Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

2019/2020

Construção de Sistemas de Software

Sistemas de Controlo de Versões para Desenvolvimento de Software

Desenvolvimento Concorrente de Software

- A construção de software em grande escala implica ter software a ser desenvolvido **cooperativamente e concorrentemente** por diferentes elementos de uma equipa

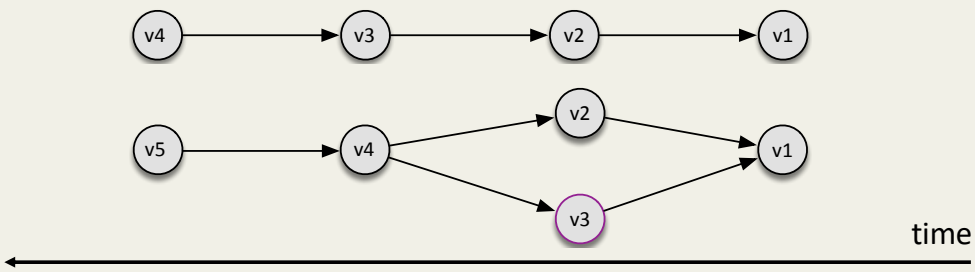
- Atualmente a utilização de **Sistemas de Controlo de Versões (VCS)** é a melhor solução disponível para os problemas e desafios subjacentes ao desenvolvimento concorrente de software

Utilização de Sistema de Controlo de Versões

- A **utilização de um VCS** para o desenvolvimento de software consiste em **ter um ou mais repositórios** que
  - podem ser **acedidos** pelos vários elementos da equipa **em simultâneo**
  - **registam as versões** de cada elemento que cada membro da equipa produz
  - **juntam** elementos (código e outros artefactos), produzindo uma versão consolidada do software desenvolvido pela equipa
  - no processo de junção, podem **detetar conflitos** que têm de ser resolvidos manualmente

Sistemas de Controlo de Versões (VCS)

- Sistemas que servem os seguintes propósitos
  - fazer a gestão de múltiplas versões de um conjunto de ficheiros, que vão sendo criadas ao longo do tempo, mantendo o histórico
  - permitir várias pessoas trabalhar simultaneamente no mesmo projeto, integrando o trabalho feito por cada uma
  - ajudar a responder a questões como: o que mudou, quando mudou, quem mudou, porque mudou
- Usam repositórios que contêm as **histórias** de edição/versões, as quais formam um **DAG** (*directed acyclic graph*)



Sistemas de Controlo de Versões (VCS)

- Podem ser agrupados de acordo com
  - a forma como fazem a **gestão da concorrência**
  - e em relação **à distribuição dos repositórios**

- **Gestão da concorrência:**
  - Pessimistas
  - Otimista

- **Repositórios:**
  - Centralizados
  - Distribuídos

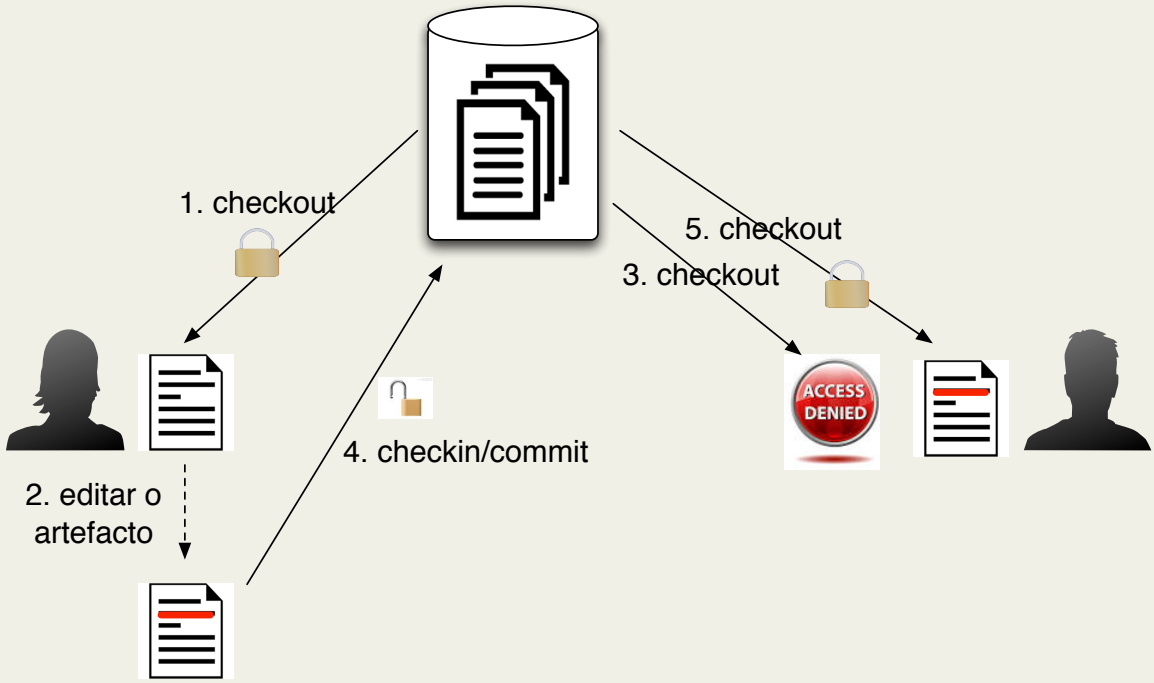
VCS Pessimistas

- Fazem o bloqueio dos recursos extraídos por cada membro da equipa e só permitem o acesso por um membro da equipa a cada ficheiro
- Esta estratégia era a concretizada pelos primeiros VCS mas foi abandonada

