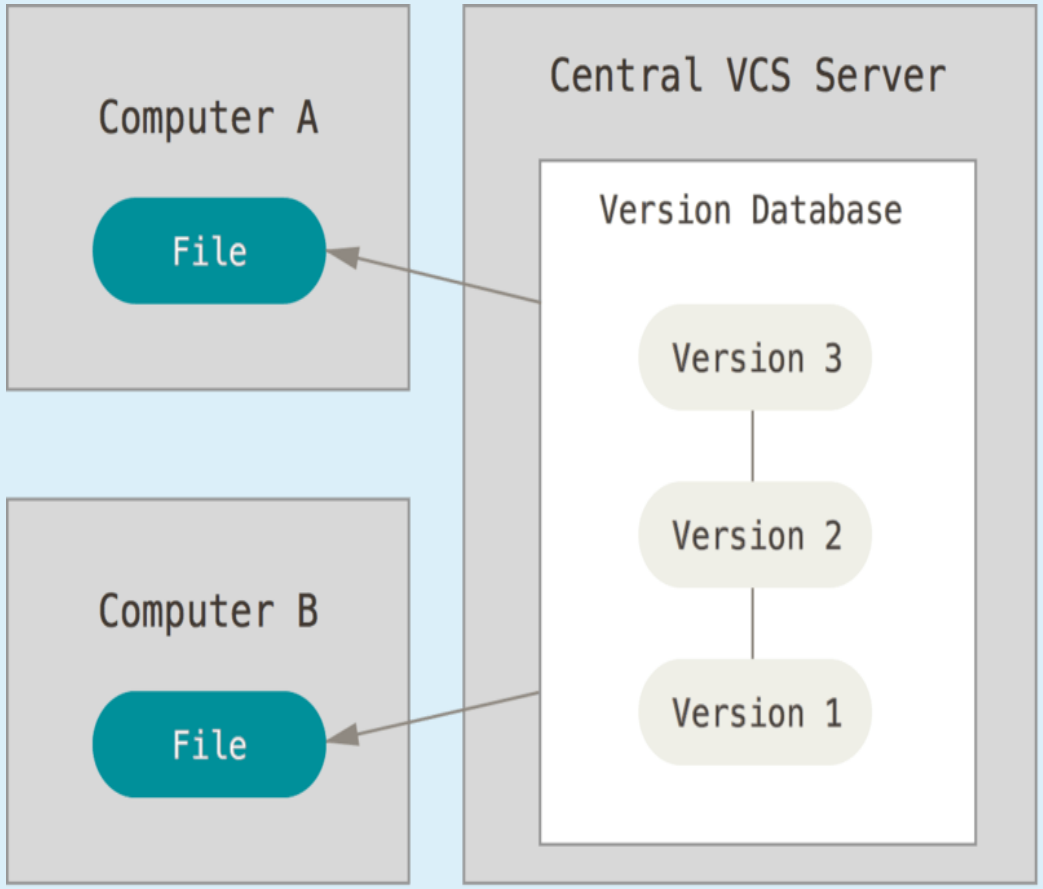
CONTROLO DE VERSÃO

TIPOS – SISTEMAS LOCAIS

- É um tipo utilizado por muitas pessoas devido á sua simplicidade, mas é também o mais vulnerável a erros.
- Consiste apenas copiar os ficheiros para outra pasta. É muito fácil esquecer em que pasta o utilizador está e substituir o ficheiro errado ou copiar por cima de ficheiros que não se pretendia.
- Para resolver estes problemas criou se um sistema de controlo de versão com uma base de dados simples que guarda todas as alterações efetuadas aos ficheiros sob controlo.
- Um dos sistemas mais populares deste tipo, chamado RCS, funciona guardando conjuntos de alterações e permite obter uma versão específica adicionando os conjuntos

CONTROLO DE VERSÃO

TIPOS – SISTEMAS CENTRALIZADOS



CONTROLO DE VERSÃO

TIPOS – SISTEMAS CENTRALIZADOS

- Os sistemas centralizados surgiram para lidar com o problema de projetos desenvolvidos por vários elementos.

