¡Se puede entender cómo funcionan Git y GitHub!

Julen Astigarraga1, Verónica Cruz-Alonso2

1. Universidad de Alcalá, Grupo de Ecología Forestal y Restauración (FORECO), Departamento de Ciencias de la Vida, 28805 Alcalá de Henares (Madrid), España
2. Department of Landscape Architecture, Graduate School of Design, Harvard University, 02138 Cambridge MA, Estados Unidos

Autor para correspondencia: [julenastigarraga@gmail.com](mailto:author@gmail.com), [veronica.cral@gmail.com](mailto:author@gmail.com)

Contribución de los autores:

Julen Astigarraga: conceptualización y redacción – borrador original. Verónica Cruz-Alonso: conceptualización, visualización y redacción – revisión y edición.

# Palabras clave

ciencia abierta; control de versiones; trazabilidad; reproducibilidad

# Keywords

open science; reproducibility; traceability; version control

La reproducibilidad es una condición *sine qua non* del método científico (Peng, 2011). Algunos autores plantean que gran parte de la ecología es ya una ciencia de *big data* (Hampton et al., 2013; Farley et al., 2018), lo que hace indispensable utilizar herramientas informáticas que garanticen la trazabilidad de todo el proceso de desarrollo de proyectos (desde su creación hasta su publicación), asegurando de esta manera su reproducibilidad (Rodríguez-Sánchez et al., 2016). En este sentido, es recomendable utilizar alguna herramienta de control de versiones como Git (<https://git-scm.com/>) combinada con plataformas en línea para albergar los proyectos (p. ej. GitHub, <https://github.com/>), facilitando así el seguimiento de los proyectos y la coordinación entre colaboradores (Blischak et al., 2016; Galeano, 2018; Rodríguez-Sánchez, 2020). Aunque existen multitud de manuales disponibles gratuitamente sobre cómo utilizar Git y GitHub (ver Apéndice 1), estas herramientas son complejas y tienen una curva de aprendizaje pronunciada. El objetivo de la presente nota es dar a conocer la estructura, funcionalidad y potencialidad de Git, así como su interacción con GitHub, para el trabajo en proyectos colaborativos en ecología.

# ¿Qué es Git?

Git es un sistema avanzado de control de versiones (como el “control de cambios” de Microsoft Word) distribuido (Ram, 2013; Blischak et al., 2016). Git permite “rastrear” el progreso de un proyecto a lo largo del tiempo ya que hace “capturas” del mismo a medida que evoluciona y los cambios se van registrando. Esto permite ver qué cambios se hicieron, quién los hizo y por qué, e incluso volver a versiones anteriores. Además, Git facilita el trabajo en paralelo de varios participantes. Mientras que en otros sistemas de control de versiones (p. ej. Dropbox, <https://www.dropbox.com/>) hay un servidor central y cualquier cambio hecho por un usuario se sincroniza con este servidor y de ahí con el resto de usuarios, Git es un control de versiones distribuido que permite a todos los usuarios trabajar en el proyecto paralelamente e ir haciendo “capturas” del trabajo de cada uno para luego unirlos. Otras alternativas de control de versiones distribuido comparables a Git son Mercurial (<https://www.mercurial-scm.org/>) o Bazaar <https://bazaar.canonical.com/>), pero Git es el más utilizado.

# ¿Qué es GitHub?

GitHub es un servidor de alojamiento en línea o repositorio remoto para albergar proyectos basados en Git que permite la colaboración entre diferentes usuarios o con uno mismo (Perez-Riverol et al., 2016; Galeano, 2018). Aunque existen distintos repositorios remotos (p. ej. GitLab, <https://gitlab.com/>, o Bitbucket, <https://bitbucket.org/>) con funcionalidad similar, GitHub es hoy en día el más utilizado. GitHub registra el desarrollo de los proyectos de manera remota, permite compartir proyectos entre distintos usuarios y proporciona la seguridad de la nube entre otras funciones. Cuando se trabaja en proyectos colaborativos, la base de la interacción entre Git y GitHub es que todos los colaboradores de un proyecto están de acuerdo en que GitHub contiene la copia principal del proyecto, es decir, GitHub contiene la copia centralizada del control de versiones descentralizado.

