Computação na Nuvem

Sistemas para controlo de versões

Version control tools

GIT basics + branching + intro to remotes

Sistemas para controlo de versões?

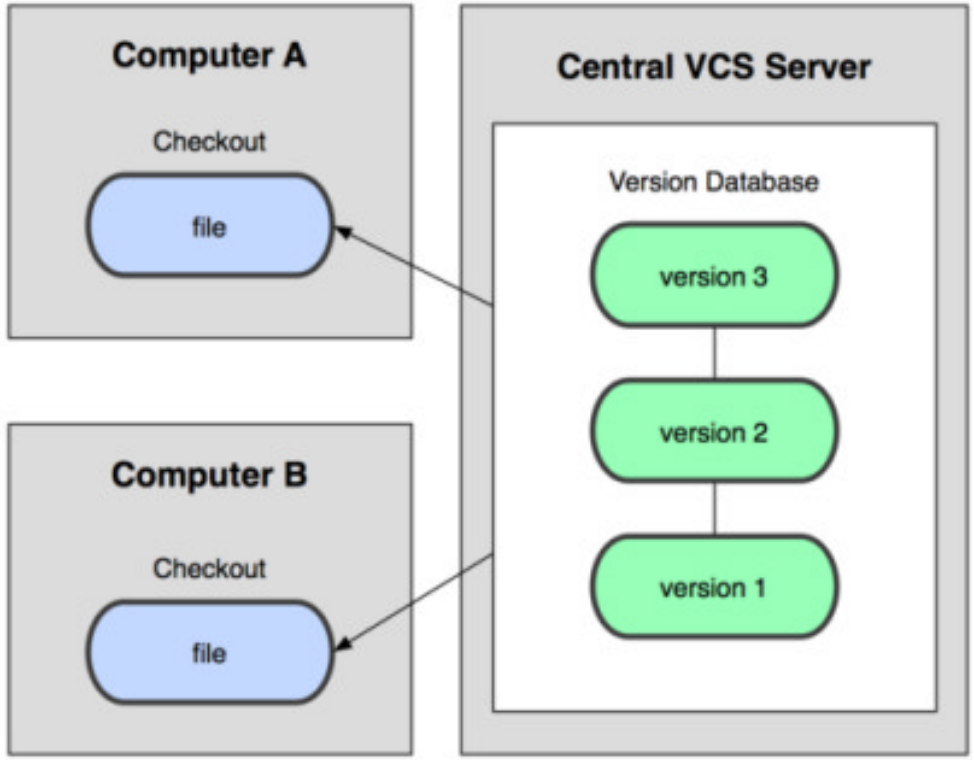
- Sistemas que registam modificações feitas a ficheiros, ao longo do tempo, para efeitos de
  - Permitir recuperar versões prévias
  - Permitir um histórico da atividade
  - Permitir entender que mudanças e porque motivo, foram acontecendo
  - Facilitar suporte a trabalhos derivados
- No original
  - VCS
    - Version Control Systems

VCSs locais e não colaborativos

- Tirando partido do sistemas de ficheiros
  - Copiar os documentos para uma pasta, idealmente nomeada a partir da data
    - Simples
    - Operação dada a erros
- Base de dados local (das modificações feitas aos ficheiros controlados)
  - RCS
    - BSD e variantes, incluindo MAC OS, ainda incluem RCS numa package de “developer tools”

VCSs colaborativos : CVCSs

- E se no projeto participarem diversas pessoas?
- CVCS
  - Centralized Version Control Systems
  - Exemplos
    - CVS
    - Subversion
    - Perforce
- Servidor
  - Concentra o repositório
- Clientes
  - Fazem checkout dos documentos nos quais pretendem trabalhar

