Introdução ao *Git*

Laboratório de Programação

Pedro Vasconcelos, Manuel Barbosa, DCC/FCUP

Fevereiro 2016

Controlo de Versões

Controlo de Versões

Um sistema de *controlo de versões* (VCS) é uma ferramenta que possibilita:

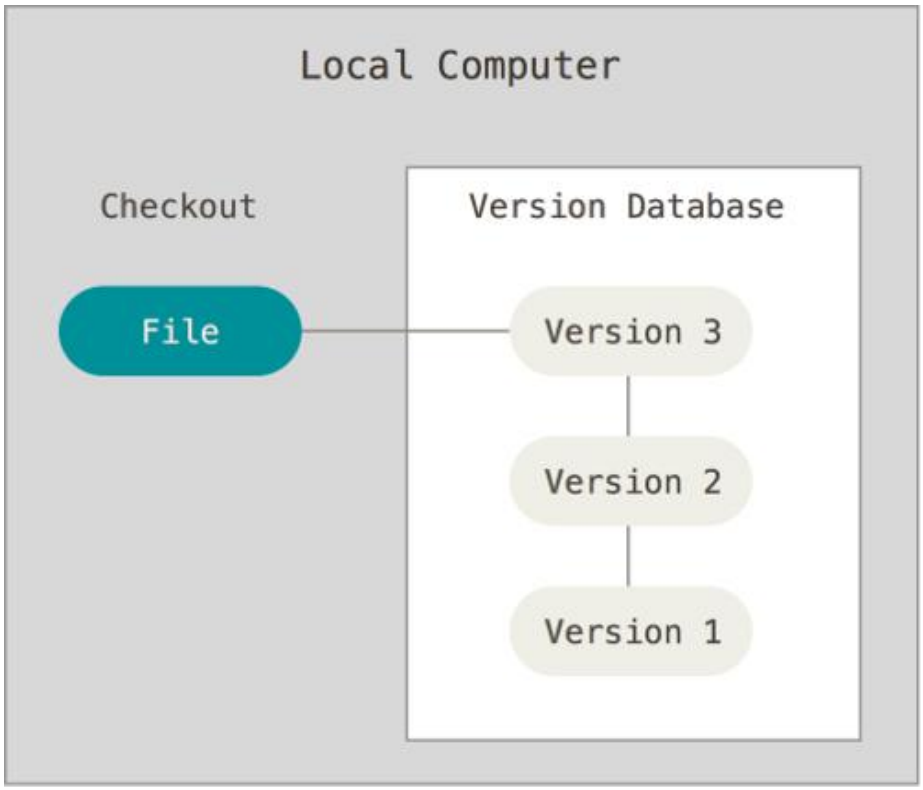
- arquivar ficheiros de um projeto (código-fonte e outros)
- registar alterações durante o desenvolvimento
- desfazer alterações ou recuperar versões anteriores
- sincronizar diferentes computadores
- colaborar com programadores (locais ou à distância)
- separar "troncos" de desenvolvimento (e.g. produção/desenvolvimento)

Sistemas locais

Os primeiros *sistemas VCS* registavam modificações de ficheiros apenas numa base de dados local.

Exemplos: *SCCS*, *RCS*.

Sistemas locais (2)



Sistemas locais (3)

Permitem:

- registar e desafazer alterações
- recuperar versões anteriores

Não permitem:

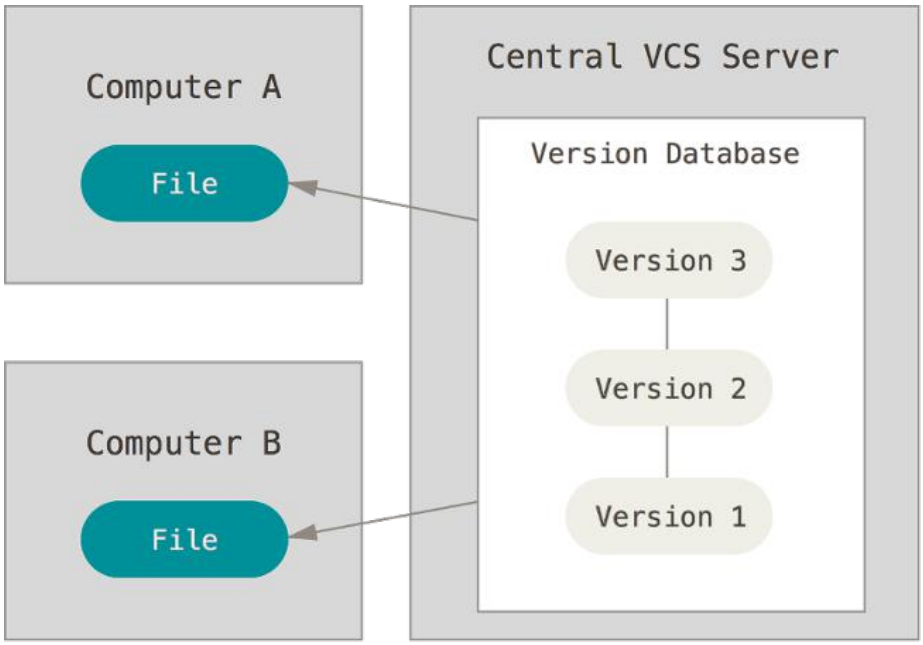
- sincronizar entre computadores
- colaboração entre programadores

Sistemas centralizados

Os *sistemas centralizados* registam alterações numa base de dados num servidor central; todos os colaboradores utilizam a mesma base de dados.

Exemplos: *CVS*, *SVN*

Sistemas centralizados (2)



Sistemas centralizados (3)

Vantagens sobre os sistemas locais:

- permitem sincronização entre computadores
- permitem colaboração entre programadores

Limitações:

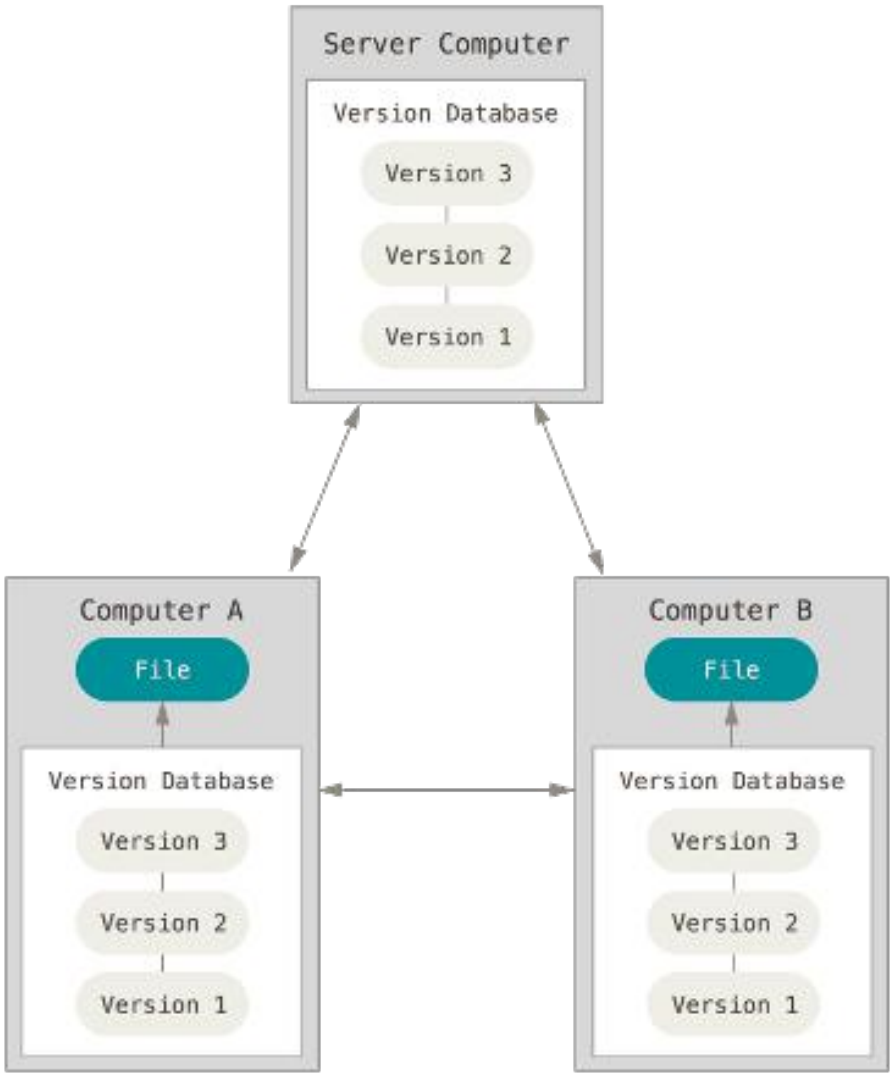
- não permitem trabalhar *off-line*: é necessário aceder ao servidor para registar modificações
- o servidor constitui um ponto crítico de falha