# Flujo de trabajo en Git y GitHub

Git es capaz de rastrear todos los archivos contenidos en un repositorio. Un repositorio es un directorio donde desarrollar un proyecto que contiene todos los archivos necesarios para el mismo. Para comprender cómo Git registra los cambios y cómo podemos compartir dichos cambios con nuestros colaboradores es importante entender cómo se estructura Git y cómo se sincroniza con GitHub. Hay cuatro “zonas” de trabajo (**Fig. 1**):

1. **Directorio de trabajo (working directory):** es donde se está trabajando. Esta zona se sincroniza con los archivos locales del ordenador.
2. **Área de preparación (staging area o Index):** es la zona intermedia entre el directorio de trabajo y el repositorio local. Es la zona de borradores. Contiene los archivos que se van a registrar en la siguiente “captura” de Git.
3. **Repositorio local (local repository o HEAD):** es donde se registran todos los cambios capturados por Git en tu ordenador.
4. **Repositorio remoto (remote repository):** es donde se registran todos los cambios capturados por Git en la nube (GitHub).

Si se ejecuta Git desde la línea de comandos de Unix shell, la transición entre zonas de trabajo se realiza mediante comandos (**Fig. 2**), aunque también se puede realizar a través de un cliente (p. ej. RStudio, <https://www.rstudio.com/>, incorpora una pestaña llamada “Terminal” que facilita esta labor). El primer paso para trabajar con Git es crear un repositorio en GitHub (repositorio remoto) y clonarlo en nuestro ordenador (respositorio local; ver Galeano (2018)).

Para añadir una actualización de algún archivo del directorio de trabajo al área de preparación se utiliza el comando git add <nombre de archivo>. Este comando indica a Git que se quieren incluir las actualizaciones de algún archivo en la próxima “captura” del proyecto y que Git las registre. Sin embargo, git add no afecta al repositorio local. Para ver el estado del directorio de trabajo y del área de preparación se utiliza git status. Permite ver qué archivos están siendo rastreados por Git, qué cambios han sido añadidos al área de preparación (*staged*) y qué archivos están siendo registrados por Git.

Para registrar los cambios que nos interesan se utiliza git commit. Al ejecutar git commit se hace una “captura” del estado del proyecto. Junto con el *commit* se añade un mensaje con una pequeña explicación de los cambios realizados y por qué (p. ej. “incluyo las referencias en el formato de Ecosistemas”). Cada git commit tiene un SHA (Secure Hash Algorithm) que es un código alfanumérico que identifica inequívocamente ese *commit* (p. ej. 1d21fc3c33cxxc4aeb7823400b9c7c6bc2802be1). Con el SHA siempre se pueden ver los cambios que se hicieron en ese *commit* y volver a esa versión fácilmente.

Por último, git push permite subir los cambios que hemos hecho a GitHub y quedarán visibles para nuestros colaboradores. Básicamente, git commit registra los cambios en el repositorio local y git push actualiza el repositorio remoto con los cambios y archivos asociados. Cuando se retoma un proyecto tras horas, días o incluso meses, con git pull se descargan todas las actualizaciones que haya en GitHub (nuestras o de nuestros colaboradores), que se fusionarán (*merge*) con el último *commit* en nuestro repositorio local.

# ¿Cómo se puede trabajar paralelamente?

Git permite crear una “rama” paralela al proyecto si se desea seguir una línea independiente de trabajo, bien por ser diferente de la principal (p. ej. probar un nuevo análisis) o bien para desarrollar específicamente una parte del proyecto (p. ej. trabajar sólo en la escritura de los métodos de un artículo mientras otros colaboradores trabajan en otras secciones). Las ramas permiten trabajar en el proyecto sin interferir con lo que están haciendo los compañeros (**Fig. 3**). En Git, una rama es un *commit* al que le se le da un nombre y que contiene un puntero a un SHA específico que es el origen de la rama. La rama “*main*” es la rama por defecto cuando se crea un repositorio. Las demás ramas se crean con git checkout -b <nombre de rama>. Para moverse de una rama a otra se utiliza git checkout <nombre de rama> (p. ej. git checkout main para volver a la rama principal).

