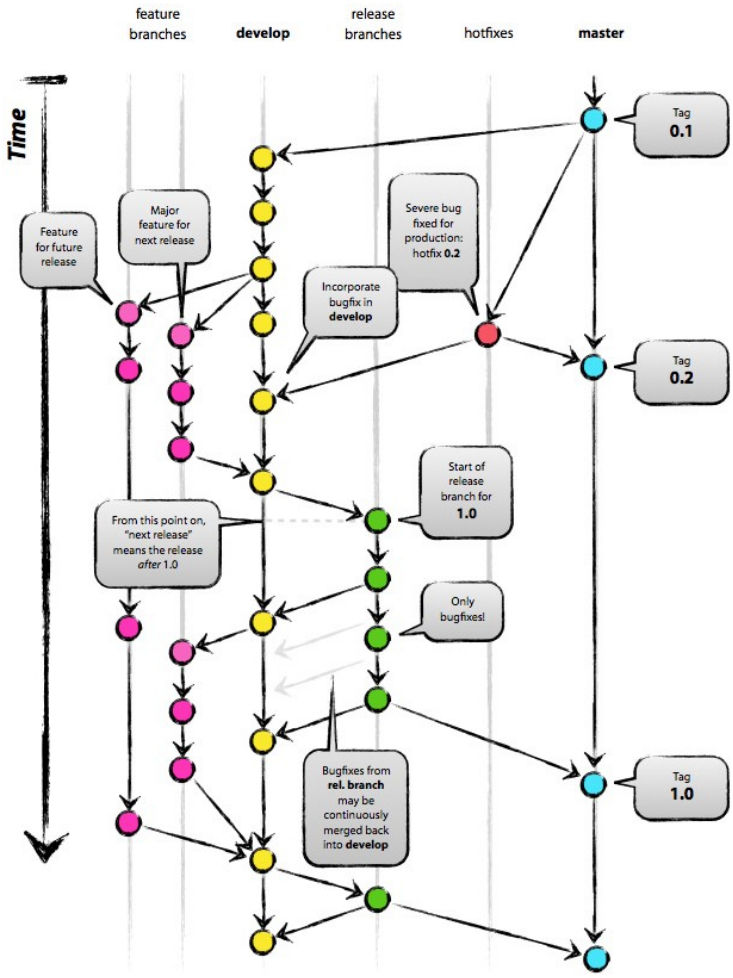
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency —https://git-scm.com/. Git is the most common version control software used today because It is different from other source control software out there. What makes it the best is that it is distributed, meaning that every collaborator have a full clone of the repository and may works with a remote repository to merge changes using uniq algorithms like rebase an so on. If you would like to know more about git, here is a good place to start.



We use git as a command line to create repositories and make changes, merge to an upstream repositories and more. There are many tools built on top of git, one of them is Github. Github is a software that allows to maximize the collaboration between different users, providing many features on top of git like issues, wiki, and many others. Git allows communities to strive and open source projects to be served. including feature branch management and pull requests which is what this post is all about.

Pull Request (PR) is a **Github** feature that allows users to collaborate better together. Usually in source control software there is a main branch that describes production—in git it is the master branch. One can create a feature branch and make changes there, then issue a PR to the master branch for someone else to review, approve or reject the changes and merge to the master branch eventually. Discussing potential improvements is crucial to write and maintain a high quality repository.

Now PR is a Github feature, but it relies on the git feature named request-pull. Git supports PRs by providing a feature that lets you request a pull from one working set to another from different branches or different repositories. The command lets you review a summarise of the feature branch with all the commits and changes that are going to be merged. When working with PRs it is recommended to work in a feature branch and not on the master and make PRs to the master. I will show only examples that describes the best practice. The request pull command line tool lets you review all the changes you did on the feature branch,

The request-pull command line arguments is not that straight forward. git request-pull --help will show you the full usage and specifications for the command.

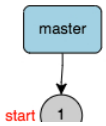
The basic signature is git request-pull [-p] <start> <url> [<end>]

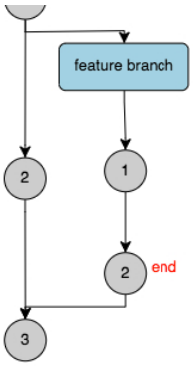
- [-p]—Run request-pull without that option will output a summarize of changed files. -p is more verbose and will output all the changes that have been made from the split commit to the end one.