Sistemas distribuídos

Nos *sistemas distribuidos* cada cópia do repositório mantém também uma *cópia completa da base de dados de modificaçoes.*

Exemplos: *GNU arch*, *Darcs*, *Mercurial*, *Git.*

Sistemas distribuídos (2)



Sistemas distribuidos (3)

Vantagens sobre sistemas centralizados:

- todas as cópias do repositório contêm a **historia completa**
- permitem **trabalhar localmente** registando alterações mesmo sem acesso a rede

- não há um ponto crítico de falha: qualquer cópia do repositório contém a história completa
- facilita introdução de **mudanças experimentais** separadas das estáveis ("*branching*")

Desvantagem:

- a utilização pode ser mais complexa do que os sistemas centralizados

Git

O que é o *Git*?

- Um sistema de controlo de versões distribuído desenvolvido desde 2005
- Inicialmente concebido para o *kernel* Linux
- Muito usado em projetos *open-source*: github.com
- Características:
  - conceção simples mas poderosa
  - adequado a projectos grandes, com muitos ficheiros e com história longa
  - eficiente em espaço e recursos computacionais
  - suporte para desenvolvimento não-linear ("*branching*")
  - completamente distribuído

O *Git* dá-te super-poderes!

Utilizar *Git* é um método de trabalho valioso:

- Garantimos a **integridade** dos ficheiros
  (diga adeus às *pen-drives* para transportar ficheiros entre casa e trabalho)
- Usando um repositório remoto, **temos sempre um *backup*** caso algo corra mal
- Permite **experimentar e explorar** sem termos medo das consequências (dá sempre para voltar atrás caso seja necessário)
- As mensagens de *commits* permitem-nos rever os **motivos das alterações**

Como funciona

- Cada repositório mantém um conjunto de ficheiros e diretórios (um mini-sistema de ficheiros)

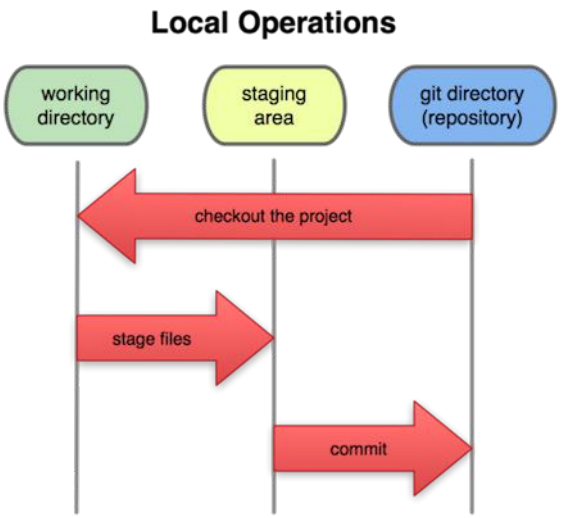
- Podem mudar ao longo do tempo:
  - acrescentar/remover ficheiros
  - editar o conteúdo
- Quando o utilizador regista uma modificação (*commit*):
  - guarda o estado atual de *todos* os ficheiros marcados (*snapshot*)
  - para os ficheiros não modificados: guarda apenas uma referência para o estado anterior
- Um repositório *Git* é uma sequência destes "*snapshots*"

Fluxo de trabalho básico

Três fases:

1. **Modify**: modificar os ficheiros no directório de trabalho.
2. **Stage**: adicionar *snapshots* dos ficheiros à "área de estágio" (*staging area*).
3. **Commit**: registar esse *snapshots* na base de dados do *Git* juntamente com uma *mensagem* de arquivo.