- Estes sistemas usam um único servidor para conter todos os ficheiros integrados em versão. Cada membro da equipa pode copiar (check out) ficheiros do servidor para o seu computador para fazer alterações. Durante muitos anos este era o standard de controlo de versão.

- Este tipo oferece várias vantagens, especialmente sobre os sistemas locais. Por exemplo, permitem controlar o acesso aos ficheiros e melhoram a comunicação entre elementos de uma equipa.

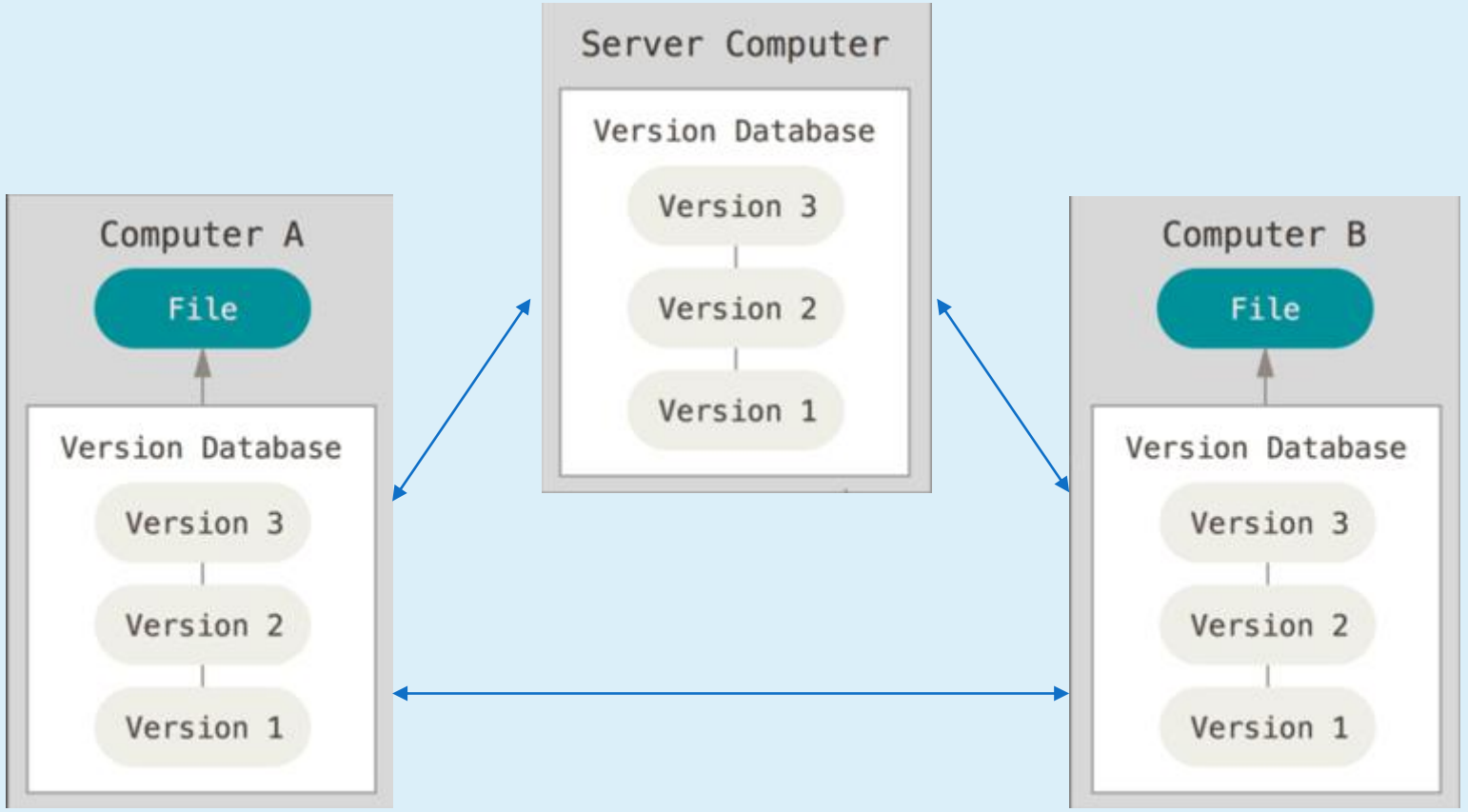
CONTROLO DE VERSÃO

TIPOS – SISTEMAS CENTRALIZADOS

- Em termos de desvantagens, se o servidor falhar então os utilizadores perdem todos o acesso e pior ainda, se os ficheiros no servidor se corromperem e não houver backups então todo o projeto é perdido exceto os ficheiros que tenham sido copiados para os computadores (este problema também existe nos sistemas locais).

CONTROLO DE VERSÃO

TIPOS – SISTEMAS DISTRIBUÍDOS



CONTROLO DE VERSÃO

TIPOS – SISTEMAS DISTRIBUÍDOS

- Este tipo de sistemas resolve as desvantagens dos sistemas centralizados. Neste tipo cada utilizador copia todo o repositório (Clonar) para o seu computador, incluindo todo o histórico. Deste modo, se algum servidor tiver problemas, há um backup no computador de cada um dos utilizadores.

- Para além destas vantagens, este tipo também funciona bem se houver vários repositórios diferentes a ser utilizados, aumentando a flexibilidade do trabalho das equipas.

- Git, o sistema mais utilizado atualmente, é deste tipo.

CONTROLO DE VERSÃO

GIT

O QUE É?

- Git é o sistema de controlo de versão mais usado mundialmente.
- Criado em 2005 por Linus Torvalds, Git é distribuído sobre licença open source e pertence à categoria de sistema de controlo de versão distribuído.