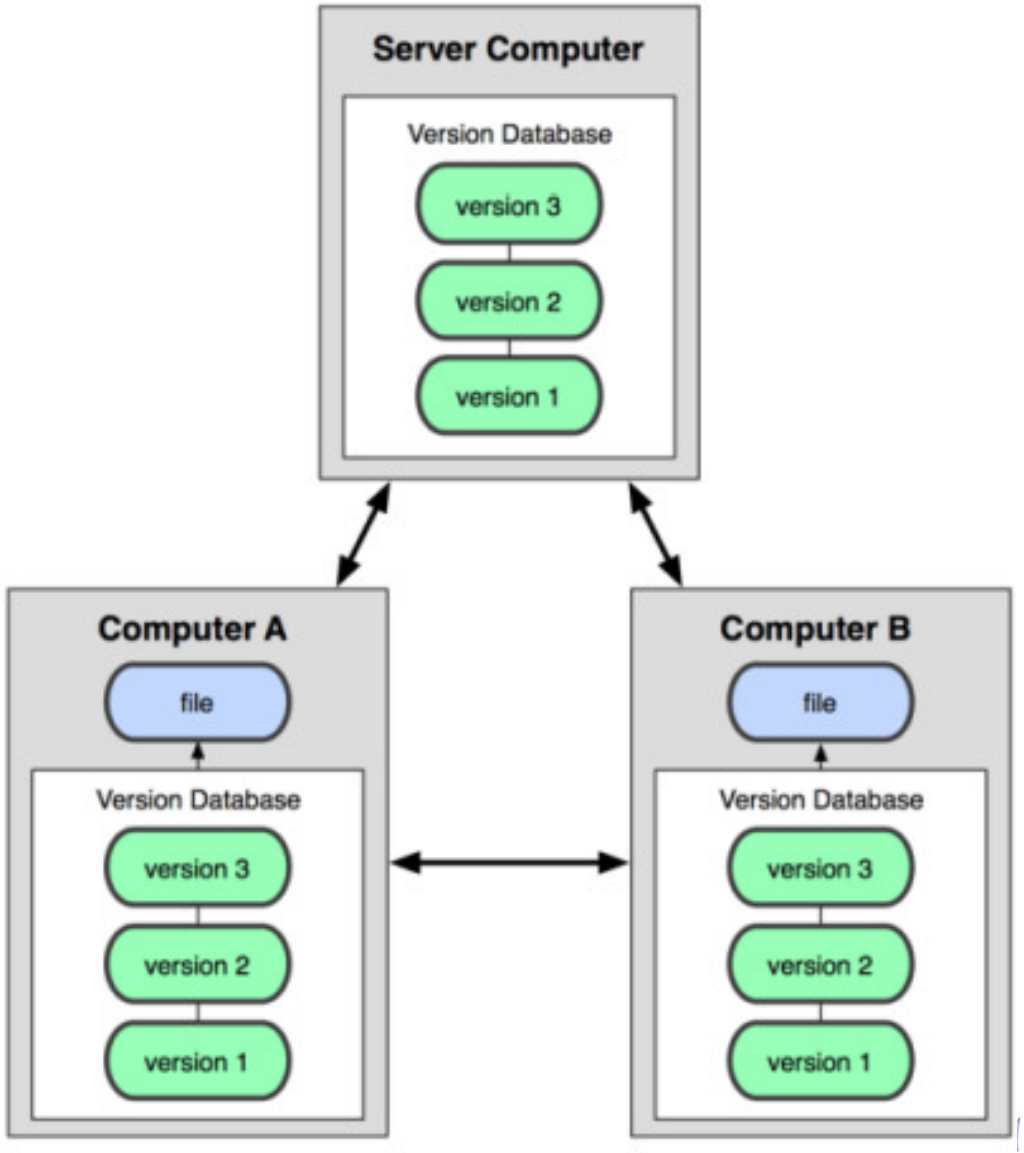


VCSs colaborativos : CVCSs

- Vantagens
  - Facilidade de administração
  - Líderes do projeto podem ter uma visão global do trabalho que anda a ser feito
- Desvantagens
  - Vulnerabilidade “todos os ovos num só cesto”

VCSs colaborativos : DVCSs

- DVCSs
  - Distributed Version Control Systems
  - Exemplos:
    - Git
    - Mercurial
    - Bazaar
    - Darcs
  - Servidor & clientes
    - Mas os clientes replicam o conteúdo do servidor
      - Redundância
      - Resiliência



DVCS : Git

- Git
  - Desde 2005
  - Sucede a “BitKeeper” para controlo de versões do Linux Kernel
  - Regista “snapshots” dos ficheiros sob seu controlo
    - Exclui ficheiros não modificados
    - Diferença: a generalidade dos VCSs regista sucessões de modificações aos ficheiros (deltas)
  - Quase todas as operações são locais
    - Menos afetado pela disponibilidade/qualidade da rede do que qualquer outro VCS
  - Content Tracking System?
  - Pack file storage?

Git : situações para os ficheiros

- Untracked
  - Presente na working directory, não staged
- Unstaged/modified
  - Houve modificação num ficheiro não staged
    - O ficheiro pode ficar staged pelo comando “add”
- Staged/modified
  - Quando um ficheiro já no stage foi detetado modificado
    - A atualização na BD faz-se pelo comando “commit”
- Commited
  - Ficheiro staged fica atualizado na BD
    - A BD também se diz “repositório” e está no folder .git nas pastas sob controlo

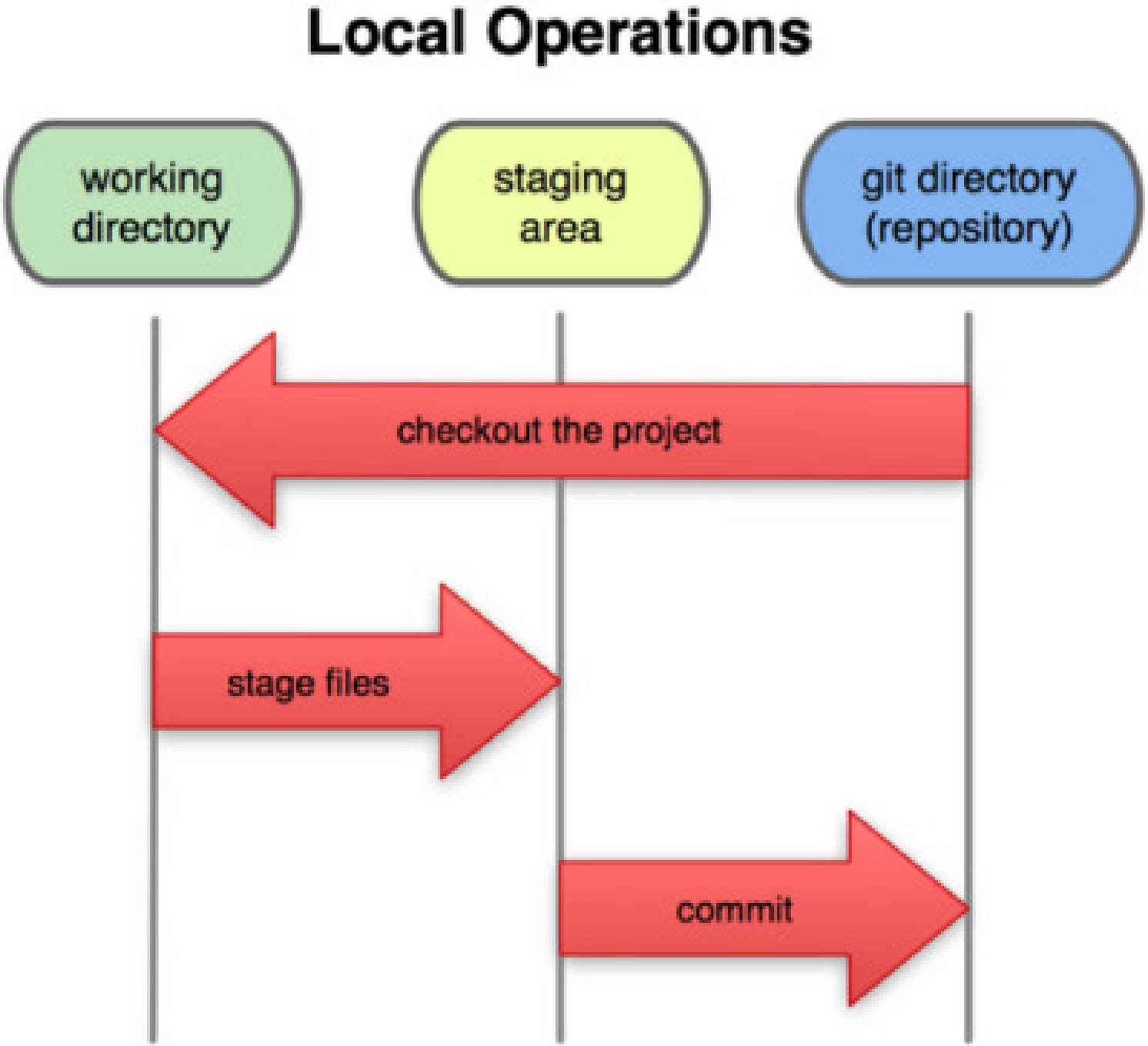
Git : 3 áreas principais num projeto Git

