The Git Kune Do

Pedro Vasconcelos, DCC/FCUP

março 2021

Esta apresentação

- Introdução ao uso do sistema de controlo de versões *Git*
- Direcionado para estudantes de primeiros anos
- Vamos usar:
  - sistema operativo GNU/Linux ou MacOs
  - editor de texto, *shell*, *browser web*

Slides:

```
$ git clone https://github.com/pbv/gitprimer
```

Controlo de Versões

Sistemas de Controlo de Versões (VCS)

Ferramentas para:

- 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 outros programadores
- separar "troncos" de desenvolvimento
  (ex: produção/desenvolvimento)

VCS distribuidos

- Cada cópia dum repositório contém a historia completa
- Permitem registar modificações mesmo sem acesso a rede
- Evitam um ponto crítico para falhas
- Facilitam a introdução de *branches* experimentais

Desvantagens:

- Necessitam de mais espaço em disco
- Utilização pode ser um pouco mais complexa

VCS distribuídos

Git

- Um VCS distribuído
- Desenvolvido em 2005 para o *kernel* Linux
- Muito usado em projetos *open-source*
- Características:
  - conceção simples mas poderosa
  - adequado a projectos grandes (muitos ficheiros e história longa)
  - eficiente em espaço e recursos computacionais
  - suporte para desenvolvimento não-linear ("*branching*")

Porquê usar *Git*?

- Sincronização de trabalhos entre computador pessoal e da universidade
  - diga adeus às *pen drives* ou *Dropbox*
- Permite experimentar modificações sem receios
  - podemos reverter facilmente se necessário
- Repositórios remotos funcionam como *backup*
- As mensagens de *commits* são um registo histórico do desenvolvimento
- Não apenas para código: documentação, relatórios, dissertações

Como funciona

- Cada repositório consiste de um conjunto de ficheiros e diretórios
- Quando registamos uma modificação (*commit*), o *Git* guarda um *snapshot* de todos os ficheiros
- Ficheiros inalterados são guardados como *referências* ao *commit* anterior

Integridade

- O *Git* associa um *hash* (40 carateres hexadecimais) a cada *snapshot*, e.g.:

```
34ac2a6552252987aa493b52f8696cd6d3b00373
```

- Garante que o conteúdo dos ficheiros não foi corrompido
- Serve também para identificar cada *snapshot*

Repositórios locais e remotos

- Quase todas as operações com *Git* são **locais**:
  - inicializar repositórios
  - acrescentar/remover ficheiros
  - registar modificações (*commit*)
  - listar a história

- O *Git* permite também *sincronizar* com repositórios remotos (mais à frente)

Utilização do Git

Linha de comando

Utilizamos o comando `git` para as várias operações:

```
git operação arg1 arg2 ...
```

Exemplo:

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

Configuração inicial

```
$ git config --global user.name "My name"
$ git config --global user.email my@email.domain
```

Criar um novo repositório

```
$ mkdir my-project
$ cd my-project
$ git init
```

- Cria um diretório `my-project/.git` para meta-dados
- O repositório está inicialmente *vazio*
- Devemos depois adicionar ficheiros e/ou sub-diretórios

Adicionar ficheiros

```
# editar os ficheiros...
$ git add src/foo.c
$ git add src/bar.h
$ git add README.txt
```

- Podemos adicionar vários ficheiros de uma só vez:

```
$ git add src/foo.c src/bar.h README.txt
```

- Os ficheiros ficam na *área de estágio*
- Temos de efetuar *commit* para os registar no *Git*

Primeiro *commit*

```
$ git commit -m "initialized repository"
```

- Todos os *commits* têm associada uma mensagem

- Se omitirmos a opção `-m`, o *Git* abre um editor de texto para compor a mensagem

Modificar ou acrescentar

Depois de modificar/criar alguns ficheiros:

1. adicionamos os ficheiros modificados à àrea de estágio
2. registamos um novo *commit*

```
#  editar / criar ficheiros ...
$ git add README.txt LICENSE.txt
$ git commit -m "modified and created files"
```

Modificar ou acrescentar (2)

Podemos optar por registar as modificações como *commits* separados:

```
$ git add README.txt
$ git commit -m "modified file"
$ git add LICENSE.txt
$ git commit -m "created file"
```

Estados de um ficheiro

Committed	guardados na base de dados local
Modified	modificados em relação à versão guardada
Staged	marcados para entrar no próximo *commit*

Consultar o estado do repositório

```
$ git status
```

`Changes to be commited` modificações **que serão incluidas** no próximo *commit*

`Changes not staged for commit` ficheiros modificados mas ainda **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
$ git log   # listar o histórico de commits
```

Exemplos

```
$ git diff
$ git diff src/foo.c
$ git log --oneline
$ git log --since=01/04/2017 --author="Pedro"
```

(Use `--help` para obter ajuda completa.)

Checkout

O *Git* permite "viajar no tempo" de desenvolvimento do projeto.

Usando `git checkout` podemos reverter o diretório de trabalho para *snapshots* específicos.

Exemplo

Listar todos os *snapshots* (mais recente primeiro):