- Durante a criação do Kernel Linux, Torvalds mostrou se insatisfeito com a capacidade dos sistemas de controlo de versão contemporâneos de uso livre.
- O novo projeto, Git, tinha objetivos concretos de velocidade, facilidade de uso e capacidade para grandes projetos.

GIT

O QUE É?

- Estes objetivos, em conjunto com design não convencional, tornariam o Git incomparável com a concorrência.

- Com o passar do tempo o Git passou a ser utilizado por muitos estudantes e desenvolvedores e quase todos os ambientes de desenvolvimento o suportam, o que o tornou a escolha óbvia para muitas empresas.

BENEFÍCIOS

Trabalho em simultâneo e offline

- Sendo um sistema distribuído, cada utilizador tem uma cópia local dos ficheiros e por isso podem trabalhar simultaneamente nos seus ramos e até offline uma vez que quase todas as operações funcionam localmente.

Desenvolvimento mais rápido

- Desenvolvimento por ramos permite implementar uma versão estável do software sem impedir que o desenvolvimento de outras funcionalidades continue noutros ramos. Esta flexibilidade acelera o lançamento de novas atualizações do software.

BENEFÍCIOS

Integração nas ferramentas

- Devido à sua popularidade, as principais ferramentas de programação suportam o Git e as suas funcionalidades por definição, tornando a sua adoção e uso ainda mais fácil.

Forte suporte da comunidade

- Sendo o Git open source e extremamente popular, não há falta de ferramentas e recursos disponíveis para as equipas.

BENEFÍCIOS

Pull Requests

- Pull Requests permitem as equipas discutir alterações aos ficheiros antes de as integrar no ramo principal. Isto permite melhorar a comunicação dentro da equipa e a partilha de conhecimento. Plataformas como o GitHub e Azure DevOps permitem expandir o uso de pull requests com comentários, visualizações e votos para aprovar a programação.