- Working directory (WD) / folder de trabalho
  - Contém os ficheiros nos quais se está a trabalhar
  - Tipicamente pretende-se guardar “snapshots” seus na base de dados, sendo necessário o passo prévio de metê-los no stage
- Index / stage / staging area
  - Um ficheiro transita da WD para o stage, via add
    - Git add ficheiro
  - Corresponde aos ficheiros/conteúdos que serão escritos na base de dados .git, na próxima instrução de commit
  - Um ficheiro no stage, ainda não commited, pode regressar à working directory via checkout
    - Git checkout ficheiro
- .git folder
  - Toda a meta informação e objetos que representam o projeto
  - Na base de dados o Git guarda tudo não por nome mas por sha1 do conteúdo
    - sha1(conteúdo) = stringHexaDe40Símbolos

Git workflow, em 3 passos

- #1
  - Ao trabalhar-se no folder de trabalho, criam-se, modificam-se ficheiros
- #2
  - Os ficheiros criados, modificados podem ir para o index ou staging area, desde que sujeitos ao comando add
- #3
  - Ficheiros que tenham sido added ao index ou staging area, passarão à base de dados .git, perante o comando commit

Git workflow, em 3 passos



Git : instalação em Windows

- Info:
  - https://git-scm.com/
  - http://msysgit.github.com/
- Download
  - http://code.google.com/p/msysgit/downloads/list?q=full+installer+official+git
- Repositório
  - http://github.com/msysgit/git

Github.com

- Diferente de Git
- Abrir uma conta
- Criar um repo (canto superior direito da interface)
- Aceitar as opções por defeito
- Depois, para refletir um repo local em Github.com:
  - No cliente, usar o nome “convencional” origin:
    - git remote add origin https://github.com/amsm/test1.git
  - Ou personalizar o nome do repo remote
    - git remote add github_test1 https://github.com/amsm/test1.git
  - Seguido de um push (implica escrever user/pass)
    - git push -u github_test1 master

Git config : depois da instalação

- Configuração Git
  - Os aspetos mais gerais de funcionamento do Git
  - Como?
    - Git config
- Onde está guardada a configuração?
  - No ficheiro de texto @

    %HOMEPATH%\.gitconfig

    e.g. C:\users\<user name>\.gitconfig

Git config : nome, e-mail, etc

- Todos os commits identificam nome e e-mail do colaborador que os executa
- Estabelecer nome
  - Git config --global user.name “Art”
- Estabelecer email
  - Git config --global user.email “art@site.com”
- Ver configuração
  - Git config --list
- Pedir ajuda
  - Git help config

Git config : nome, e-mail, etc

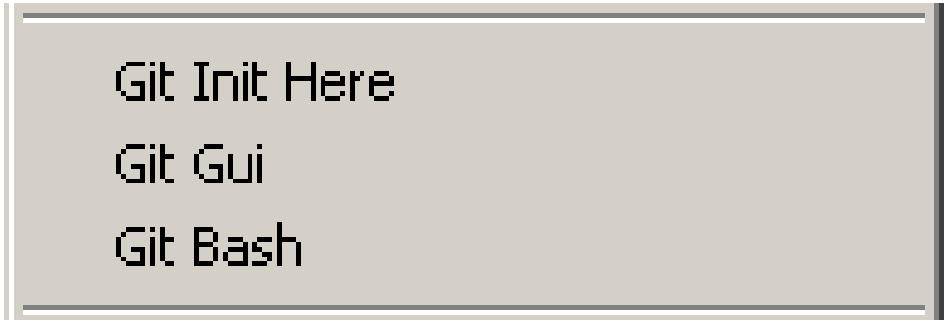
- Configurações específicas de um repositório, que fazem override das configurações globais:
- Estabelecer nome
  - Git config user.name “Art”
- Estabelecer email
  - Git config user.email art@site.com
- Remover configurações:
  - Git config --unset --global user.email
  - Git config --unset user.email

Git config : editor

- O editor por defeito é o VIM
- Para indicar outro
  - git config --global core.editor
- Exemplo para o NotePad++
  - http://notepad-plus-plus.org
  - git config --global core.editor "'*<path absoluta até ao EXE do editor>*' -multiInst -nosession"
  - git config --global core.editor "'C:\wp\util\notepp\notepad++.exe' -multiInst -nosession"