```
$ git log --all --oneline
7fd2d99 (HEAD -> main, ...) last commit
7cf2ce7 second commit
432bffa first commit

# reverter ao primeiro commit
$ git checkout 432bffa
# avançar até ao último commit
$ git checkout 7fd2d99
# alternativa
$ git checkout main
```

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 repositório *Git* num outro computador!

Github e Gitlab

- Serviços de *hosting* para repositórios Git
- Populares para *software open-source*
- Repositórios *públicos* ou *privados*
- Permitem criar contas gratuitas
- Contas profissionais para estudantes/professores

https://github.com/

https://gitlab.com/

Clone — copiar um repositório remoto

```
$ git clone <url-remoto>
```

- Accesso por HTTPS ou SSH
- Obtemos uma cópia local que podemos editar livremente

Exemplo (esta apresentação):

```
$ git clone https://github.com/pbv/gitprimer
```

Commit — regitar modificações

Primeiro registamos *commits* no repositório local:

```
# editar README src/foo.c
$ git add README src/foo.c
$ git commit
```

O *commit* é **local** — nada foi enviado para o repositório remoto.

Push — enviar modificações

Usamos o comando *push* para enviar *commits* locais para o repositório remoto que lhe está associado.

```
$ git push
```

Envia todos os *commits* que fizemos no repositório local e ainda não existem no repositório remoto.

Pull — receber modificações

Usamos *pull* para pedir *commits* ao repositório remoto:

```
$ git pull
```

Descarrega e aplica todos os *commits* que existem no repositório remoto e não no repositório local (por exemplo: de outros colaboradores).

Permissões e colaboração

- Podemos *ler* qualquer repositório público
- Mas só podemos submeter *commits* se tivermos permissão de *escrita*
  - ex: repositórios nossos ou da nossa equipa

Colaboração em trabalhos ou projetos

- Criamos um repositório *privado* num servidor *GitHub* ou *GitLab*
- Damos acesso de leitura/escrita aos membros do grupo
- O repositório é usado como *ponto de sincronização* entre os colaboradores
- Começe usando apenas um *branch* (por omissão: *main*)
- Se tiver mais experiência poderá introduzir *branches* separados

Merge — juntar ramos de desenvolvimento

- Um *merge* junta dois ramos desenvolvimento divergentes
- O *Git* tenta fazer *merge* automático quando executamos `push` ou `pull`
- Em caso de **conflitos** o *merge* tem de resolvido manualmente pelo programador

Conflitos

Se dois *commits* separados modificarem um mesmo ficheiro o *Git* pode sinalizar um **conflito**.

O conflito é detetado quando tentarmos sincronizar com um repositório remoto (`pull` ou `push`).

Resolver conflitos

1. Editar os ficheiros afetados e juntar as alterações
2. Registar um novo *commit* de resolução
3. Efetuar `push` 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 (1)

Editamos `file.txt` e juntamos as duas modificações:

```
Hello world
Goodbye
```

Resolver o conflito (2)

Registamos um *commit* de resolução:

```
$ git add file.txt
$ git commit -m "resolução de conflito"
```

Resolver o conflito (3)

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

```
$ git push
```

Recomendações

Escolher *commits*

- Use `git add` efetuar *commits* coesos
  - não junte as modificações todas num só *commit*
  - agrupe modificações que fazem sentido em conjunto
- Tente compor boas mensagens:
  - não dizer quais os ficheiros alterados
  - explicar o **sentido** das alterações

Exemplos

Evitar

```
$ git commit -m "últimas alterações"
$ git commit -m "alterações do Pedro"
```

Melhor

```
$ git commit -m "resolve o bug da tabela"
$ git commit -m "remove duplicação de código"
$ git commit -m "geração de código para ciclos"
```

Mudar nomes

Como fazer para mudar o nome de um ficheiro ou diretório?

Solução

```
$ git mv <nome-atual> <nome-novo>
```

Desfazer modificações

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

Solução

```
$ git checkout -- <ficheiro>
```

(Reverte modificações para o estado registado no último *commit*.)

Desfazer *stagging*

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

Solução

```
$ git reset HEAD <ficheiro>
```

Utilizar SSH com GitHub

1. Gerar uma chave SSH
2. Adicionar a chave ao `ssh-agent` local
3. Adicionar a chave à sua conta Github

https://docs.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Sumário dos comandos

`init`	inicializar um repositório
`clone`	copiar um repositório remoto
`add`	adicionar ficheiros à àrea de estágio
`commit`	registar alterações no repositório local
`checkout`	reverter para um *snapshot* específico
`push`	enviar alterações ao repositório remoto
`pull`	puxar alterações do repositório remoto

Mais informação

- Git Community Book
- Atlassian git tutorial
- Git immersion

Em caso de desespero ;-)

https://xkcd.com/1597/