Git (libro Amazon)

devops, cursos, git

uso básico

configuración inicial

- `git config –global user.name <nombre>`
- `git config –global user.email <email>`
- `git config –global core.editor <EDITOR>`
- `git condig –global push.default {matching|simple|current|nothing}` : ver *sincronizando directorios*
- `git config –global core.excludesFile <FILE>` : fichero global de exclusión de ficheros
- `git config –list`

iniciando repositorio

- `git init`
- `git clone <http|https|ssh>`
- `git config –list` : en el directorio del repositorio te da info sobre el mismo

trabajando con Git

- Directorio de trabajo → Index → Head
- `git status`
- `git add`
    - `git add -u` : añade al Index los archivos que deben ser borrados
    - `git add -A` = git add -u & git add .
- `git commit`
    - `git commit -m «<COMENTARIO>»`
    - `git commit -a` : git add -A & git commit

sincronizando repositorios

- `git clone`
- `git remote -v[vv]` : lista información de los repositorios remotos
    - `git remote add <ALIAS> <DIRECCION_REPOSITORIO>`
    - `git remote rm <ALIAS>`
    - `git remote rename <ALIAS> <NUEVO_ALIAS>`
- `git pull <ALIAS> <RAMA>`
    - `git pull` = git fetch & git merge
    - ALIAS por defecto es **origin**
    - RAMA pode defecto es **master**
- `git push <ALIAS> <RAMA>`
    - hemos de estar al día en nuestro respositorio local para poder hacer un push
    - comportamiento por defecto v2.x : **simple** en lugar de **matching**
        - simple: solo sube rama activa a la rama de la que has hecho pull, si no tiene el mismo nombre da error
        - matching: sube todas las ramas, si no existe la crea
        - current: sube los cambios de la rama activa a la rama del mismo nombre, si no existe, se

crea
- nothing: test/debug
- upstream : idem *simple* pero no da error si tiene otro nombre

.gitignore

- https://github.com/github/gitignore

viendo el historial

- `git log`
  - parámetros:
    - `-#_entradas`
    - `–oneline`
    - `-p` : más detalle, con diff
    - `–graph`

borrado de archivos

- `git rm` <ARCHIVO> : borra archivo + borra archivo del Index
  - `git rm –cached` <ARCHIVO> : borra solo del Index
  - `git reset HEAD` <ARCHIVO> : idem anterior

arrepentimientos

rehacer un commit

- `git commit –amend` : si no hay modificación de archivos (no has modificado en Index), edita el comentario
- si has olvidado algún archivo, lo añades y ejecutas la instrucció anterior

deshacer cambios de un archivo

- `git checkout –` <ARCHIVO> : deshace los cambios que has hecho, lo recupera del HEAD

volviendo al pasado

- `git reset –hard` <HASH_COMMIT>:
  - deshace commits posteriores al indicado
  - recupera los archivos del commit indicado
  - DESAPARECEN TODOS LOS CAMBIOS POSTERIORES
  - se recomienda hacer un PUSH o ejecutarlo sobre otra rama

resolviendo conflictos

- hay commits posteriores en tu rama, error al hacer push
  - hacer pull
  - resolver conlictos, si los hubiese (primero aparece lo tuyo, entre <<<<< y ==== y lo que hay en el remoto está entre ===== y >>>>>>

    - hacer push

viendo/recuperando archivos antiguos

- `git show <HASH_COMMIT>`
- `git show <HASH_COMMIT>:path/to/file`
- `git show <HASH_COMMIT>:path/to/file > archivo_copia`

más usos de Git

organización

no hay reglas de como organizar tu trabajo, aunque si que hay reglas de como **NO** hacerlo... se ha de organizar para que el trabajo de los otros no te distraiga

qué poner en el directorio principal

- `README.md` : fichero mostrado en la página principal de Github. Explicar en pocas palabras de que va el proyecto, como instalarlo, prerrequisitos, licencia, navegación por el repositorio
- `INSTALL.md` : instrucciones detalladas de instsalación
- `.gitignore`
- `LICENSE`
- `TODO`

estructura habitual con directorio test

- Git tiene una estructura plana (el todo es tratado en un conjunto, al contrario que CVS o Subversion, que podían tratar un directorio como un proyecto independiente)
- Git permite trabajar con submódulos para emular este comportamiento. Con sentido en proyectos de terceros del que depende tu proyecto o proyectos/equipos muy grandes
    - `git submodule add <URL_REPOSITORIO> <DIRECTORIO>` + `cd <DIRECTORIO>` + `git submodule init` + `git submodule update` + `git pull`

flujos de trabajo

ramas

Una rama es un nombre a un commit específico y todos los commits que son antecesores del mismo

ramas ligeras: etiquetas

- `git tag <etiqueta>` : crea etiqueta, asociada a un commit, para marcar algún tipo de hito. De caracter local
    - `git tag -a <etiqueta>` : añade una nota desde editor, o **-m «comentario»** inline
    - `git show <etiqueta>` : mostrará la nota
- `git tag`
- `git describe` : indica el camino desde la última etiqueta a un commit concreto
- `git push –tags`

creando y fusionando ramas

- `git checkout -b <RAMA>` : git branch <RAMA> + git checkout <RAMA>
  - al hacerlo, sobreescribe los cambios que no esten pasados a commit.
  - si se quieren conservar sin realizar el commit, se pued hacer un `git stash` : almacen temporal
  - para recuperar, `git stash apply –index`
  - para establecer un origen por defecto (upstream): `git push –set-upstream <ALIAS_REMOTO> <RAMA>`
  - `git branch` nos indica las ramas y en la que estamos (marcada con un *)
  - `git branch –all` nos muestra todas las ramas

fusionando

1. `git checkout <RAMA_QUE_RECIBIRÁ_LA_FUSION>`
2. `git pull <ALIAS_REMOTE> <RAMA_QUE SE FUSIONARÁ`
3. una vez fusionadas correctamente, se podría descartar la rama «muerta»:
   1. `git branch -d <RAMA_MUERTA>` : en local
   2. `git push <ALIAS_REMOTE> :<RAMA_MUERTA>` : actualizar el borrado en el repositorio remoto

- `git checkout <ARCHIVO>` : recupera desde **master** (por defecto) el fichero a la rama actual

los misterios del rebase

reescritura de historia (OJO con los push/pull)

quién hizo qué

- `git log –pretty=short` : resumen de commits, autores, fechas...
- `git blame` : cambios a nivel de fichero

Usando Git como los profesionales: GitHub

github pages

- páginas estáticas
- originalmente, rama **gh-pages**, ahora desde la raiz del proyecto o el subdirectorio **docs**
- se puede configurar en **Settings** → **GitHub pages** → Automatic Page Generator (deja elegir algunas plantillas). El generador coge el fichero **README.md** y lo parsea a HTML, y genera un domino **usuario.github.io/proyecto** que sirve las páginas HTML

hooks

ganchos o eventos que se activan cuando se produce alguna acción → petición REST (debidamente formada)

- **Settings** → WebHooks & Services → Configure services
- se activan al hacer un **push** a GitHub
- tipos de servicios:
  - integración continua
  - mensajería
  - entrega continua

- sistemas de trabajo en grupo
- análisis del código

cliente GitHub (hub)

Hooks, ejecutando código tras una orden Git

estructura repositorio