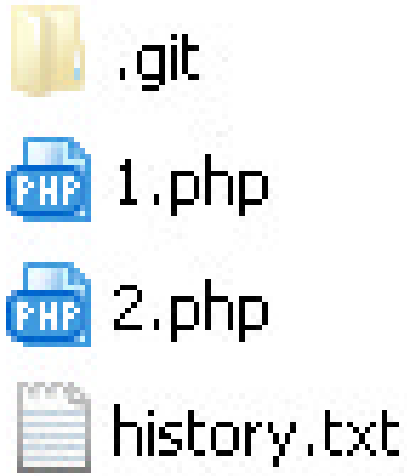
Git : utilização elementar (1)

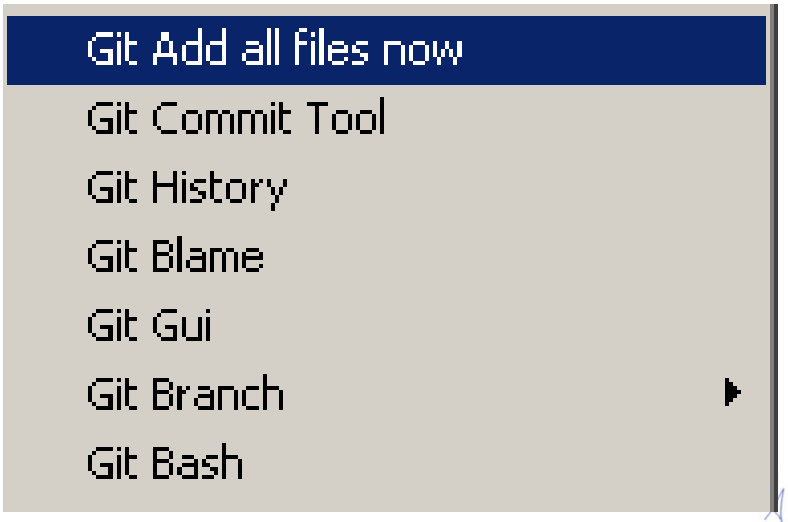
- Criação do repositório
  - Command line
    - cd <folder que se quer sob controlo>
    - <path instalação do git>\bin\git init
  - GUI
    - Right click sobre a pasta a controlar
    - Escolher “git init here”



Git : utilização elementar (2)

- Controlar alguns ficheiros
  - Command line
    - Git add *.php
    - Git add history.txt
    - Git commit -m “version 1”
  - GUI
    - Operações similares ou idênticas são possíveis via
      - Right-click sobre a pasta controlada, e depois...
        - » Git add all files now
        - » Git GUI





Git : utilização elementar (3)

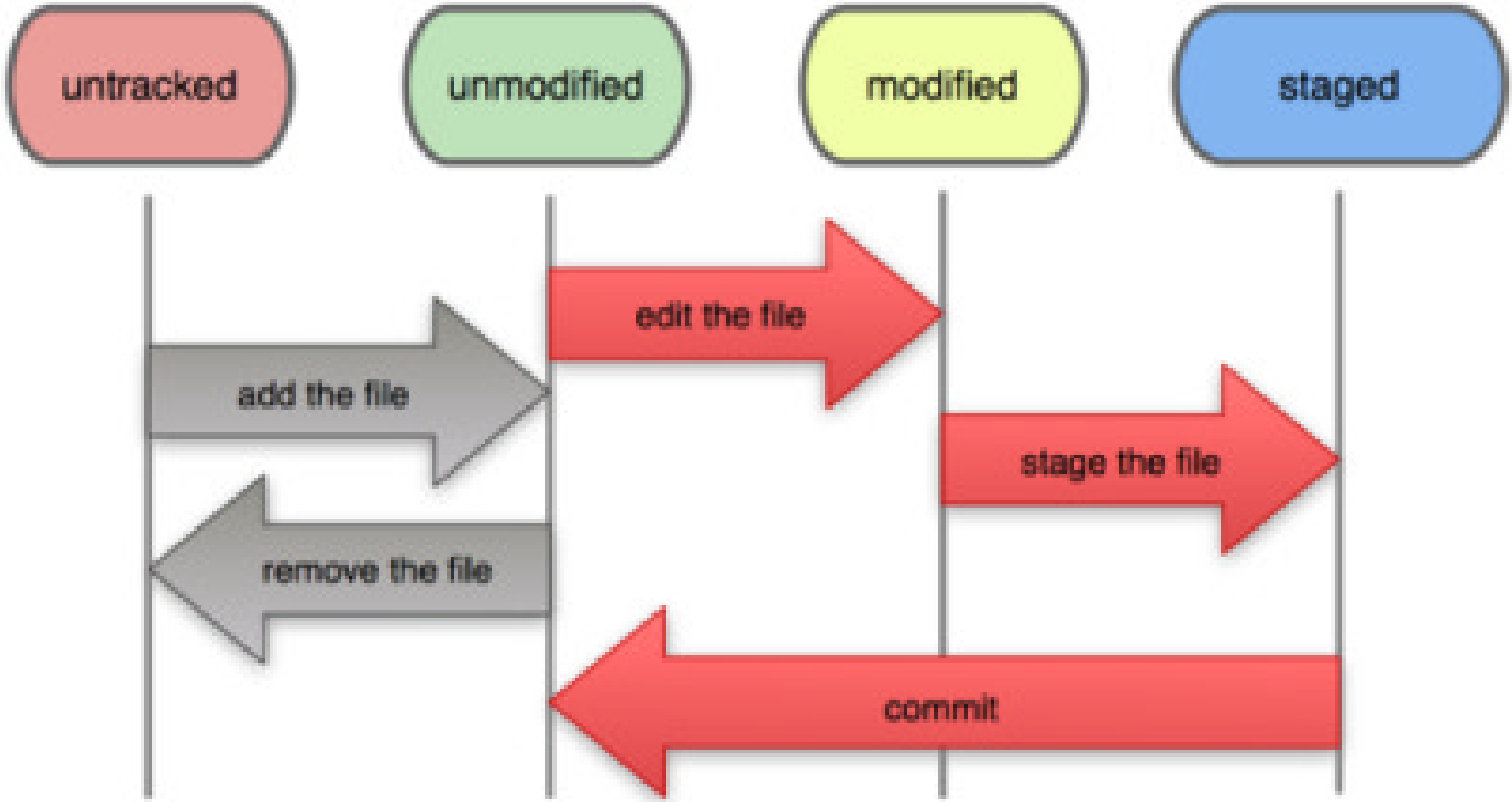
- Clonar um repositório
- Command line
  - Git clone url-da-origem
    - Exemplo:
      - Git clone d:\projeto
      - Git clone git://github.com/some-project/file.git <pasta-de-destino>

