simplificando git

piérdele el miedo a la línea de comandos

de antemano

- La charla se grabará y se compartirá su acceso
- Las diapositivas se compartirán al final de la charla
- Cualquier duda no dudes en comentarla (chat o voz, lo que prefieras :D)

¿para quién está pensada la charla?

- Aquellas personas que le tienen reparo a los comandos
- Quienes no hayan usado, o apenas, git
- Quienes tengan algo de experiencia pero les gustase más soltura
- De principiante a intermedio
  - Avanzado es responder en stackoverflow, y eso me queda lejos
- Personas con afán de revisitar conceptos

estructura de las charlas

- I – Introducción y modelo mental, los fundamentos
- II – Manejo de *armas*, aprende a defenderte
- III – Profundización y salvavidas, entiende errores y aprende su *origin*

presentaciones

Soy **Pepe**, aunque Jose no me molesta, hago cosas de front, experto en tirar abajo producción, años de experiencia constatada rompiendo ramas en conflictos de merge.

Odiaba los comandos… y ahora… no tanto

temas a tratar

- Origin
- Upstream
- stategy option
  - ours/theirs
- Merge
- Rebase
- Bisect
- Checkout
- Branches
- branch rename
- crear ramas con checkout
- Remote
- git stash
  - keys y pop
- git fetch vs git pull vs git push
  - git fetch especial (git pull && git push)
- git cherry pick
- Entornos
  - stating, local, remote, origin
- Add, commit, reset
- git config
  - repo, system, global, user
- Git patch
- Git submodules and subtrees
- Git log
- Git rebase interactive
- Reflog
- Deltas
- Git hooks
- Gitflow y tbd (**t**runk **b**ased **d**evelopment)

comprenderás

- Qué es git
- Commits
  - cómo crearlos, por qué y a dónde añadirlos
- Ramas
- merge y rebase
- Un modelo mental para evitar problemas

introducción a git

Antes de nada, vamos a explicar algunos conceptos que ayudarán a entender/repasar cómo funciona Git

Git es un Sistema de Control de Versiones (VCS), esto quiere decir que nos permite **gestionar** un **historial de versiones** de lo que queramos

Vamos a empezar desde lo más básico e ir construyendo desde ahí

historial de versiones y versionado

¿por qué necesitamos saber git?

- Estándar de industria
- Buena comunidad
- Años *battle-tested* en diferentes entornos y proyectos
- Relativamente sencillo

Alternativa, siempre hay una:

- Subversion, fue la herramienta que motivó a Linus Torvald a crear Git

ESQUEMAS

- Un cambio altera la lista enlazada, y cada cambio en la lista enlaza es un nuevo hash, explicar entonces porque hace un rebase el cambio y aparecen commits a descargar
- Dibujo de una lista enlazada
- Dibujo simplificado de un hash

Ideas

- en git, para lo de git, si la rama es un hilo, un merge es hacerle un nudo y un rebase es deshilacharlo y volverlo a hilar desde determinado punto, según el contexto, una cosa interesa más que la otra

modelo mental

Antes de empezar explicaremos dos conceptos, por encima:

- **hashes**
- **listas enlazadas** (Linked Lists)
- **cadenas**, plantearme si es mejor usar cuerdas o cadenas como analogía

La idea de este “modelo mental” es tener ejemplos cercanos para poder entender la *magia* de git

hashes

Funciones matemáticas
Únicas, uid, un hash no da dos veces el mismo id
Inmutabilidad, un cambio, nuevo hash

listas enlazadas

Estructuras de datos
Nodos anteriores y posteriores

ramas; merge y rebase, modelo mental

Tanto el **merge** como el **rebase** se verán más adelante en detalle.

Pero para poder empezar a explicar conceptos, entendamos que:

- Una **rama** (branch) es una cuerda formada por distintos cambios (**commits**)
- Un **merge** es la acción de juntar dos cuerdas mediante un nudo
- Un **rebase** es la acción de deshilachar dicha cuerda y volver a hilarla añadiéndole la cuerda que queramos

Un **merge** es borrón y cuenta nueva, un **rebase** es reconstruir la historia a partir de cierto punto.

ramas, merge y rebase, dibujado

branch

git branch

renombrar una rama

git checkout

conflictos de merge

git merge

git rebase

entornos

- Local
- remote

Origin, un alias

git remote

estados

untracked

- git no tiene constancia de la existencia de este archivo, es decir, se acaba de “crear”, al menos para git

unstaged

- se ha modificado (su contenido ha cambiado, o se ha eliminado), pero no está en **staging**

staging

- hemos registrado cambios, y los subimos al área antes de un **commit**

committed

- un conjunto de cambios en **staging** que se confirman, o **commitean**

remote

- un **commit** que se encuentra en la fuente de verdad (**remote**) de la rama

git add

Nos permite habilitar el seguimiento y pasar a **staging** los archivos o carpetas que queramos.

