Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

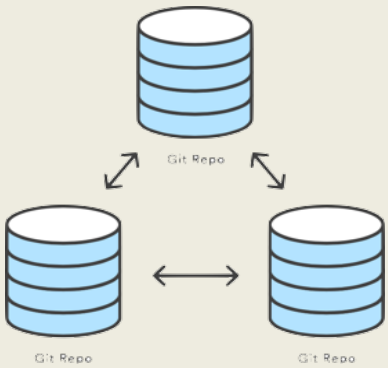
2019/2020

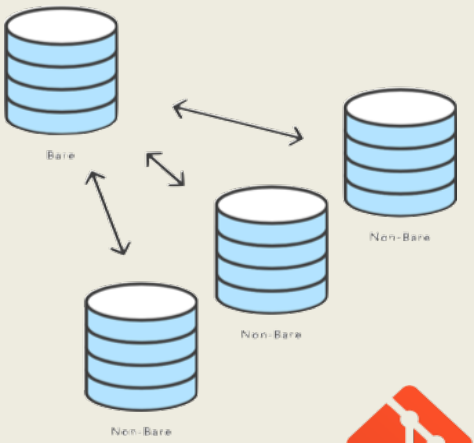
Construção de Sistemas de Software

Controlo de Versões com o GIT

Git

- Um sistema de controlo de versões distribuído
  - permite a colaboração *repo-to-repo*
  - um caso particular, é quando existe um repositório central





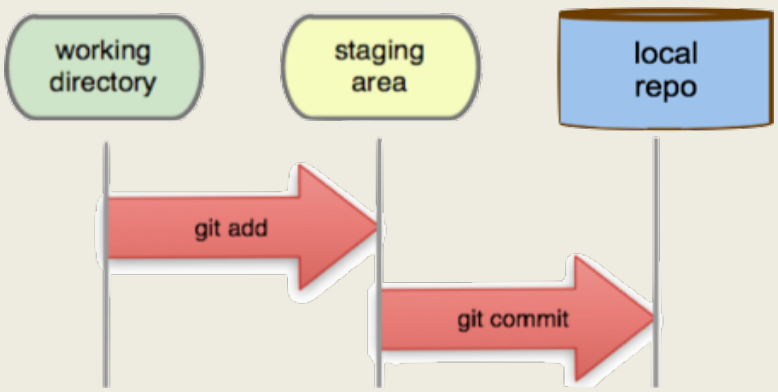
https://git-scm.com

Repositório Local

- Mesmo quando se trabalha sozinho num projeto (sempre na mesma na máquina), pode ser importante registar a evolução do nosso trabalho, ou seja, registar as diferentes versões que vão existindo ao longo do tempo
  - para ter possibilidade de recuperar uma versão anterior
  - para saber quando se fez uma determinada alteração e porquê
  - para ter as alterações organizadas de forma a ser fácil revertê-las
  - ...
- Neste caso basta criar um **Repositório Local**

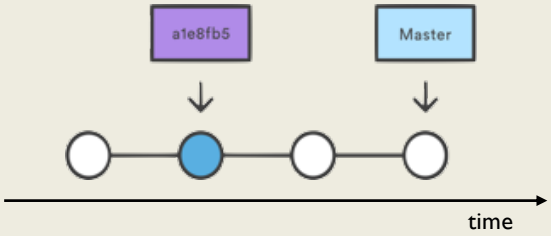
Repositório Local

- Todo o trabalho é realizado na ***working directory*** (designada também por ***working tree***)
- Para que as mudanças realizadas fiquem registadas é necessário
  - adicionar ficheiros mudados à ***staging area*** (ou ***index***)
  - executar a operação `commit`, fornecendo texto explicativo



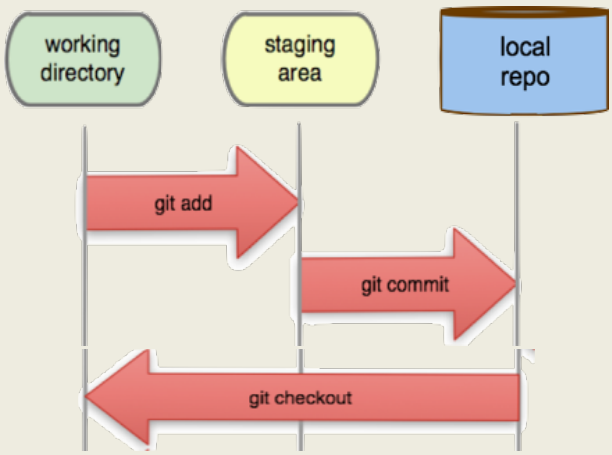
Repositório Local

- Sempre que é executada a operação *commit,* é adicionado ao repositório um novo objecto, chamado ***commit object,*** que
  - tem o "estado" do sistema de ficheiros, a data, o autor e a mensagem do *commit*
  - pode ser referenciado através do seu nome SHA1, um código de *hash* com 40 caracteres (normalmente basta usar os primeiros 7, `a1e8fb5`)
  - tem como pai o *commit object* original, i.e., aquele sobre o qual foram feitas as mudanças
- Assim, o repositório é basicamente um conjunto de *commit objects* que podem ser visualizados como um grafo conexo (DAG)