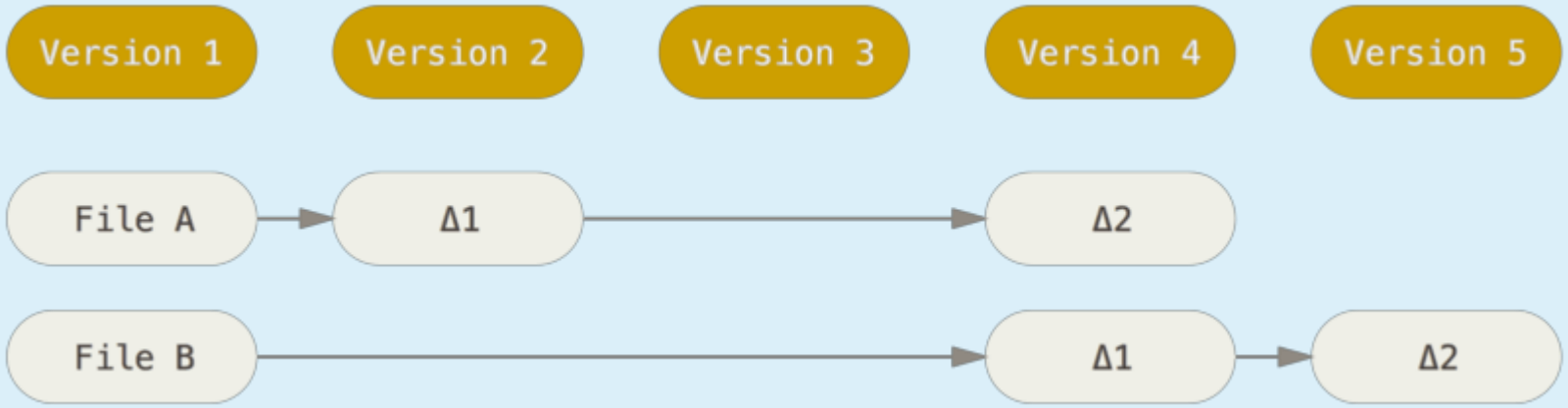
Regras de ramificações (Branch Policies)

- Plataformas como o GitHub e Azure DevOps permitem a aplicação de regras de ramificações que ajudam a garantir a qualidade e a segurança dos ramos.

GIT

COMO FUNCIONA?

- A grande diferença entre o Git e outros sistemas de controlo é a maneira a perspetiva sob a informação.
- Conceptualmente, os outros sistemas guardam a informação como uma lista de alterações dos ficheiros. Para estes, a informação é um conjunto de ficheiros e as alterações feitas ao longo do tempo.



GIT

COMO FUNCIONA?

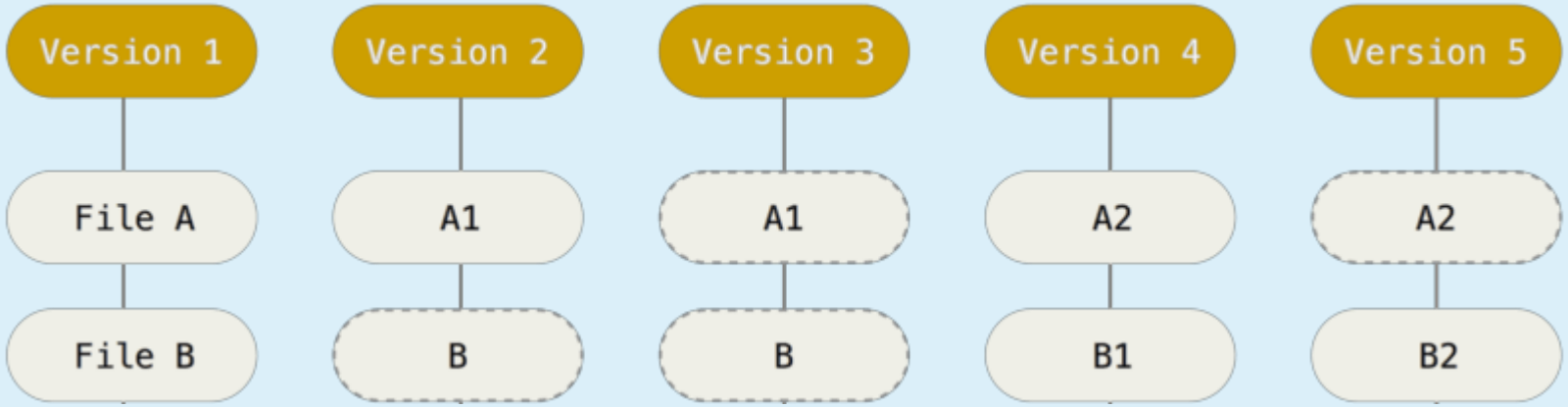
- Git tem uma perspetiva diferente: Vê a informação como uma serie de instantâneos (Snapshots) de um sistema de ficheiros em miniatura.