- **Desvantagem**: Reduzem a oportunidade de trabalho em paralelo (o mesmo recurso não pode ser usado por mais de um membro da equipa)
- **Vantagem**: São simples de concretizar e de usar. Não têm problemas com resolução de conflitos.

- Exemplo: RCS (GNU Revision Control System)

VCS Pessimistas

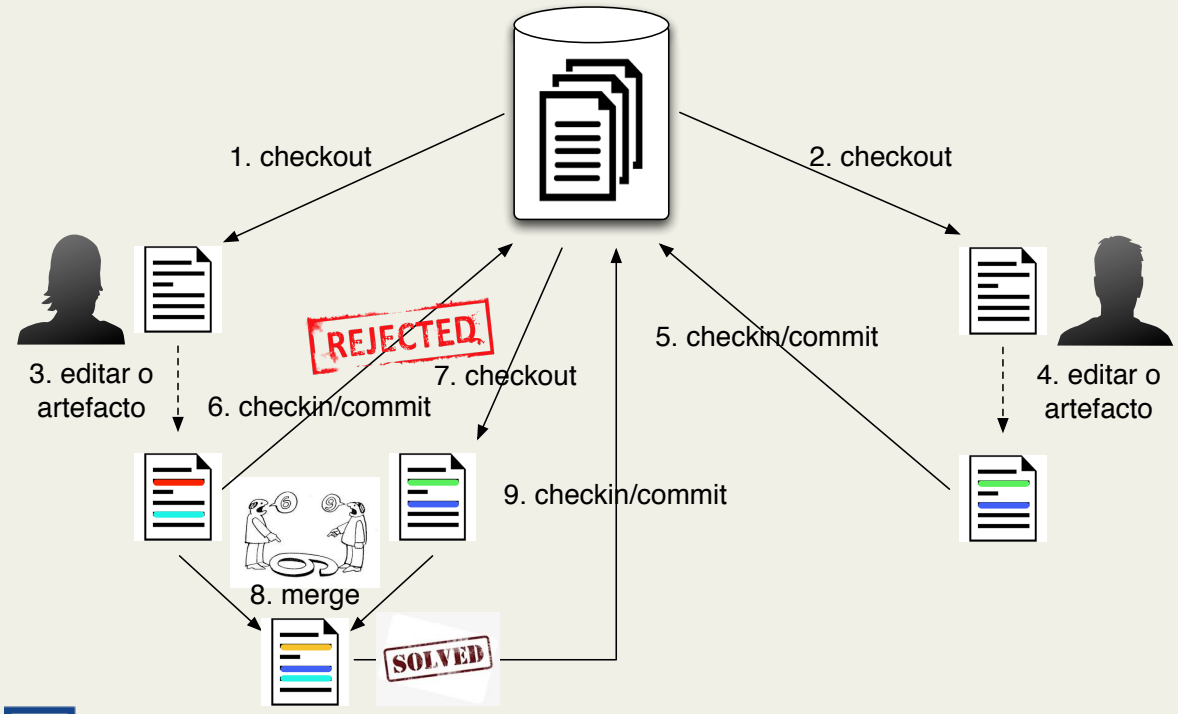


VCS Otimistas

- Cada membro da equipa **extrai uma cópia** do trabalho para a sua máquina
- Trabalha em paralelo, sem quaisquer restrições, no projeto (trabalha sobre sobre a sua cópia)
- Quando publica o seu trabalho há lugar a uma **junção** com os trabalhos anteriormente produzidos pelos restantes membros da equipa
- Pode haver lugar a **conflitos** quando ambos os membros da equipa alteram mesmo ficheiro/a mesma zona do ficheiro