Git : utilização elementar (4)

- Estado dos ficheiros
  - Tracked (3 sub estados)
    - Estavam na última snapshot
      - Unmodified / não modificados
      - Modified / modificados
      - Staged / modificados e acrescentados ao index
  - Untracked
    - Não estavam na snapshot anterior
    - Não foram added
- Os comandos git são normalmente recursivos
  - E.g. Git add <pastas>
    - Aplica-se a todos os ficheiros de todas as sub-pastas

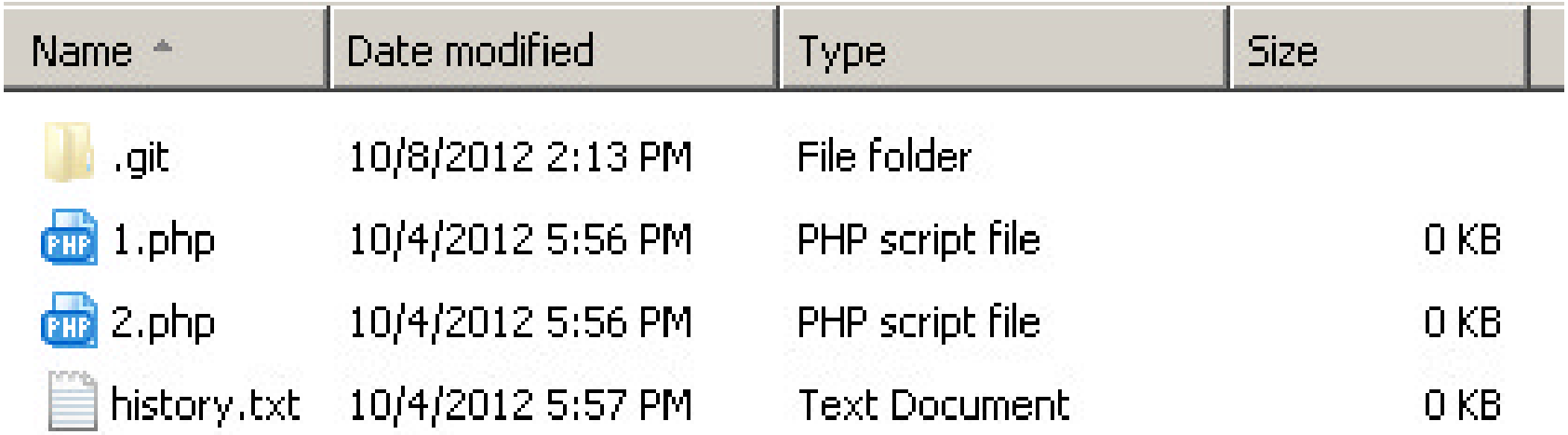
Git : utilização elementar (5)

- Ciclo de vida dos ficheiros sob Git controlo
- Atitude
  - Fazer add dos ficheiros a acompanhar
  - Fazer commit dos ficheiros num estado digno de registo



Git : utilização elementar (6)

- Para saber do estado dos ficheiros
  - Git status
  - Exemplo, havendo 3 ficheiros vazios, antes commited, numa pasta “jungle2”.



Name	Date modified	Type	Size
.git	10/8/2012 2:13 PM	File folder	
1.php	10/4/2012 5:56 PM	PHP script file	0 KB
2.php	10/4/2012 5:56 PM	PHP script file	0 KB
history.txt	10/4/2012 5:57 PM	Text Document	0 KB

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Git : utilização elementar (7)

- Git status : para saber do estado dos ficheiros
- Exercício:
  - Ao exemplo anterior, acrescentar um ficheiro vazio novo.txt
  - Notar que o ficheiro é detetado como “untracked”
  - Para fazer track, fazer git add : o ficheiro aparecerá como “new”

```
admin@AS5333 /I/jungle2 (master)
$ git add novo.txt

admin@AS5333 /I/jungle2 (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   novo.txt
#
```

Git : utilização elementar (8)

- Git status : para saber do estado dos ficheiros
- Exercício: acrescentar algum conteúdo ao ficheiro “novo.txt” e pedir git status – o ficheiro estará assinalado como new e modified
  - Simultaneamente “staged” e “unstaged” : cautela!
    - Commit submete apenas o último add