- Cada vez que se guarda o estado do projeto o Git basicamente tira uma “foto” da situação nesse momento e guarda uma referencia desse instantâneo.

- A informação é um linha de instantâneos.

GIT

COMO FUNCIONA?



- Esta perspetiva torna o Git uma espécie de mini sistema de ficheiros do que apenas um registo de alterações como nos outros sistemas de controlo de versão.

GIT

COMO FUNCIONA?

Quase todas as operações são locais!

- Este facto torna o Git muito mais rápido do que outros sistemas.
- É possível continuar a trabalhar offline.

Integridade

- Tudo é verificado (checksummed) antes de ser guardado e reconfirmado pela mesma verificação.
- É impossível alterar os ficheiros sem o Git saber. Não se perde informação em transito nem por corrupção.
- Esta verificação é a referencia que o Git usa na sua base de dados.

GIT

COMO FUNCIONA?

Estes são os três estados possíveis de um ficheiro no Git:

Modified (Alterado)

- Alterou se o ficheiro mas ainda não se submeteu á base de dados.

Staged (Preparado)

- Marcou se este ficheiro alterado na sua versão atual para fazer parte da próxima submissão.

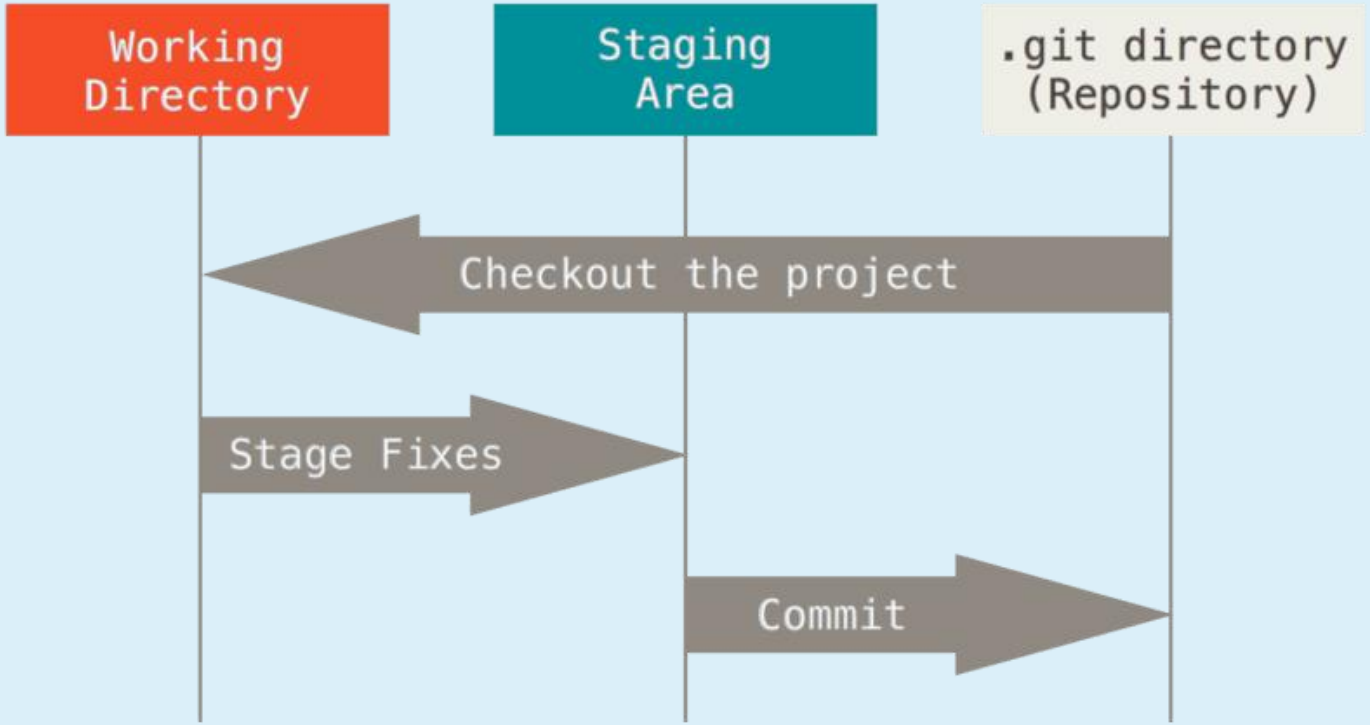
Committed (Submetido)