Cuando el trabajo desarrollado en una rama se da por finalizado y se quiere unir a la rama principal “*main*” hay que hacer la unión utilizando el comando git merge (**Fig. 3**). Con git checkout main se posiciona el puntero de Git en el último *commit* de la rama principal a la que quiero unir la otra rama y con git merge <rama a unir> se fusionan los cambios hechos en las dos ramas. Esto se puede hacer en la terminal como acabamos de ver pero también se puede hacer con el botón “*pull request*” en la página del proyecto en GitHub (ver Galeano, 2018).

Git puede encontrar conflictos al fusionar ramas que hay que arreglar manualmente. Esto ocurrirá si en las dos ramas se ha cambiado la misma parte de un archivo. Git muestra dónde están los conflictos así:

<<<<<<código del main=======código de la rama a unir>>>>>>

Para solucionarlo hay que escoger los cambios de la rama principal o de la rama a unir según corresponda. Una vez solucionados, Git permite completar el *merge* (es decir, un nuevo *commit* que contendrá las ramas fusionadas). La mejor manera de evitar conflictos o por lo menos reducir su dificultad es realizar cambios pequeños y sincronizar frecuentemente con GitHub.

# Agradecimientos

JA está financiado por el Programa Predoctoral de Formación de Personal Investigador No Doctor del Departamento de Educación del Gobierno Vasco y VCA por el Real Colegio Complutense (Becas Postdoctorales 2020). Agradecemos al grupo de Ecoinformática, especialmente a F. Rodríguez-Sánchez e I. Bartomeus, por adentrarnos en el mundo de Git y Github, y a P. Ruiz-Benito por su colaboración en todo nuestro proceso de aprendizaje.

###### REFERENCIAS

Blischak, J.D., Davenport, E.R., Wilson, G. 2016. A Quick Introduction to Version Control with Git and GitHub. *PLOS Computational Biology* 12: e1004668.

Farley, S.S., Dawson, A., Goring, S.J., Williams, J.W. 2018. Situating Ecology as a Big-Data Science: Current Advances, Challenges, and Solutions. *BioScience* 68: 563-576.

Galeano, J. 2018. ¿Por qué usar GitHub? Diez pasos para disfrutar de GitHub y no morir en el intento. *Ecosistemas* 27: 140-141.

Hampton, S.E., Strasser, C.A., Tewksbury, J.J., Gram, W.K., Budden, A.E., Batcheller, A.L., Duke, C.S., Porter, J.H. 2013. Big data and the future of ecology. *Frontiers in Ecology and the Environment* 11: 156-162.

Peng, R.D. 2011. Reproducible Research in Computational Science. *Science* 334: 1226-1227.

Perez-Riverol, Y., Gatto, L., Wang, R., Sachsenberg, T., Uszkoreit, J., Leprevost, F. da V., Fufezan, C. et al. 2016. Ten Simple Rules for Taking Advantage of Git and GitHub. *PLOS Computational Biology* 12: e1004947.

Ram, K. 2013. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine* 8: 7.

Rodríguez-Sánchez, F. 2020. Quince consejos para mejorar nuestro código y flujo de trabajo con R. *Ecosistemas* 29: 2129-2129.

Rodríguez-Sánchez, F., Pérez-Luque, A.J., Bartomeus, I., Varela, S. 2016. Ciencia reproducible: qué, por qué, cómo. *Ecosistemas* 25: 83-92.