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   novo.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   novo.txt
```

Git : utilização elementar (9)

- Git : ignorar ficheiros
  - Criar ficheiro .gitignore
  - Ficheiro de texto que lista que ficheiros devem ser ignorados, como se não existissem na pasta de trabalho
  - Exº:

*.exe #ignorar ficheiros .exe

!principal.exe #considerar este, apesar da regra anterior

/FICH #ignorar o ficheiro FICH, mas não a pasta FICH

Lixo/ #ignorar tudo na pasta lixo

Old/*.doc #ignorar todos os .doc na pasta old

- Ficheiros já tracked nunca serão ignored
  - Cautela!
  - .gitignore não tem efeitos retroativos

Git : utilização elementar (10)

- Sem argumentos, git diff serve para ver as diferenças entre a diretoria de trabalho e o index/stage (que não é exatamente a mesma coisa que entre o index/stage e a dir)
- Git diff
  - Não havendo diferenças, o output é vazio
  - Havendo diferenças, são listadas.
    - Formato “unified diff format”
    - Símbolos especiais são assinalados com notação própria,
      - ^M para “carriage return”
  - Notar que são diferenças da dir em relação ao index/stage
    - Ou seja, entre a diretoria de trabalho e o último add
  - Para diferenças entre a dir e a BD, @último commit
    - Git diff head
- git diff --cached
  - Diferenças entre index/stage e BD, último commit
  - Pensar em “cached” como “staged”
- Forma geral com argumentos: git diff sha1-1 sha1-2

Git : utilização elementar (11)

- Para cristalizar mudanças
  - Git commit –m “descrição do que mudou”
  - Git commit --message “descrição”
  - Git commit –m “desc” --author “Jon <jon@gmail.com>"
- Recordar que só seguem as versões added dos ficheiros, não necessariamente as últimas!
- Ler os resultados de um commit
  - Relativamente ao snapshot anterior
    - [branch-name sha1]
    - #ficheiros modificados
    - #linhas inseridas
    - #linhas removidas

```
$ git commit -m "v2"
[master c3c9766] v2
 1 file changed, 14 insertions(+), 1 deletion(-)
```

Git : utilização elementar (12)

- Para fazer um commit das últimas versões de ficheiros, mesmo que não tenham sido added (mas já estejam tracked devido a comandos de add anteriores), fazendo bypass da staging area, deve usar-se a opção –a
  - "all modified" (mas têm que estar staged)
  - Git commit –a –m "v3"

  ou
  - Git commit --all –m "v3"
- Imediatamente após o commit pode associar-se-lhe uma tag
  - Git tag <nome>
  - Git tag –d <nome> apagaria a tag
- Para associar a tag a um commit específico, não necessariamente o último
  - Git tag <nome> <sha1> (7 primeiros digitos bastam)
- Para remover um ficheiro do projeto e/ou do Git
  - (1) Apagá-lo só da diretoria de trabalho
    - Rm ficheiro
  - (2) Apagá-lo do Git stage e da pasta
    - Git rm ficheiro
      - Git rm só apaga ficheiros tracked
        - » Se o ficheiro tiver sido modificado após add, poderá ser necessário
          - Git rm –f ficheiro
  - (3) Commit das mudanças
    - Git commit –m "ficheiro apagado"

Git : utilização elementar (13)

- Rename?
  - Mv file1 file2
  - Git rm file1
  - Git add file2

  Ou
  - Git mv file1 file2

  Seguidos do commit
  - Git commit –m “renamed file1 to file2”

Git : undos (1)

- Se se fez um commit errado, por exº na mensagem, por exº nos ficheiros staged, etc, *e* se detetou isso imediatamente a seguir, para refazer basta
  - Git commit --amend -m “nova mensagem”
    - Pega no que está staged e faz commit disso, permitindo corrigir a mensagem prévia, substituindo o commit anterior
  - Exemplo:
    - Git commit –m “primeiro commit”
    - Git add ficheiroEsquecido
    - Git commit -v --amend –m “primeiro commit, com esquecido”
      - -v é de “verbose” / mostrar as diferenças commited
    - Pode fazer-se um alias para commits verbosos (cv)
      - git config --global "alias.cv" "commit -v“
      - Para usar: git cv --amend --m “mensagem”

Git : undos (2) e relacionados

- Se não se gosta das modificações feitas a um ficheiro desde o último commit e se se quiser reverter ao que estava
  - Git checkout ficheiro #e reaparece na pasta de trabalho
- Se o ficheiro já só estiver disponível numa revisão n passos anterior à HEAD
  - Git checkout HEAD~n ficheiro

  Ou
  - Git checkout commit-tag ficheiro
- Listar commits com tags
  - Git log --decorate
- Ver a primeira tag de um commit
  - Git describe --contains id-para-o-commit
  - Exemplo: git describe --contains head~1
- Ver todas as tags de um commit
  - Git tag --contains id-para-o-commit
- Para vermos os ficheiros existentes num commit
  - Git ls-tree head

Git : undos (3)

- Para remover um ficheiro do index/stage *e* da working directory
  - Git rm ficheiro
- Para remover do index/stage, mas deixar na diretoria de trabalho
  - Portanto, o ficheiro transita de staged para unstaged
  - Git rm --cached ficheiro
- Renomear ficheiros
  - Git mv old new
    - git rm --cached old; mv old new; git add new

Git : undos (4) : reset

- Git checkout ficheiro
  - Recupera do stage para a working directory o ficheiro
- Se as modificações que se querem recuperar já foram commited?
  - Git reset
    - Pode ser *extremamente* confuso
    - Entender utilizações básicas
  - Git reset HEAD
    - O index/stage torna-se igual ao último commit
    - Seguido de git checkout, a working directory recupera esses files
    - Poderiamos fazer um alias
      - » git config --global alias.unstage "reset HEAD"
  - Git reset HEAD ficheiro
    - Recupera do último commit, para o stage, apenas o ficheiro especificado

Git : ver a história de commits (1)

- História do repositório
  - Git log
    - Mostra todos os commits, por ordem crono inversa
    - Pressionar ‘Q’ no teclado, se necessário, para terminar
    - Pressionar SPACE para a próxima página
  - Git log --decorate
    - Decora o log com info extra, principalmente tags
    - Git log --decorate=full
      - Mais info ainda
  - Git log -2
    - Mostra apenas os 2 últimos commit
    - Equivale a
      - Git log –n 2
  - Git log –p -2
    - Mostra as diff introduzidas nos 2 últimos commit
    - Ler as diff
      - Diff header começa por diff –git ; a/ e b/ representam ficheiros
      - O modo 100644 indica que é um ficheiro “regular”
      - formato @@ from-file-range to-file-range @@ [header]
      - from-file-range está na forma -<start line>,<number of lines>
      - to-file-range está na forma +<start line>,<number of lines>

Git : ver a história de commits (2)

- Git log –p -2