- O ficheiro está guardado de forma segura na base de dados local.

GIT

COMO FUNCIONA?

E estas são as três secções de um projeto Git:



GIT

COMO FUNCIONA?

Working Directory

- Copia de trabalho (Checkout) de uma versão do projeto. Retirada da base de dados e colocada no disco local para se usar ou alterar.

Staging Area

- Ficheiro que guarda informação sobre o que fará parte do próximo Commit. Tambem chamado de Index.

Repository (Repositório)

- Onde é guardado a metadata e a base de dados do projeto. É a parte mais importante e é o que é copiado quando se clona (Clone) um repositório de outro computador.

GIT

COMO FUNCIONA?

Branching

- Quase todos os sistemas de controlo de versão suportam Branching (Ramificação) mas o Git consegue fazer estas operações muito mais rapidamente e incentiva os utilizadores a utilizar vários branches em simultâneo no dia-a-dia.

Pull Request

- É uma solicitação feita por um colaborador num projeto para incorporar as suas alterações num Branch do repositório.
- Inclui as alterações efetuadas e uma descrição do que foi alterado e porquê.
- Permite facilitar a comunicação entre os membros de uma equipa.

CONTROLO DE VERSÃO

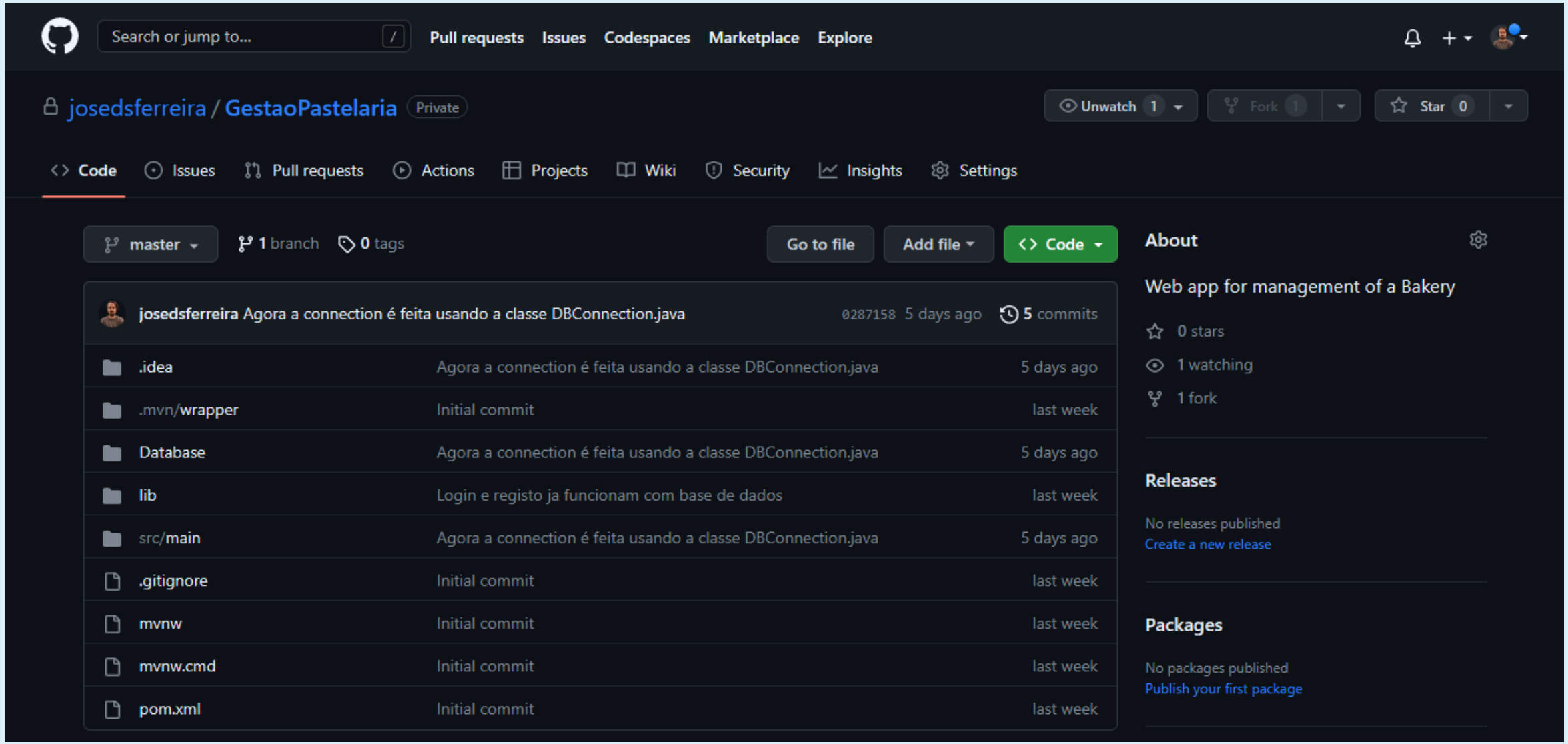
GITHUB

O QUE É?

- GitHub fornece um serviço de hosting de repositórios Git na cloud. Em essência, torna muito mais simples para indivíduos e equipas a utilização do Git para colaboração e controlo de versões.

- Devido ao design amigável do GitHub, até os programadores inexperientes podem beneficiar do Git. Sem GitHub, a utilização de Git requer tipicamente um pouco mais de experiência na linha de comando e de know-how técnico.

GITHUB


Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
