A tour of Git and GitHub

Nicolas Escobar

July 7, 2024

What's Git?

- Git $\neq$ GitHub
- Git = version control.
- GitHub = collaboration + sharing.

Git is a way to keep track of *some* of the changes of *some* the contents of a directory.
The full record of those changes is called a *repository* or *repo*.
I'll focus on Git first.

How do I use Git?

- It ships out of the box for Mac and Linux machines.
- Easy to install on Windows*.
- CLI, VSCode and RStudio.

Why would I use Git?

- A lot of people use it.
- A lot of systems are integrated with it.
- It has more features than OneDrive.
- Local system.

Ideal everyday Git

- Issue
- Make and save changes
- Stage the changes you want to keep
- Commit

Commit messages

Everytime you commit, you have to create a message. It's like an email to yourself:

- Timestamp
- Author
- Subject line
- Body
- Line by line changes in staged files

Git is flexible

Git it's just a bunch of notes to yourself. You can:

- Work 9-5.
- Stage everything
- Commit message: "my update"

- Work on *one* issue
- Stage only changes related to that issue
- Write informative commit message

What's the point?

- You have something that works on Monday
- You work all week to add something to it
- You realize on Friday that you broke it

Now you can:

- Check commit messages to see at which commit you broke it
- Send the entire project back to the commit before that

Why would you need branches?

Motivation:

- `lm` + base R plot.
- You need *ggplot*
- You want `glm`

You want to work in parallel.

Branch workflow

- Work on `main` branch
- Implement `ggplot`
- Commit (c_2)

- Create an `experimental` branch
- Implement `glm`
- Commit (c_1)

Branch workflow continued

- Merge (this commits, c_3)
- Test

Either:

- It worked
- Go on based on c_3

- It didn't work
- Revert to c_2

What exactly does Git do?

- `.git` directory
- Each commit has a unique ID.
- `HEAD` pointer

Limitations

- Works best for text files (.R, .txt, ...)
- Not great for large files

Collaboration

Simple syncing has drawbacks.
In previous example:

Nico:

- Expect `lm`
- Change plot
- End up with `glm`

Humphrey:

- Changes to `glm`

Humphrey's changes might have broken my version

GitHub

Solution:

- Version of the repo in the cloud
- Also local versions
- Stablish a policy for updating cloud repo using local repos

The cloud version is called a *remote* repo.

GitHub:

- Cloud service to host remote repos
- Provides a URL

Linking a remote repo

Hey Git: the repo at this URL is going to be the remote repo for this local repo.

- Both local and remote may have several branches
- Associate to each local branch a remote branch, called the *upstream*
- `nico-branch` and `humphrey-branch` vs `main` and `experimental`

Workflow

- Humphrey made changes, so `experimental` is behind
- I checkout `experimental` and *pull* `humphrey-branch`
- I work on `main`. It is now ahead of `nico-branch`
- When I'm ready to share, I *push* to `nico-branch`.

Forks

What if Humphrey asks Cleto (an external collaborator) for help?

- Cleto can *fork* the GitHub repo.
- Different repo, but can sync to the original one.
- Cleto creates `cleto-branch` and works on it
- Cleto creates a *pull request* to the `humphrey-branch` of the original GH repo
- Humphrey (or me) may or may not accept is

Issues and other PM

Large open source projects need etiquete

- Users open issues (bugs, enhancements, documentation)
- Maintainers filter, organize and assign them to developers
- Developers may make commits and reference the issue in the commit messages
- External collaborators can fork, make changes and pull requests

GitHub Actions

(Advanced)
How does `tidyverse` release a new version? Through GH `Actions`.

- Wickham tells GH: this is the official `tidyverse` repo
- Wickham tells GH: whenever I make a commit to this repo, build the R package and submit it to CRAN

Demo!