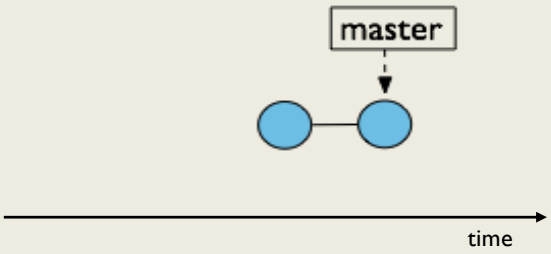
Repositório Local

- Para colocar na *working directory* a versão guardada num determinado *commit object* basta executar a operação **checkout**



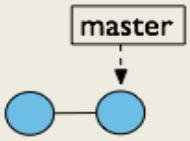
Branching & Merging

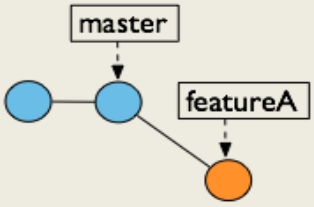
- Os ramos (***branches***) possibilitam o desenvolvimento em paralelo a partir de um dado ponto, sem afetar o ramo pai.
- Cada *branch* é representado por uma ***head*** e vice-versa.
- Quando é criado um repositório é criado por omissão o ramo ***master***.



Branching & Merging

- A operação **branch** permite criar um ramo com um determinado nome a partir de um *commit object* no repositório.

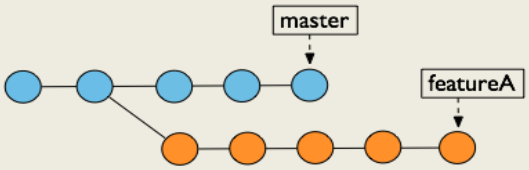




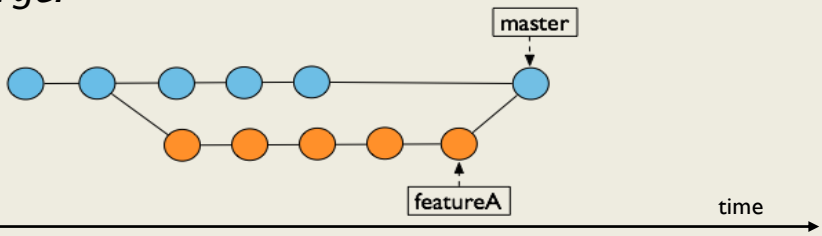
- Recorrendo à operação **checkout** podemos trocar para esse ramo e trabalhar sobre ele, juntando mais *commits*, sem afetar o ramo pai.

Branching & Merging

- Fazendo **`checkout`** do ramo master podemos também ir trabalhando sobre esse ramo sem afetar o ramo novo.



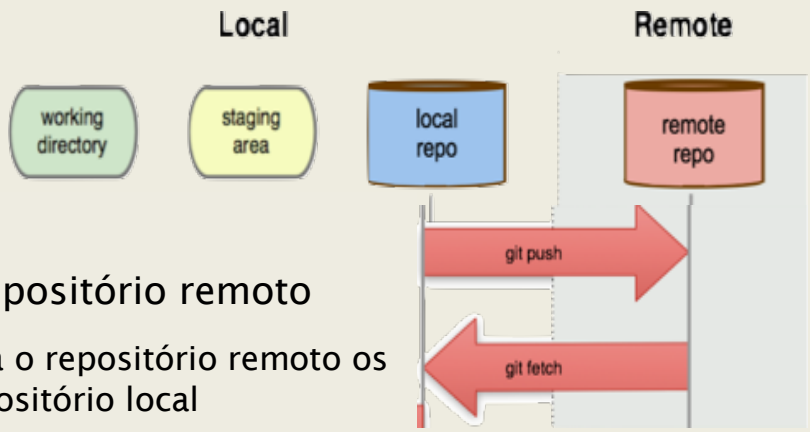
- A operação **`merge`** permite integrar no ramo em que estamos a trabalhar as alterações entretanto efetuadas num ramo especificado. Se não houver conflitos é criado um novo *commit object* com o resultado do *merge.*