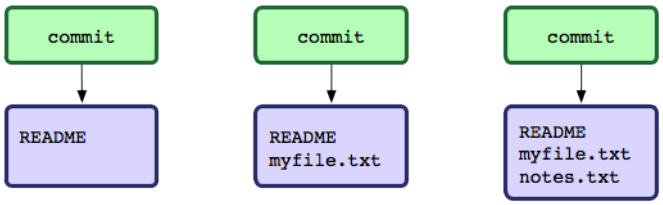
Operações básicas



Commit

- Uma "imagem" dos ficheiros tal como estavam quando estagiados

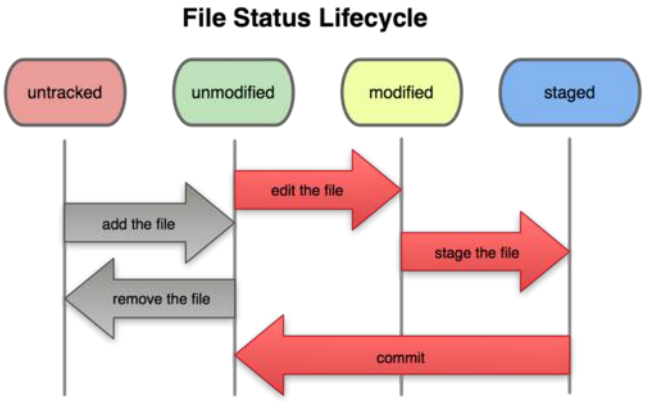
- Uma **mensagem de arquivo** que descreve a alteração efetuada
- Meta-informação do autor e data



Qualquer *commit* pode ser inspecionado e recuperado se assim quisermos.

Ciclo de Vida

Os ficheiros no directório de trabalho podem estar em quatro estados diferentes em relação ao *commit* atual.



Utilização do *Git*

Configuração inicial

Antes de usar *Git* pela primeira vez:

Escolher a sua identidade

```
git config --global user.name "John Doe"
git config --global user.email john@doe.com
```

Mais configurações (opcionais)

```
git config --list
```

Obter ajuda

```
git help
```

Inicializar um repositório local

```
mkdir my_repo
cd my_repo
git init
```

- inicializa um diretório `my_repo/.git` que vai conter a base de dados do repositório
- o repositório começa *vazio*: temos de adicionar ficheiros e/ou diretórios

Adicionar ficheiros

```
git add ficheiro1
git add ficheiro2
```

Podemos também adicionar vários ficheiros de uma só vez:

```
git add ficheiro1 ficheiro2
```

Os ficheiros ficam na *área de estágio* — temos de fazer um *commit* para os registar na base de dados do *Git*.

Primeiro *Commit*

```
git commit -m "Iniciar o meu repositório"
```

Podemos também adicionar ou remover ficheiros mais tarde.

Modificar ficheiros

Depois de fazer alterações aos ficheiros na área de trabalho (e.g. usando um editor de texto):

1. adicionar novamente os ficheiros à àrea de estágio
2. registar um *commit* com uma mensagem descritiva.

```
emacs    # editar ficheiros
...
git add ficheiro1 ficheiro2
git commit -m <mensagem>
```

Consultar o estado

```
git status
  # ver estado da área de trabalho e de estágio
```

`Changes to be commited` ficheiros modificados **que vão ser incluidos** no próximo *commit*

`Changes not staged for commit` ficheiros modificados mas **não incluidos** no próximo *commit*

`Untracked files` ficheiros na área de trabalho que o *Git* não está a gerir

Outras consultas

```
git diff
 # listar modificações desde o último *commit*
```

```
git log
  # Listar o histórico de *commits*
```

Estes comandos aceitam muitas opções extra; ver ajuda no manual:

```
git diff --help
git log --help
```

Sincronização e colaboração

Repositórios remotos

Em *Git* todos os repositórios têm a mesma estrutura e suportam os mesmos comandos.

Um *repositório remoto* é apenas um diretório *Git* que está localizado noutro computador.

Usar um repositório remoto

Para copiar um repositório remoto já inicializado basta fazer *clone*:

```
git clone URL-do-repositório-remoto
```

- O URL de um servidor remoto é tipicamente `https://...` ou `ssh://...`
- Obtemos uma cópia local completa do repositório que podemos consultar e modificar livremente

Usar um repositório remoto (cont.)

Podemos fazer alterações à copia local dos ficheiros tal como anteriormente.

```
  # editar ficheiros localmente
...
git add ficheiro1
git add ficheiro2
...
git commit -m "mensagem descritiva..."
```

Após este comando, esta modificação foi registada **apenas no repositório local** — ainda nada foi enviado ao servidor remoto!