```
$ git log p -2
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the verison number

diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,7 +5,7 @@ require 'rake/gempackagetask'
 spec = Gem::Specification.new do |s|
-    s.version   =   "0.1.0"
+    s.version   =   "0.1.1"
     s.author    =   "Scott Chacon"
```

```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

diff --git a/lib/simplegit.rb b/lib/simplegit.rb
index a0a60ae..47c6340 100644
--- a/lib/simplegit.rb
+++ b/lib/simplegit.rb
@@ -18,8 +18,3 @@ class SimpleGit
     end

 end
-
-if $0 == __FILE__
-  git = SimpleGit.new
-  puts git.show
-end
\ No newline at end of file
```

Git : ver a história de commits (3)

- História do repositório
  - Git log --stat
    - Outra forma de visualizar
    - Indica nº de ficheiros modificados
    - Indica quantas linhas acrescentadas e removidas, por ficheiro

```
$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the verison number

 Rakefile |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Git : ver a história de commits (4)

- Git log --pretty=format:"*format string*"
  - Muda a formatação do output de git log
- Formatos:
  - %s : frase do commit
  - %H ou %h : commit sha1 completa ou abreviada
  - %T ou %t : árvore de commits, completa ou abreviada
  - %P ou %p : identificar [abreviadamente] os commits pais
  - %[a|c][n|e|d|r] : name, email, date ou relative date do autor ou de quem fez o commit
    - %an , %ae , %ad ...
- Exemplo:
  - Git log --pretty=format:"%h - %an, %ar : %s"

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 11 months ago : changed the verison number
085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code
a11bef0 - Scott Chacon, 11 months ago : first commit
```

Git : ver a história de commits (5)

- Git log --pretty=oneline
  - Uma linha por commit

```
$ git log --pretty=oneline
ca82a6dff817ec66f44342007202690a93763949 changed the verison number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 removed unnecessary test code
a11bef06a3f659402fe7563abf99ad00de2209e6 first commit
```

- Git log --pretty=format:"*format string*" --graph
  - --graph tenta traçar em ASCII os branch/merge

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Added a method for getting the current branch.
* | 30e367c timeout code and tests
* | 5a09431 add timeout protection to grit
* | e1193f8 support for heads with slashes in them
|/
* d6016bc require time for xmlschema
*  11d191e Merge branch 'defunkt' into local
```

Git : ver a história de commits (6)

- Restritores em Git log
  - --author
  - --committer
  - --since, --until, --after, --before
    - Aceitam vários formatos de data, incluindo o militar
      - Git log --since=2.weeks
      - Git log --since="2012-10-01"

Git : conceitos de branch e head

- Branches
  - Representam ramos de desenvolvimento
  - A branch por defeito é “master”
  - Todas as branches chamam ao commit +recent HEAD
    - HEAD~1 representa o commit anterior ao +recente
    - HEAD~2 representa o commit anterior ao HEAD~1
    - HEAD~n-1 representa o commit anterior ao HEAD~n
  - Cat .git/refs/heads/master
    - Mostra o sha1 para último commit em master
    - Sha1 : valores 160 bits , representados em strings de 40 digitos hexadecimais
  - Git show head
  - A prática mais comum é que os branches em git sejam “short-lived” ou “topical”
    - Abre-se um novo branch para desenvolver uma nova feature, função, etc.
    - Quando o desenvolvimento estiver completo, faz-se merge com o “master” branch e apaga-se o “topical” branch

Git : Branching

- Branches : contextos de trabalho
- Git branch
  - Lista os branches
- Git branch branch-name
  - Cria o branch branch-name
  - Em alternativa
    - Git checkout –b branch-name
- Mudar de branch
  - Git checkout branch-name
    - Muda a working directory para o conteúdo do último commit do branch-name
- Apagar branch
  - Git branch –d branch-name
- Trazer para o contexto ficheiros de outro contexto
  - Git merge branch-name
    - Pode implicar gestão de conflitos

Git : remote repositories (1)

- Git remote
  - Lista os remote aliases conhecidos
- Git remote -v
  - Lista os remote aliases e mostra as localizações associadas
- Git remote add remote-name remote-location
  - Torna remote-name um alias da remote-location
- Git remote rm remote-name
  - Apaga o alias com nome remote-name

Git : remote repositories (2)

- Para sincronização com todos os remotes
  - Git remote update
- Downloads de repositórios remotos
  - Git fetch remote-name
    - Sincroniza com remote-name, fazendo download dos branches, ficheiros, etc, necessários
- Git fetch remote-name remote-branch:name-for-branch
  - Permite controlar o branch a fetch'ar e decidir o seu nome no repositório importador
- Git pull remote-name
  - Equivale a dois comandos seguidos:

    git fetch remote-name

    git merge remote-name/remote-branch

Git : remote repositories (3)

- Uploads para repositórios remotos
  - Git push remote-name branch-name
    - Tenta construir o branch local com nome branch-name no repositório remoto com nome remote-name
      - O sucesso depende as autorizações de escrita
    - Só resultará se o remote-name estiver atualizado localmente
    - Para garantir que o remote-name está atualizado
      - git fetch remote-name; git merge remote-name/branch-name
      - Depois disto o push deverá resultar
      - Git push remote-name branch-name