- **Exemplos**: CVS, SVN, GIT, Mercurial, Bazaar

VCS Otimistas



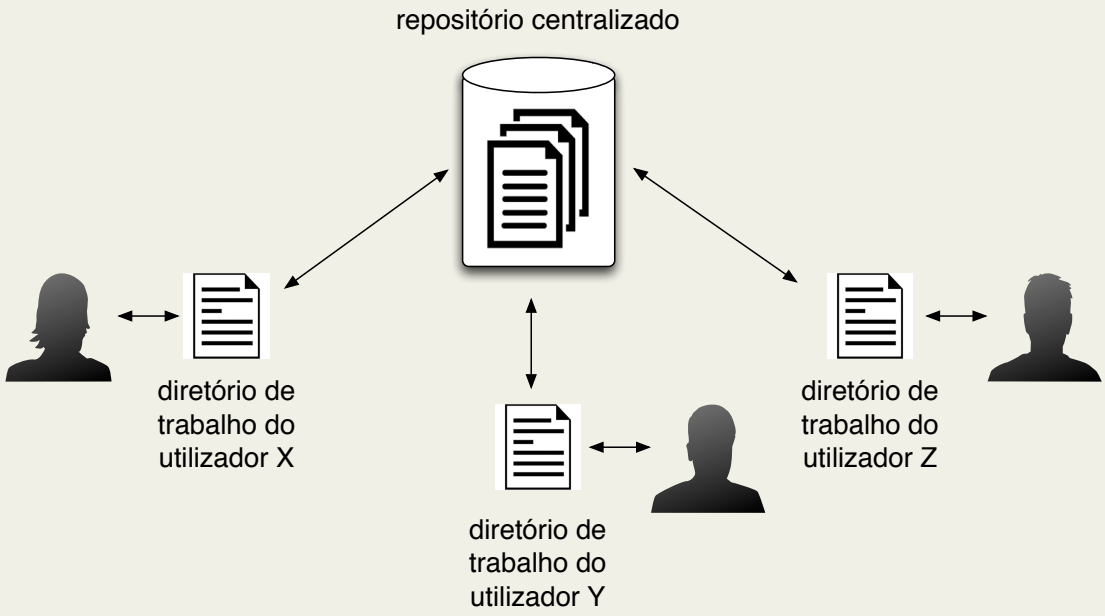
Ciências ULisboa | Informática

VCS Centralizados

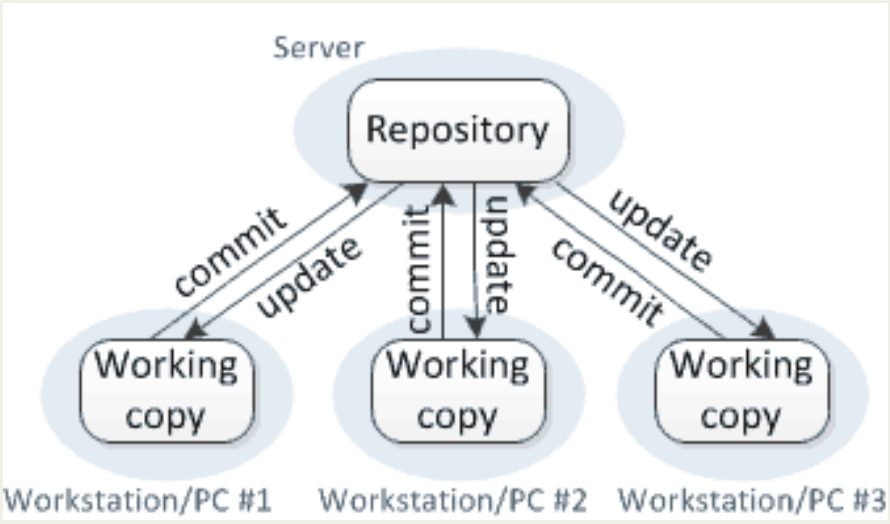
- Existe um repositório central onde são mantidas todas as versões do trabalho
- Cada membro da equipa
  - extrai uma **cópia inicial** do repositório para a sua máquina (***checkout***)
  - efetua alterações na sua **cópia de trabalho local** e publica-as (***commit***)
  - integra alterações publicadas por outros membros da equipa na sua cópia local (***update***)
  - no processo de integração pode ter que resolver **conflitos** antes de conseguir publicar as suas alterações
- **Exemplos**: CVS, SVN e TFS (Team Foundation Server)