Repositório Local + Repositório Remoto

- Quando o trabalho é realizado em equipa (ou se trabalha sozinho mas se usam diferentes máquinas) é mais complicado, pois as mudanças são realizadas de uma forma distribuída
- Neste contexto é necessário que haja a colaboração entre diferentes repositórios (*repo-to-repo collaboration*)
  - **Clonar** um repositório remoto
    - A operação **`clone`** copia um repositório existente com toda a sua história
    - Útil por exemplo para quem vai começar a trabalhar num projeto em andamento
  - **Sincronização** com um repositório remoto
    - com operações **`push`** e **`fetch`** e **`merge`** e **`pull`**

Repositório Local + Repositório Remoto



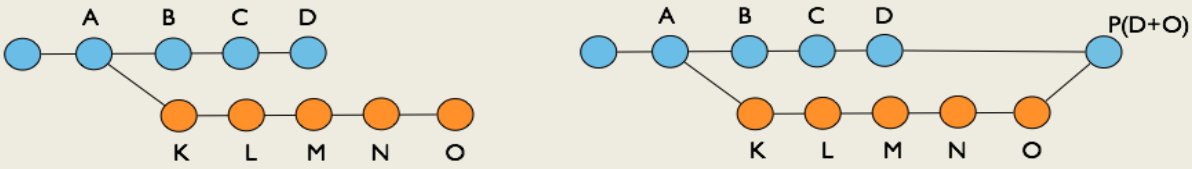
Sincronização com um repositório remoto

- A operação **`push`** leva para o repositório remoto os *commits* que estão no repositório local
  - só para repositórios *bare*, que não têm *working directory* (servidor)
  - é como se o *remote* fizesse *merge* com o local
- A operação **`fetch`** importa *commits* do repositório remoto para o local

Repositório Local + Repositório Remoto

Sincronização com um repositório remoto

- ...
- A operação **`merge`** junta os *commits* realizados em cada um dos repositórios para um dado ramo
  - se houver mudanças conflituantes têm de ser resolvidas
  - se não houver conflitos é criado um novo *commit object* e o resultado é também colocado na *working directory*



- A operação **`pull`** realiza um *fetch* seguido de um *merge* (ou um *rebase*)

Workflow recomendado para trabalho em grupo (GitLab)

Para o trabalho individual de cada elemento dentro de um grupo que

- tem em mãos um projetos com alguma complexidade
- com disciplina suficiente para trabalhar de forma organizada e levar a cabo revisões de código

recomenda-se o seguinte fluxo de trabalho:

1. Fazer *pull* do ramo *master* para o atualizar com as mudanças mais recentes na origem (*remote*).
2. Criar um novo ramo XYZ para fazer o trabalho individual, seja ele o desenvolvimento de uma nova *feature* ou *refactoring* ou outra coisa qualquer
3. Quando a tarefa estiver pronta, fazer *merge* de XYZ com o ramo *master*:
   - Atualizar o seu ramo *master* com as mudanças que existam na origem (*remote*) usando o *pull*.
   - Fazer *merge* das mudanças que existam no master para o ramo XYZ e resolver os conflitos que existam.
   - Fazer *push* do ramo XYZ para a origem de forma a torná-lo visível.

Workflow recomendado para trabalho em grupo (GitLab)

4. No GitLab, debaixo do repositório do projeto, usar o botão para criar um *pull/merge request* a partir do *feature branch. Os pedidos de merge* são uma forma de organizar as mudanças que ainda não foram aprovadas e juntas ao ramo *master*.
5. Atribuir a responsabilidade de rever o código e dar feedback a um dos outros membros do grupo.
6. Pode continuar a melhorar o código até ele ser revisto, aprovado e ser feito o seu *merge* com o repositório:
   - Fazer as mudanças necessárias na versão local do ramo XYZ.
   - Fazer *commit* das mudanças e fazer *push* para a origem.

   No GitLab, o ramo é removido do servidor quando é feito o *merge*.

Ciências ULisboa | Informática

Git Cheat Sheet

REBELLABS by ZEROTURNAROUND

Create a Repository

From scratch -- Create a new local repository
```
$ git init [project name]
```

Download from an existing repository
```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed
```
$ git status
```

Show the changes to files not yet staged
```
$ git diff
```

Show the changes to staged files
```
$ git diff --cached
```

Show all staged and unstaged file changes
```
$ git diff HEAD
```

Show the changes between two commit ids
```
$ git diff commit1 commit2
```

List the change dates and authors for a file
```
$ git blame [file]
```

Show the file changes for a commit id and/or file
```
$ git show [commit]:[file]
```

Show full change history
```
$ git log
```

Show change history for file/directory including diffs
```
$ git log -p [file/directory]
```

Working with Branches

List all local branches
```
$ git branch
```

List all branches, local and remote
```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory
```
$ git checkout my_branch
```