Archivos o carpetas, porque podemos añadir elementos y subelementos de una.

git add . Añade todo el contenido a partir del punto de llamada.

git add se ve sujeto al **.gitignore**, solamente aquellos archivos que NO se encuentren excluidos por el **.gitignore** se subirán

.gitignore

Permite ignorar archivos, o crear reglas para **ignorar**, estas reglas pueden ser **exclusivas** o **inclusivas**, ignora todo esto, no ignores nada salvo que tenga esto.

Añadir ejemplos de exclusivo e inclusivo, y de ficheros normales

Es importante destacar que si git ya tiene constancia de un archivo, añadirlo a un **.gitignore** podrá no tener el funcionamiento que deseas.

- **git rm –cached […fichero(s)]**, nos ayudará, los detalles más adelante

git commit

git push

git pull

git fetch

siguiente sesión

En siguientes sesiones profundizaremos sobre git un poco más, aprenderás trucos, distintas metodologías y si lo he hecho bien, git y sus comandos ya no te serán tan extraños :D.

En la siguiente sesión, **ganando confianza con los comandos**, veremos en más detalle el rebase, da para rato, y comentaremos gitflow, y algunas cositas más.

gracias por vuestra atención

¡¡Que tengáis un buen día!!

simplificando git

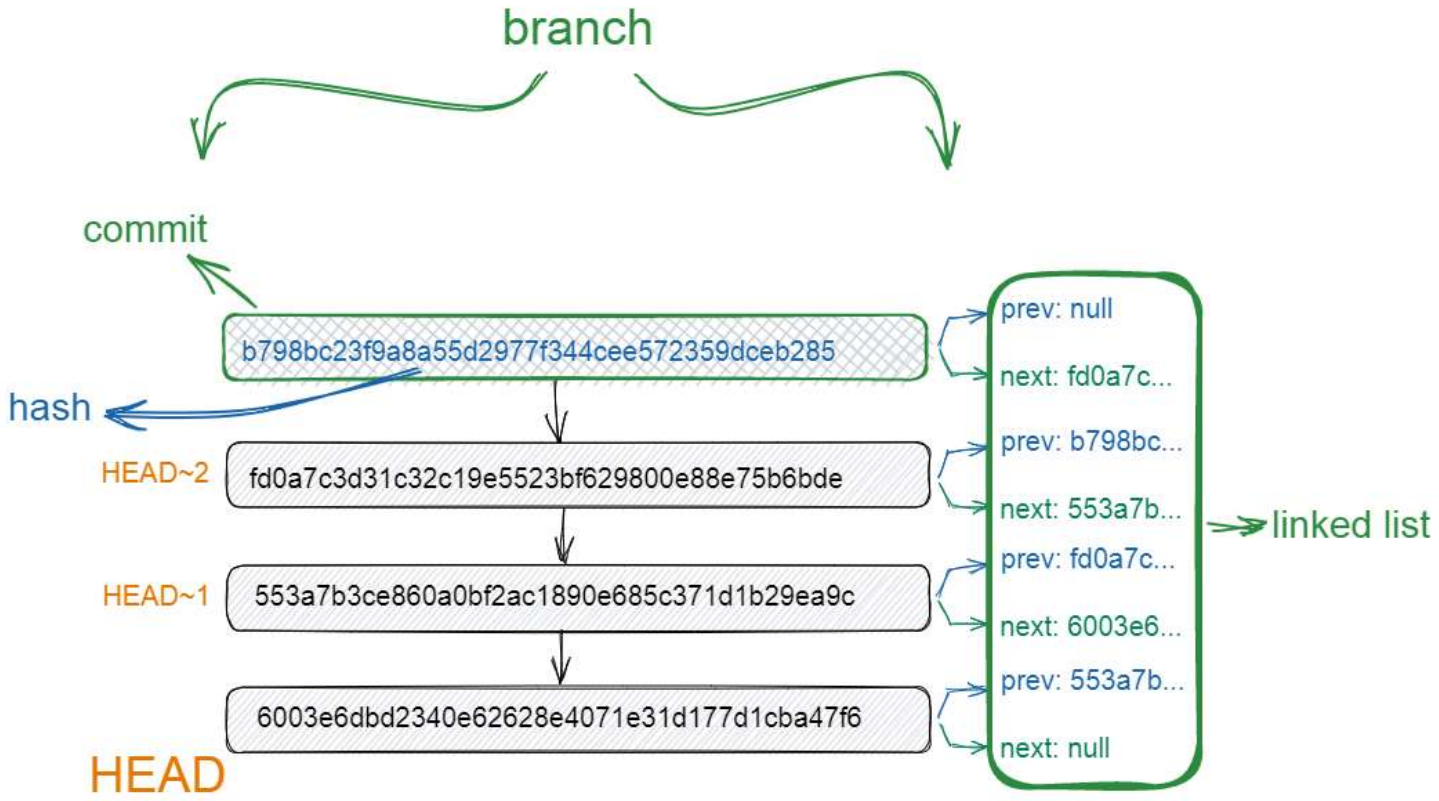
ganando confianza con los comandos

te sonarán

- Rebase interactivos
  - Squash

recapitulando...

- git init
- git add
- git commit
- git branch
- git checkout
- git push
- git pull
- git fetch
- git merge
- git rebase



ganando confianza con los comandos

- simplifying-git-gaining-confidence-in-the-cli

- Git rebase interactive
  - Squash
  - Edit
  - Rename
  - Remove
  - Add
- Git log
- Git config

rebase interactivo

git rebase -i

asfasfa

rename

Sigue siendo un cambio en la *cuerda*, así que se reconstruye igual que cualquier otro cambio

squash

edit

delete

entornos de configuración

Por orden de importancia:

repository

- El contenido en local

user

- Un usuario del sistema operativo

system

- Aplicaría a todo el sistema

global

- Lo que se aplique aquí sería el nuevo valor por defecto