Enviar modificações

Para enviar os seus *commits* locais para o repositório remoto usamos o comando *push*.

```
git push
```

Receber modificações

Para receber alterações que outros tenham enviado ao repositório remoto usamos *pull*:

```
git pull
```

Este comando descarrega *commits* no repositório remoto feitos desde a última vez aplica-os ao repositório local.

Como coordenar com colaboradores

Há várias formas de usar *Git* com repositórios remotos:

- com um *branch* único partilhado por todos os colaboradores — semelhante ao uso num sistema centralizado
- com *branches* distintos para desenvolvimento separado

Nesta unidade curricular

Sugerimos usar o *Git* de forma centralizada:

- com um repositório central num servidor *Gitlab*;
- serve como o *ponto de sincronismo* entre os colaboradores;
- com um único *branch* de desenvolvimento (*master*).

Repositório central

- Servidor *Gitlab*:
  https://gitlab01.alunos.dcc.fc.up.pt.
- Interface *web* para projetos em *Git*
- Semelhante ao *Github* mas mantido num servidor do DCC
- Cada aluno tem uma conta pessoal (*login/password* do LabCC)
- *Grupos* de dois alunos para os trabalhos — criados durante as aulas

Conflitos

Se um dos nossos *commits* modificar parte dum ficheiro que outro colaborador também editou, o *Git* vai sinalizar um **conflito**.

O *Git* **não resolve o conflito sozinho**; o utilizador deve:

1. editar o(s) ficheiro(s) afetado(s) e juntar as modificações
2. registar um novo *commit* de resolução
3. enviar a resolução para o repositório remoto

Exemplo

```
git pull
# CONFLICT (content): Merge conflict in file.txt

<<<<<<< HEAD:file.txt
Hello world
=======
Goodbye
>>>>>>> 77976da35a11db4... :file.txt
```

Entre `<<<<<<` e `=====` é a **modificação local**.

Entre `======` e `>>>>>>` é a **modificação remota**.

Resolver o conflito

Editamos `file.txt` e juntamos as modificações:

```
Hello world
Goodbye
```

Registamos um novo *commit* de resolução:

```
git add file.txt
git commit -m "resolver conflito"
```

Resolver o conflito (cont.)

Por fim, fazemos o `push` da resolução para o repositório remoto.

```
git push
```

Sumário

`clone` copiar um repositório remoto

`add` adicionar ficheiros alterados à àrea de estágio

`commit` registar alterações no repositório local

`push` enviar alterações ao repositório remoto

`pull` pedir alterações do repositório remoto

Recomendações

Alguns cuidados a ter

- Fazer bons *commits*:
  - use o *git add* para juntar apenas as alterações relacionadas
  - escolha *boas mensagens*: o **porquê** do commit, não a quais os ficheiros (desnecessário!)
- Ter em atenção que, se alterarmos a história do repositório de forma descuidada, não estamos apenas a afectar-nos mas também aos nossos colegas de trabalho.

Exemplos de más mensagens

```
git commit -m "Últimas alterações."
git commit -m "Alterações do Pedro."
git commit -m "Adiciona o Jogador.java."
git commit -m "Alterações no Jogador.java."
git commit -m "Adiciona cenas."
git commit -m "Revision"
git commit -m "Blablabla"
git commit -m "WTFWTFWTF"
```

Exemplos de boas mensagens

```
git commit -m "Resolve o bug do prémio"
git commit -m "Remove duplicação de código"
git commit -m "Acrescenta contagem de pontuação"
```

Extras

Mudar nomes

P: Como fazer para mudar o nome de um ficheiro ou diretório que foi registado no *Git*?

R: Usamos `git mv` para mudar o nome de um ficheiro preservando a história de alterações.

```
git mv nome-atual nome-novo
```

Desfazer modificações

P: Editei um ficheiro na área de trabalho, mas agora quero desfazer essas alterações.

R: Usamos `git checkout` para reverter modificações de volta para o estado registado no último *commit*:

```
git checkout -- nome-do-ficheiro
```

Desfazer *stagging*

P: Adicionei um ficheiro à área de estágio, mas afinal não quero incluí-lo no próximo *commit*.

R: Usamos `git reset HEAD` para remover um ficheiro da área de estágio:

```
git reset HEAD nome-do-ficheiro
```

Referências

- Git Community Book
- Pro Git
- Git Reference
- Github
- Atlassian git tutorial
- Git immersion

GUIs para *Git*

- Sourcetree (MacOS, Windows)
- Giggle (Linux)