Create a new branch called new_branch
```
$ git branch new_branch
```

Delete the branch called my_branch
```
$ git branch -d my_branch
```

Merge branch_a into branch_b
```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit
```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit
```
$ git add [file]
```

Stage all changed files, ready for commit
```
$ git add .
```

Commit all staged files to versioned history
```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history
```
$git commit -am "commit message"
```

Unstages file, keeping the file changes
```
$ git reset [file]
```

Revert everything to the last commit
```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)
```
$ git fetch
```

Fetch the latest changes from origin and merge
```
$ git pull
```

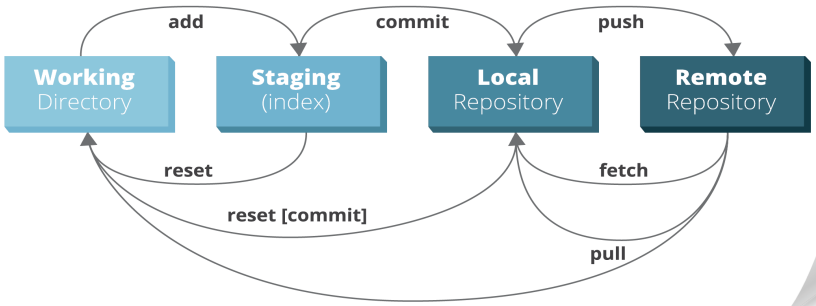
Fetch the latest changes from origin and rebase
```
$ git pull --rebase
```

Push local changes to the origin
```
$ git push
```

Finally!

When in doubt, use git help
```
$ git command --help
```

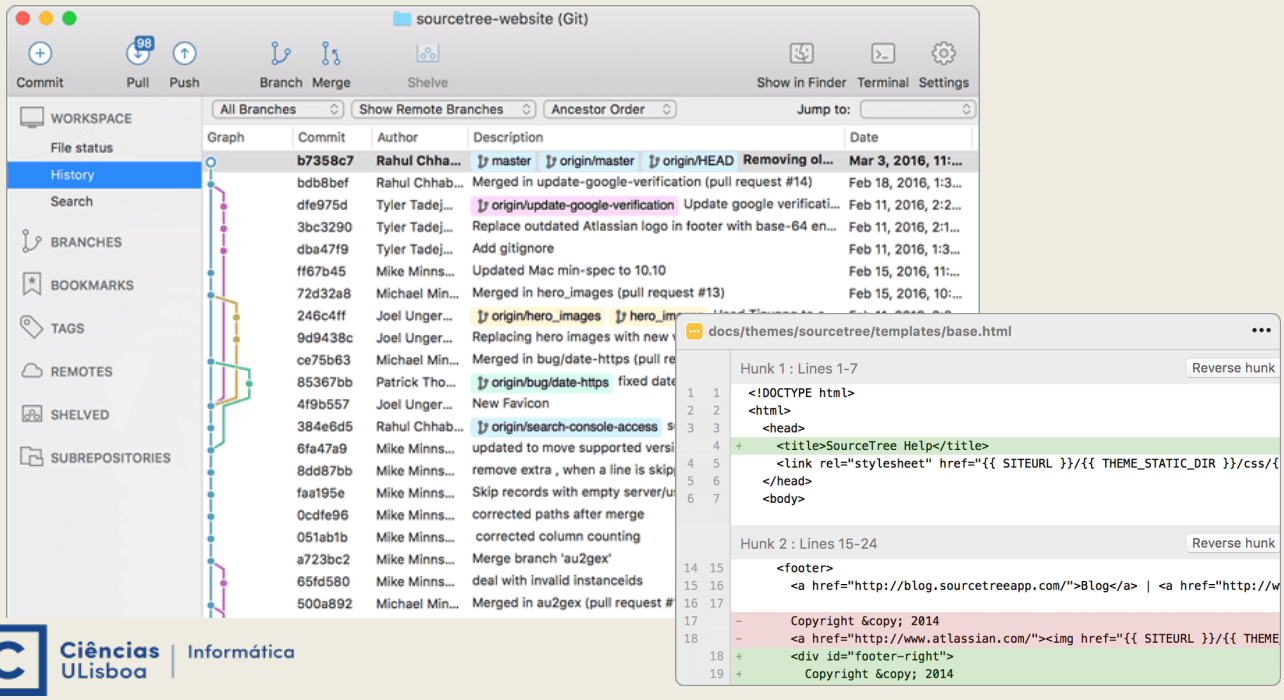
Or visit https://training.github.com/ for official GitHub training.



BROUGHT TO YOU BY JRebel

Ferramentas

- Linha de comandos
- *SourceTree* (mac e windows)



Ciências ULisboa | Informática

Ferramentas

- Linha de comandos
- *SourceTree* (mac e windows)
- Eclipse