Pseudo Código:

```
if repoConfig.has("user.name")
   return repoConfig.get("user.name")
if userConfig.has("user.name")
   return userConfig.get("user.name")
if systemConfig.has("user.name")
   return systemConfig.get("user.name")
if globalConfig.has("user.name")
   return globalConfig.get("user.name")

return ""
```

git config

gitflow

ramas del gitflow

releases en gitflow

en la última sesión

simplificando git

dominando la línea de comandos

sabrás

- zanjar los problemas de case sensitiveness
- Cuándo compensa usar el cliente web
  - [nota de autor: poner en su slide] evitar confusiones con los tags
- La rompedora filosofía de trunk base development

los salvavidas

- --no-verify
  - Commit and push
- --amend
- --allow-empty
- Cherry-pick
- --strategy-options
- Git bisect
- Rm --cached

- git hooks
- git mv
- git rm

git mv

git rm

CaSe iNsEnSiTiVeNeSs

¿Es hola y Hola lo mismo? Pues no se, depende de qué se entienda por igual.

Git también lo entiende como un depende, depende del sistema, por eso en Windows a veces cambiamos mayúsculas o minúsculas de un fichero y “nos ignora”.

La solución:

- git config --global core.ignorecase true
- git mv PascalCase.java camelCase.java

git cherry-pick

git hook

Los hooks son parte del ciclo de vida de git

Stage → commit → push → merge → pull

En nuestro *modelo mental* serían los amarres [a trabajar en ello, la analogía]

hooks, ¿buenos o malos?

Los pre-commits y pre-pushes, pueden parecer tentadores, pero son peligrosos

cómo saltarte los hooks

En un mundo donde hay hooks que frenan más de lo que ayudan, saber ignorarlos puede ahorrarte mucho tiempo y sufrimiento

trunk base development

créditos

uiGradients, los degradados han sido en su mayoría de aquí

- https://uigradients.com/

excalidraw, diagramas y dibujitos

- https://excalidraw.com/

Learn Git Branching, página web interactiva

- https://learngitbranching.js.org/

expandiendo el conocimiento

Aprendiendo Git – Miguel Angel Durán

- https://leanpub.com/aprendiendo-git

Learn Git Branching, página web interactiva

- https://learngitbranching.js.org/