Ciências ULisboa | Informática

VCS Centralizados

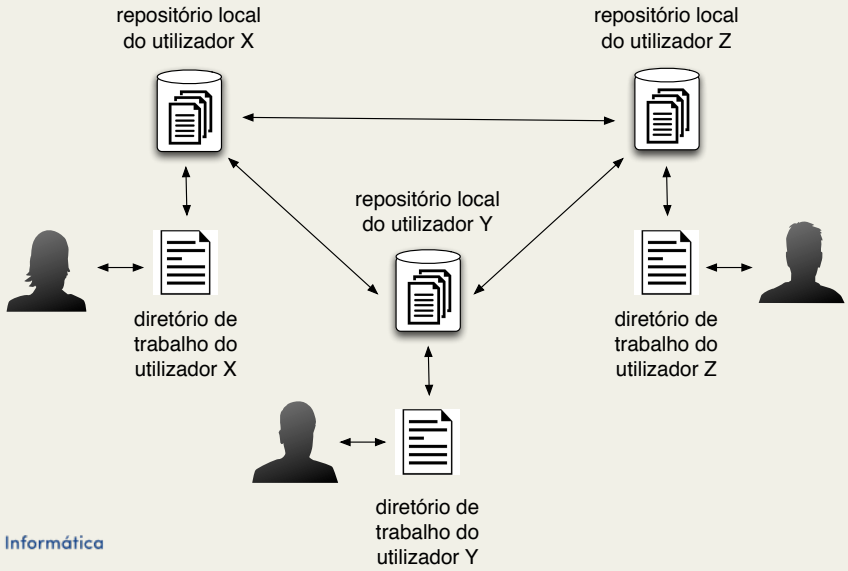


VCS Centralizados



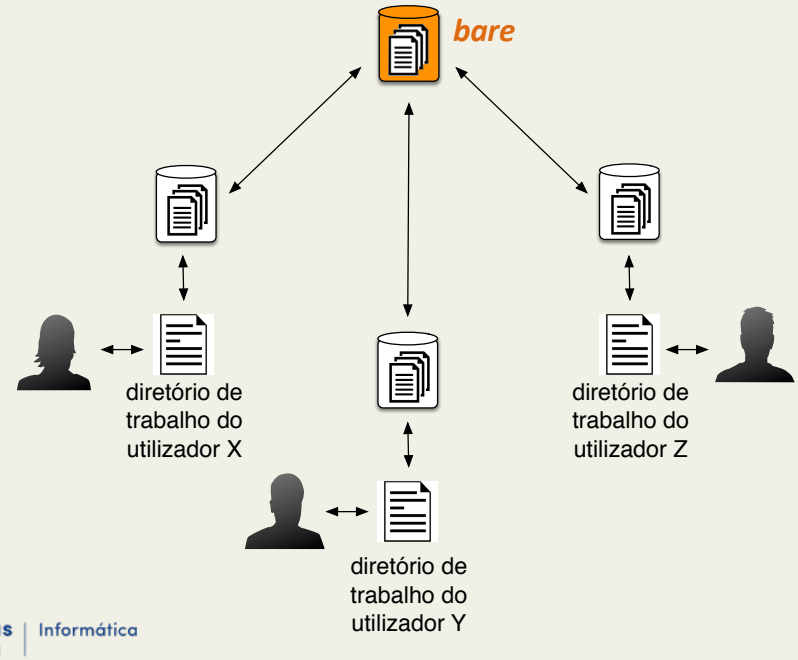
VCS Distribuídos

- Não existe o conceito de um repositório central; existem vários repositórios que, depois de sincronizados, são iguais
- Além dos vários repositórios, existem também as cópias locais de trabalho (***working copy***)



VCS Distribuídos

- Por razão de organização do trabalho é conveniente, em muitas situações, ter um **repositório central** (***bare***), i.e., que não tem ***working copy***



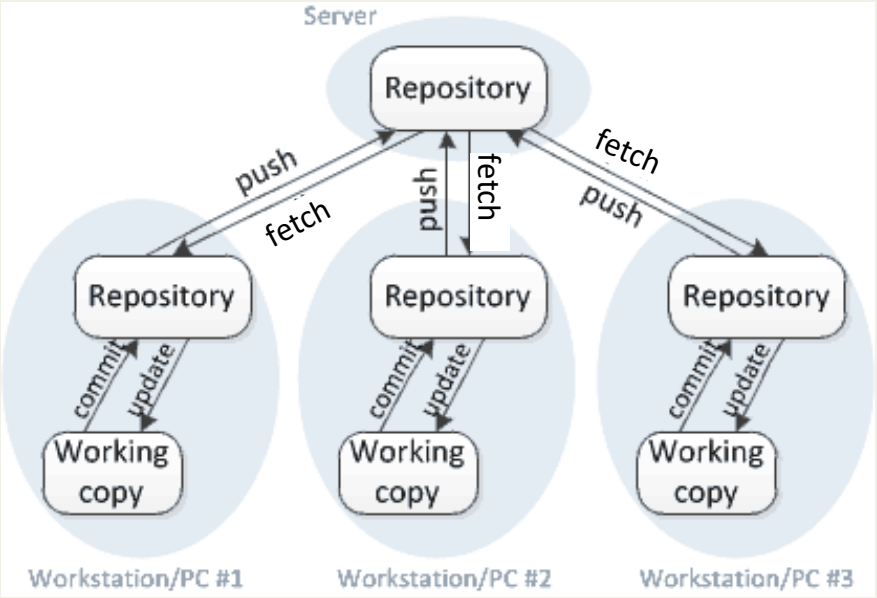
VCS Distribuídos

- Neste cenário, cada membro da equipa faz **uma cópia do repositório** para a sua máquina (***clone***)
- A **sincronização de cada repositório** com o central ocorre **apenas** em resultado de operações de publicação e obtenção da informação de um repositório local com o outro repositório:
  - **Push** publica a informação do local
  - **Fetch** obtém informação do central
  - A operação de junção — **Merge** — ocorre no repositório
- As duas primeiras operações não afetam o estado da cópia de trabalho