###### PIES DE FIGURA

**Figura 1**. Diferentes zonas de trabajo en Git y GitHub. Imagen de fondo de Semana (<https://www.semana.com/>).

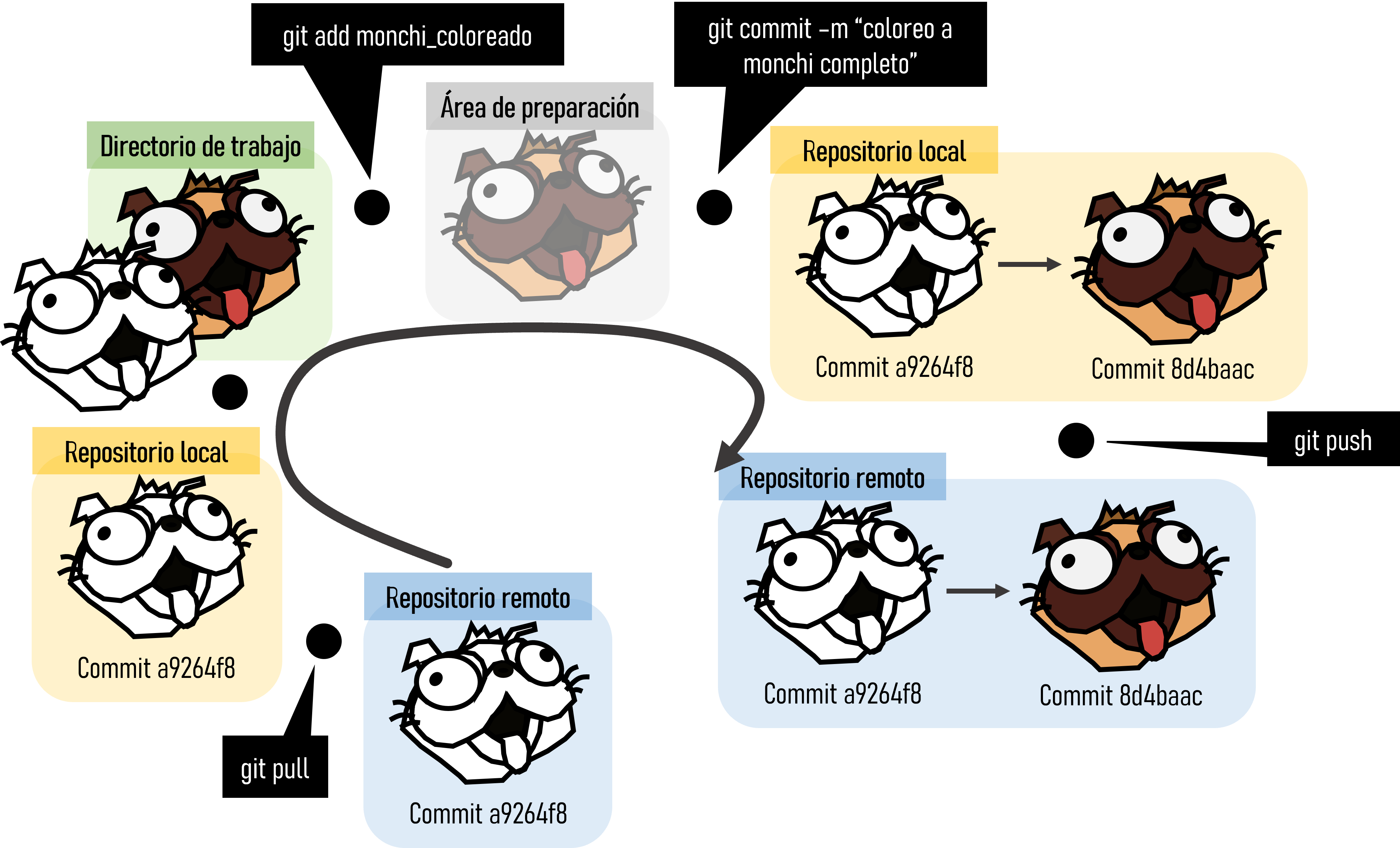
**Figura 2**. Flujo de trabajo en Git y GitHub.