Git : exemplo de criação de repositório remoto partilhado

- (1) no servidor, numa pasta exemplo coding\remote.git onde se pretende o repositório fazer
- git init --bare
- (2) noutro repo, local ou remoto, fazer
  - Git init;
  - Echo hello>hello.txt
  - Git add .
  - Git commit –m “c1”
  - Git remote add server //server/coding/remote.git
  - Git push server master
- (3) qualquer cliente que agora faça
  - Git pull server master
    - Receberá hello.txt

Git : internals (1)

- As estruturas internas mantidas pelo Git
  - Blobs
    - Binary large objects
      - Representam UM ficheiro
  - Trees
    - Objetos que representam diretorias
    - Contêm blobs e outras trees
  - Commits
    - Representações do estado do repositório
    - Ponteiro para uma tree
  - Tags
    - Nomes em alternativa a sha1
    - Normalmente aplicados a commits

Git : internals (2)

- Todo o endereçamento é pelos 160 bits sha1 de cada objecto
  - "content address"
- Cada objeto commit
  - Aponta para o seu "pai"
    - Quem lhe antecede na sequência de commits

Referências

- http://gitref.org
- http://git-scm.com/book

Git - ajudas

- Git help comando
- Exemplo de pedido de ajuda sobre git branch
  - Git help branch
- branching
- Branches are used to develop features isolated from each other. The *master* branch is the "default" branch when you create a repository. Use other branches for development and merge them back to the master branch upon completion.
- create a new branch named "feature_x" and switch to it using
  git checkout -b feature_x
  switch back to master
  git checkout master
  and delete the branch again
  git branch -d feature_x
  a branch is *not available to others* unless you push the branch to your remote repository
  git push origin <branch>
- update & merge
- to update your local repository to the newest commit, execute
  git pull
  in your working directory to *fetch* and *merge* remote changes.
  to merge another branch into your active branch (e.g. master), use
  git merge <branch>
  in both cases git tries to auto-merge changes. Unfortunately, this is not always possible and results in *conflicts*. You are responsible to merge those *conflicts* manually by editing the files shown by git. After changing, you need to mark them as merged with
  git add <filename>
  before merging changes, you can also preview them by using
  git diff <source_branch> <target_branch>

Git – observação do repo

- Git log
  - Toda a história, no formato por defeito
- Git log –n <number>
  - Os n commits +recentes
- Git log --oneline
  - Toda a história, cada commit condensado em 1 linha
- Git log --stat
  - O mesmo que “git log” +info de modificações por ficheiro, por commit, com #linhas acrescentadas e removidas
- Git log –p
  - “patch view”. Vista detalhada com todas as diff em cada commit

Git – observação do repo

- Git log --author =“<exp>”
  - Commits apenas pelos authors que satisfizerem a <exp>
  - Exº
    - git commit -a --author="Jota <jota@gmail.com>" -m "c5, por jota“
    - Git log --author=“jota”
      - Lista apenas os commits de j
- Git log --grep=“<exp>”
  - Commits cuja message satisfaz a <exp>
- Git log <since>..<until>
  - Since e until devem ser “revision refs”, por exemplo commits-ids ou branch names

Git – observação do repo

- Git log <ficheiro>
  - Lista apenas os commits em que o <ficheiro> foi incluído
- git log --graph --decorate --oneline
  - Graph faz um grafo dos commits
  - Decorate permite ver nomes de branches e tags
  - Tudo condensado em uma linha

Git |UNDOs | git checkout

- Git checkout
  - Checkout para ficheiros
    - Potencialmente destrutivo
  - Checkout para commits
    - Não destrutivo / detached HEAD
  - Checkout para branches

Git |UNDOs | git checkout

- Git checkout <commit> <file>
  - A <file> vem para a working directory e APAGA outra com o mesmo nome que eventualmente exista
    - Destrutivo
  - A <file> é acrescentada ao stage
- Git checkout <commit>
  - Todos os ficheiros da working directory tornam-se o que eram no <commit> identificado, mas nada foi apagado, apenas a HEAD deslocou-se para o “passado”
    - Estado corrente preservado no branch master
    - Não destrutivo
    - Situação de “detached” HEAD

Git |UNDOs | git checkout

- Git checkout <commit>
  - HEAD deixa de apontar para um branch, como habitualmente acontece
  - HEAD passa a apontar para um commit
    - “Detached HEAD”

Git |UNDOs | git checkout

- Ver revisões anteriores
  - Git log --oneline
    - Apresenta commits com 7 símbolos da hash
  - Pode fazer-se git checkout de um desses commits
    - A working dir fará mirror do <commit> escolhido
    - Para voltar ao estado corrente
      - Git checkout master
        - » Uma vez aí
          - Git revert
          - Git reset
          - Para reverter alguma alt indesejada => estudar

Git reset – usos sugeridos

- Git reset <file>
  - Remove a <file> do stage
  - Mantém a <file> na WD
- Git reset
  - Torna o stage idêntico ao <commit> +recente
  - Equivale a anular qualquer modificação ao stage desde o <commit> anterior
    - Ou seja, equivale a “limpar” o stage
  - Não toca na WD
  - Oportunidade para recomeçar o stage
- Git reset --hard
  - Stage + WD tornam-se idênticos ao último <commit>

Git reset – usos perigosos

- Git reset <commit>
  - Não toca na WD
  - O stage torna-se o que era em <commit>
  - As mods posteriores a <commit> estão concentradas na WD
  - Podem assim refazer-se commits desde <commit> de outra forma
- Git reset --hard <commit>
  - WD e stage tornam-se o que eram em <commit>
  - Vão-se mods posteriores a <commit>
  - PERDE-SE HISTÓRIA do projeto: perdem-se todos os commits posteriores a <commit>
- git reset --soft HEAD~1
  - WD e stage sem alterações
  - Vai-se o último commit (ou tudo posterior a HEAD~1)