VCS Distribuídos

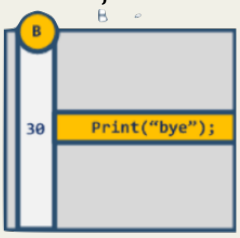
- A relação entre o repositório local e a versão de trabalho é semelhante à que ocorre nos repositórios centralizados:
  - Extração de uma cópia de trabalho (***checkout/update***)
  - Publicação das modificações produzidas (***commit***)
  - Como a junção ocorre no repositório, não há lugar à operação de junção com a cópia local
- Estas operações não afetam qualquer outro repositório além do local
- **Exemplos**: Git, Mercurial, Bazar

VCS Distribuídos

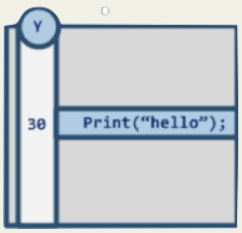


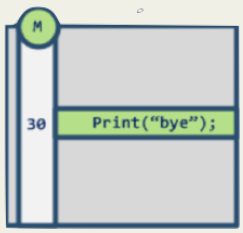
Conflitos

- Os VCS otimistas permitem que vários utilizadores editem simultaneamente as suas cópias de trabalho
- Têm que ter mecanismos para detetar mudanças conflituosas e mecanismos para ajudar a resolvê-las
- Em documentos de texto, é comum a análise ser baseada nas linhas



Conflito sintático
Conflito de edição





Fonte: https://www.drdobbs.com/tools/three-way-merging-a-look-under-the-hood

Conflitos e Resolução de Conflitos

- Os VCS otimistas são capazes de juntar as mudanças simultâneas feitas por diferentes utilizadores em diferentes documentos
- Em geral são também capazes de juntar automaticamente algumas mudanças simultâneas feitas por diferentes utilizadores no mesmo documento

  comum para documentos de texto:

  a versão final de cada linha é a versão original se não foi editada por nenhum dos utilizadores ou se foi apenas por um deles
- As mudanças que o VCS não consegue juntar automaticamente são apresentadas como conflitos para serem resolvidos manualmente

Merge

- A operação de *update* modifica a cópia de trabalho (Y) com as edições que estão no repositório (X) e ainda não foram aplicadas na cópia de trabalho
- Num VCS centralizado, é possível fazer esta operação em qualquer altura, mesmo que a cópia de trabalho tenha alterações que ainda não foram publicadas (*commited*)
- Num VCS distribuído,
  - só é possível fazer esta operação se não houver alterações não publicadas
  - se tiver havido alterações, é primeiro preciso primeiro fazer *commit;* nessa altura o repositório vai ter edições que foram feitas concorrentemente (X e Y) e é preciso fazer *merge* destas mudanças e fazer *commit* do resultado
  - ficam assim registadas as decisões feitas na operação de *merge*

Boas Práticas

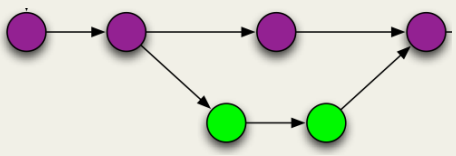
- Usar mensagens descritivas
- Fazer de cada *commit* uma unidade lógica (propósito)
- Evitar *commits* indiscriminados
- Incorporar as mudanças dos outros frequentemente
- Partilhar as mudanças frequentemente
- Coordenar-se com os colegas
- Ter em atenção que se deteção de conflitos é baseada em linhas, se deve evitar linhas grandes e ter atenção à reformatação
- Não fazer *commit* de ficheiros gerados

Branching

- Em muitas situações faz sentido ter o desenvolvimento a correr em vários ramos paralelos, **a evoluir de forma independente**, eventualmente para juntar mais tarde

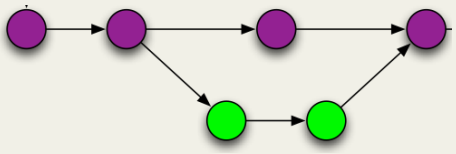
  e.g., a manutenção do XXX 1.0 e o desenvolvimento do XXX 2.0

- Padrões de utilização comuns
  - Para isolar mudanças exploratórias
  - Para isolar atividades de desenvolvimento como a correção de um *bug* ou desenvolvimento de *feature*
  - Para manutenção de versões anteriores



Branching

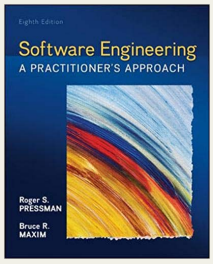
- O desenvolvimento paralelo e independente é possível abrindo, num determinado ponto *(commit)* de um ramo, um novo ramo
  - O ramo original é conhecido como ***parent*** ou ***upstream***
  - O ramo que não tem *parent* é conhecido como ***master*** *ou* ***trunk***
  - Modificações feitas no ramo *parent* não afetam o novo ramo e vice-versa
- Geralmente a ideia é, mais tarde, integrar as mudanças realizadas no novo ramo no *parent* através de um *merge*
- Quando a ideia é prosseguir independentemente é chamado um ***fork***



VCS vs SCM

- O controlo de versões é um componente da **Gestão de Configurações de Software** (***Software Configuration Management***)

  SCM is a set of ***tracking and control activities*** *that are initiated when a software engineering project begins and terminates when software is taken out of operation*



- Outras componentes
  - *Build Managers*
  - *Package Managers*
  - *Deployment Managers, VMs, Containers*
  - ...