**Figura 3**. Proceso de creación y unión de ramas.

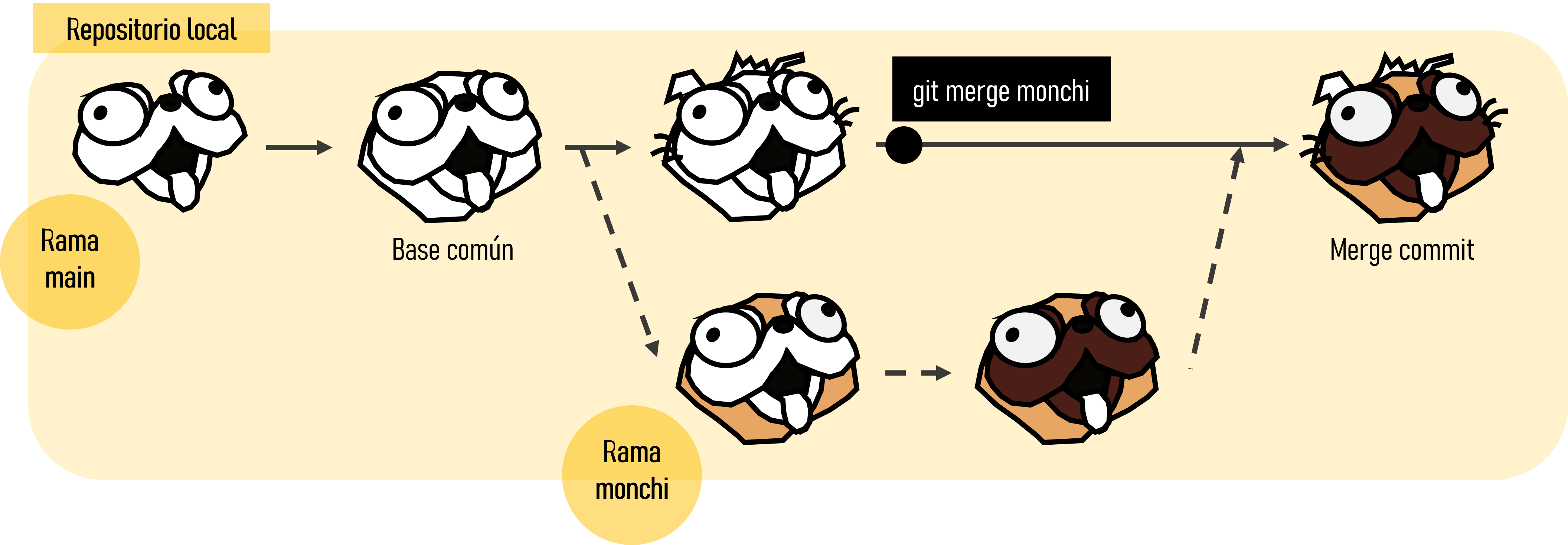
###### FIGURA 1



###### FIGURA 2



###### FIGURA 3



###### Apéndice 1: recursos para entender Git y Github

**Información más detallada de los temas tratados en esta nota y ejercicios**

* [Introducción a Git y GitHub, dos herramientas para una ecología más colaborativa y reproducible](https://github.com/Julenasti/intro_git-github)

**Ciencia reproducible**

* [Ciencia reproducible: qué, por qué, cómo](https://github.com/ecoinfAEET/Reproducibilidad)

**Control de versiones (Git)**

* [Manual de referencia de Git](https://git-scm.com/docs)
* [Software Carpentry](http://swcarpentry.github.io/git-novice/)
* [Atlassian Bitbucket](https://www.atlassian.com/git/tutorials)
* [Oh Shit, Git!?!](https://ohshitgit.com/)
* [git - la guía sencilla](https://rogerdudler.github.io/git-guide/index.es.html)

**Integrar Git, GitHub y RStudio**

* [Happy Git and GitHub for the useR](https://happygitwithr.com/)

**Enseñar y aprender con GitHub**

* [GitHub Education para profesores e investigadores](https://docs.github.com/en/education/explore-the-benefits-of-teaching-and-learning-with-github-education/use-github-in-your-classroom-and-research/about-github-education-for-educators-and-researchers)