josedsferreira / GestaoPastelaria
Private
Unwatch 1
Fork 1
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
josedsferreira Agora a connection é feita usando a classe DBConnection.java
0287158 5 days ago
5 commits
.idea Agora a connection é feita usando a classe DBConnection.java 5 days ago
.mvn/wrapper Initial commit last week
Database Agora a connection é feita usando a classe DBConnection.java 5 days ago
lib Login e registo ja funcionam com base de dados last week
src/main Agora a connection é feita usando a classe DBConnection.java 5 days ago
.gitignore Initial commit last week
mvnw Initial commit last week
mvnw.cmd Initial commit last week
pom.xml Initial commit last week
About
Web app for management of a Bakery
0 stars
1 watching
1 fork
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package

GITHUB

O FLOW

- O GitHub é desenhado a volta de um workflow especifico centrado nos Pull Requests.

- Funciona quer estejam a trabalhar numa equipa bastante coesa num só repositório, numa empresa distribuída globalmente ou com uma rede de estranhos a trabalhar num projeto.

- Eis como funciona geralmente:

GITHUB

O FLOW

1. Criar um Fork
2. Criar um Branch a partir do Master
3. Desenvolver o projeto com alguns Commits
4. Fazer Push deste Branch (integrar a ramificação) para o projeto GitHub
5. Abrir um Pull Request no GitHub
6. Discutir e opcionalmente continuar a desenvolver mais Commits
7. O dono do projeto faz Merge (integra) ou fecha o Pull Request
8. Sincronizar o Master atualizado de volta para o nosso Fork

PARA SABER MAIS:

Pro Git book, de Scott Chacon e Ben Straub

- Disponível em git-scm.com

Documentação GitHub

- Disponível em docs.github.com