Git

- VCS que vão usar intensivamente em CSS
- Nas aulas TPs vão aprender mais sobre este VCS, nomeadamente:
  - Sincronização de diretório de trabalho com repositório via índice
    - *Checkout, Commit, Update, Add, Remove*
  - Sincronização de repositórios (*remotes*)
    - *Clone, Push, Fetch, Merge, Pull*
  - Ramos (*branches*)
    - DAG (*Direct Acyclic Graph*) de *commits*
    - Criação de ramos (*branch*), junção de ramos (*merge*), eliminação de ramos
  - Ligação a novos repositórios

Fluxos de trabalho

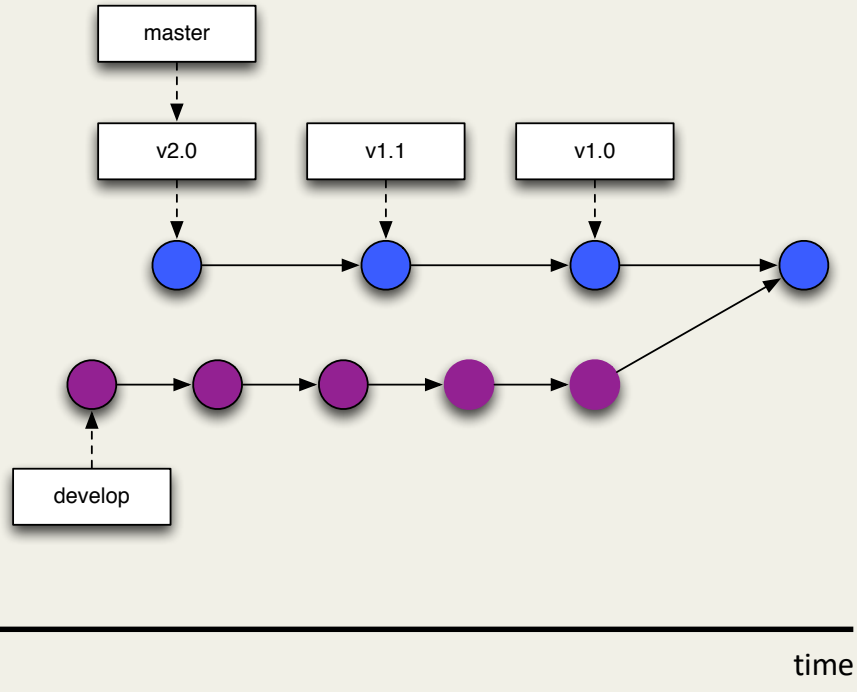
- Existem também **boas práticas** no que diz respeito à organização do fluxo de trabalho
- Estas são consolidadas em torno de **tipos de fluxos de trabalho**
  - disciplinam a forma como se usa toda a flexibilidade do VCS
  - especialmente crítica no caso de DVCS, dado que a sua flexibilidade é muito maior
- Quando se trabalha em equipa é especialmente importante **acordar qual o tipo de fluxo de trabalho** que se vai seguir

Fluxos de trabalho

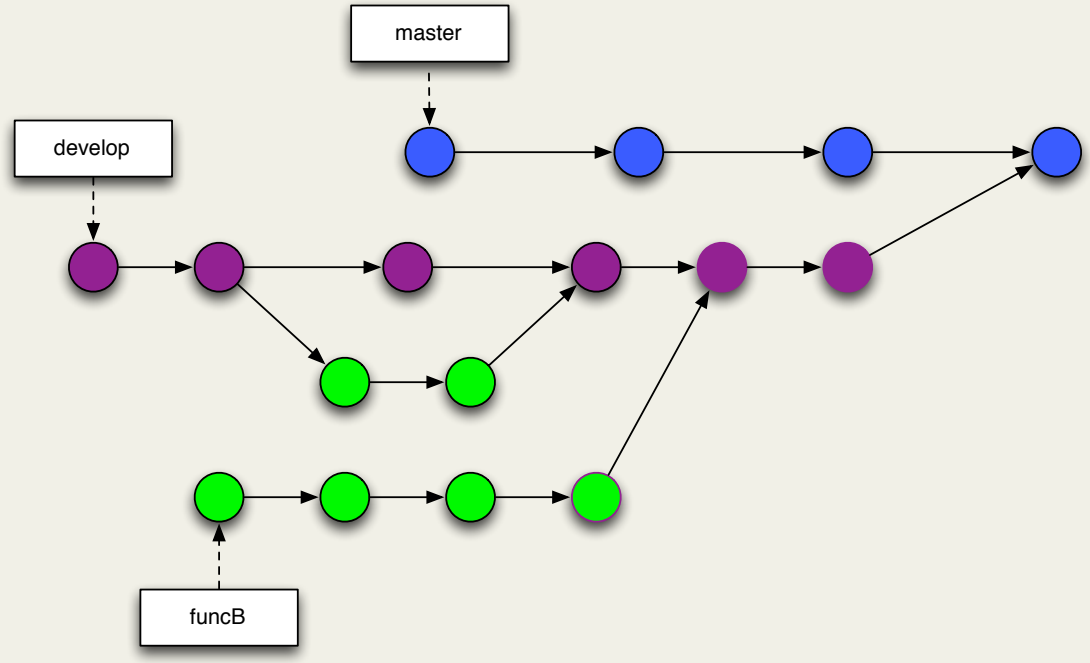
Alguns **tipos de fluxos de trabalho** mais comuns são:

- **Centralizado**
  - sem ramos
  - todos mudanças publicadas no *master*
- ***Feature Branching***
  - um ramo por funcionalidade
- ***Git flow***
  - modelo de ramificação desenhado em torno de *releases* do projeto
- ***Forking workflow***
  - para desenvolvimento em ambiente *open source*

Git Flow: ramos principais

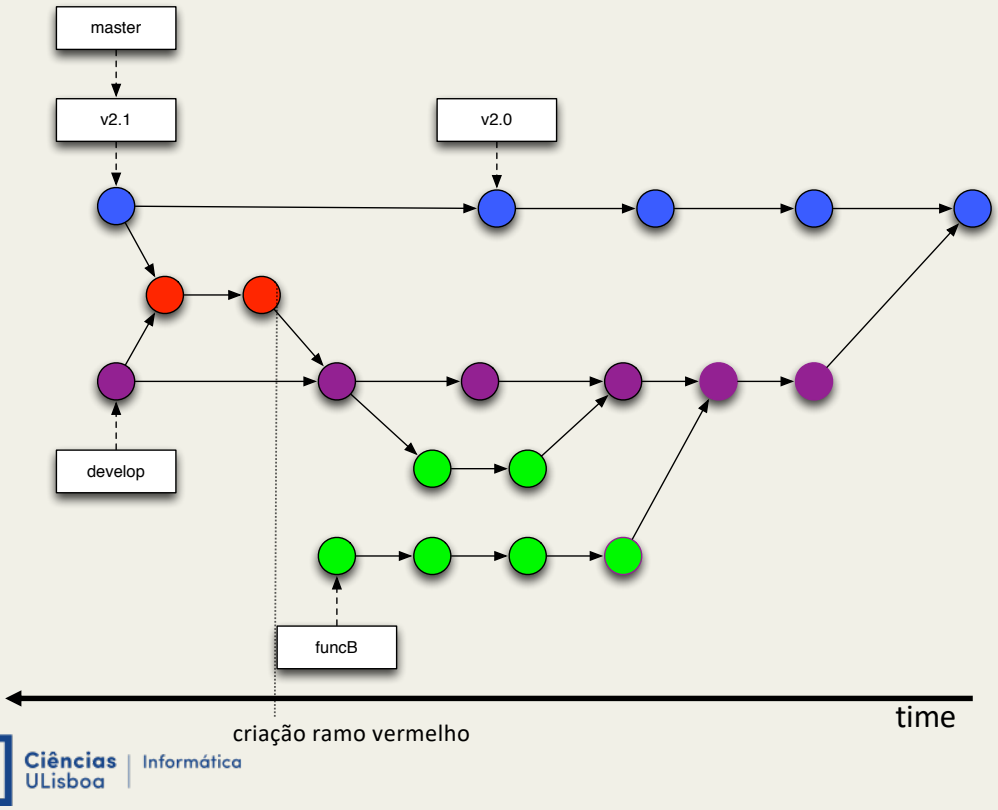


Git Flow: desenvolvimento de novas funcionalidades



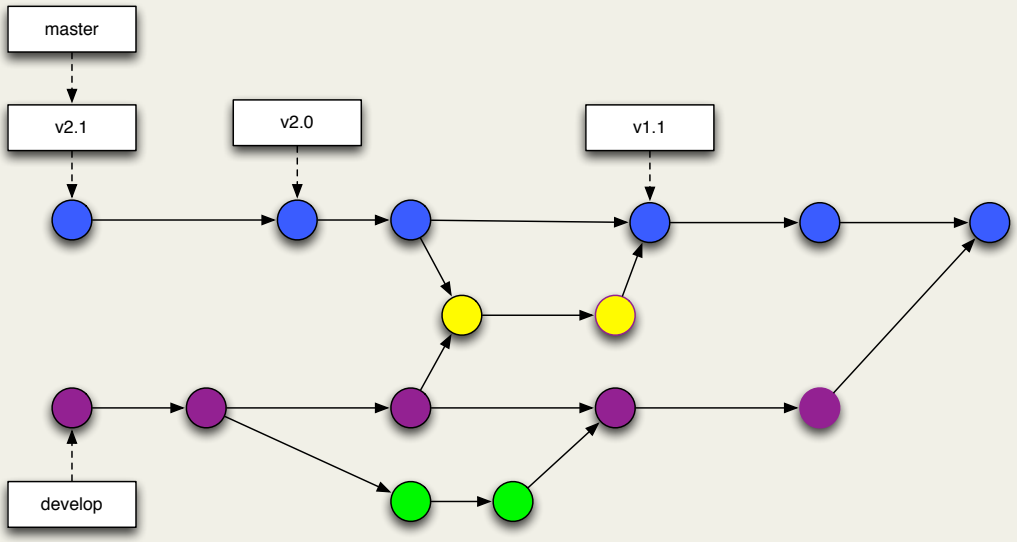
Ciências ULisboa | Informática

Git Flow: construção de uma nova versão da aplicação



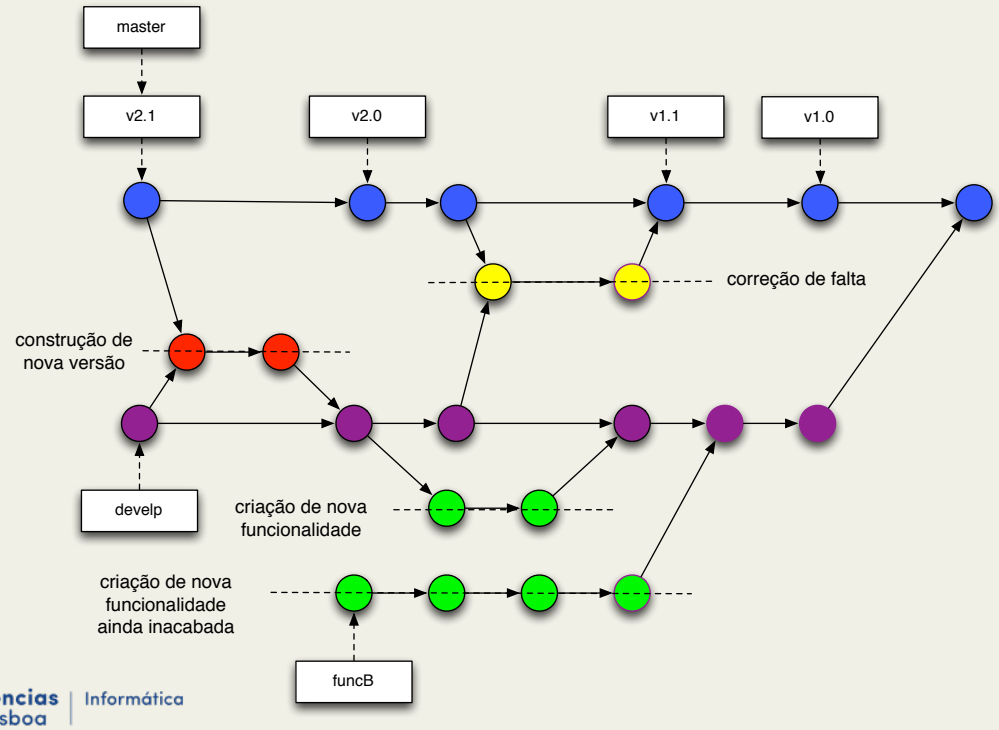
Ciências ULisboa | Informática

Git Flow: correção de faltas detetadas em produção



Ciências ULisboa | Informática

Git Flow: resumo dos fluxos de trabalho



Ciências ULisboa | Informática

CVS vs Qualidade do software

- **Livre de *bugs***
  - identificar quando e onde alguma coisa deixou de funcionar
  - procurar outros problemas semelhantes
  - ganhar confiança que o código não foi mudado acidentalmente
- **Fácil de Entender**
  - porque foi uma mudança feita
  - o que mudou ao mesmo tempo
  - a quem posso perguntar coisas sobre este código
- **Pronto para ser mudado**
  - gerir e organizar mudanças
  - aceitar e integrar mudanças feitas por outros membros
  - isolar trabalho especulativo/exploratório em ramos

Sumário

- Construção de software em grande escala e o desenvolvimento de software em equipa efetuado de forma concorrente
- Sistemas de controlo de versões
  - gestão de concorrência otimista e pessimista
  - repositórios centralizados e distribuídos
  - grafo de versões, conflitos, ramos, *merge*
- As atividades fundamentais no contexto de DVCS
  - criar e clonar um repositório;
  - registar alterações no repositório (*commit*);
  - extrair cópia local (*update/checkout)*
  - sincronizar o repositório local com repositórios remotos (*push/fetch*);
  - criar ramos (*branch)*
- Organização do fluxo de trabalho no desenvolvimento de software utilizando sistemas de controlo de versões como, por exemplo, o Git.