- <start>—Is the starting point you want to merge to. Most of the time we will given it the master branch and git will calculate the start commit by it's own. The start commit is the common ancestor the feature branch splitted from.

- <url>—is the repository to compare to. It can be a local repository and it can be a remote one.

- <end>—The end point we want to stop comparing to. Usually we won't state the end commit because we would like to merge all recent changes. To make things simple we won't show this usage.





In order to merge the latest feature branch changes to the master branch, first we want to make sure we are on the feature branch as this is a request from the feature branch to be merged to the master. Go to the feature branch with git checkout -b [branch-name]. Then we can run git request-pull master ./ to run a comparison from the feature branch to the master. Notice that we state the master as the start and the local repository to compare so to get an accurate results, we need to be pulled from the latest master.

Let's examine such an output:

```
➜  alonn24.github.io git:(visualizations) g request-pull master ./
The following changes since commit fc39a42f9720969b2d956c7f2e88ba9ad669a8ec:

  web-components: add polymer word counter (2018-06-08 01:08:15 +0300)

are available in the git repository at:

  ./

for you to fetch changes up to 6521c28c5d08a2b6a4ca11d8e43c2cf22250e2b6:

  visualizations initial commit (2018-06-13 21:42:18 +0300)

----------------------------------------------------------------
alony (1):
      visualizations initial commit

 visualizations/package.json                | 26 ++++++++++
 visualizations/src/d3/html-data-binding.ts | 18 ++++++++
 visualizations/src/d3/svg-graphics.ts      | 25 ++++++++++
 visualizations/src/gojs/simple-diagram.ts  | 36 ++++++++++++++++
 visualizations/src/index.html              | 15 ++++++
 visualizations/src/index.ts                |  8 ++++
 visualizations/src/tsconfig.json           | 14 ++++++
 visualizations/webpack.config.js           | 38 ++++++++++++++++
 8 files changed, 180 insertions(+)
 create mode 100644 visualizations/package.json
 create mode 100644 visualizations/src/d3/html-data-binding.ts
 create mode 100644 visualizations/src/d3/svg-graphics.ts
 create mode 100644 visualizations/src/gojs/simple-diagram.ts
 create mode 100644 visualizations/src/index.html
 create mode 100644 visualizations/src/index.ts
 create mode 100644 visualizations/src/tsconfig.json
 create mode 100644 visualizations/webpack.config.js
➜  alonn24.github.io git:(visualizations)
```

We can separate the command output into 2 parts:

1. The commits details—we can see the calculated start commit (SHA1 + description) and the end commit (SHA1 + description) following by all the commits in between. In our case there is only one commit "visualizations initial commit"

2. The files that have been changed—we can see that we have 8 files changed with 180 insertions. If we would run the command with the -p flag we would also see the changes and not only a summary.

In the Github PR request page, we can see that the changes details display there is exactly what we see here, with small modifications. To see all the changes for the PR we can run the above example with -p, but to see the changes for each commit we will have to run another command given the exact commit gif diff [SHA1] [SHA1]^.

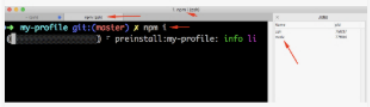
Up until now we worked with git features alone, meaning that pull requests is a git term and the CLI supports everything that is shown in the UI implementations such as Github. Now to issue a real PR request in Github we need to use the Github public API. HUB is a great tool that lets you interact with Github API and make changes to the repository from the command line, for example you can create a PR in Github by running hub pull-request.

That's it, from now on I'm doing my PRs from the command line only.

Git # Github # Terminal # Git Pr # Command Line

Continue the discussion

More by Alon Yehezkel



Don't Lose Your Head With iTerm2

Alon Yehezkel
Jun 12

Git

JS Template Literals on Steroids

Alon Yehezkel
Jun 26

Javascript

Minimize Your Tests Boilerplate

Alon Yehezkel

Javascript

ML Programming Made [too much] Easy — Part 2

Alon Yehezkel
Jun 07

Machine Learning

Hackernoon Newsletter curates great stories by real tech professionals

Get solid gold sent to your inbox. Every week!

Email Address *

First Name

Last Name

TOPICS OF INTEREST

- [x] Software Development
- [x] Blockchain Crypto
- [x] General Tech
- [x] Best of Hacker Noon

Get great stories by email

More Related Stories

15 Tips to Enhance your Github Flow

Gabo Esquivel
May 11

Git

12 cool things you can do with GitHub

David Gilbertson

Github

cmd meltdown: I know that command!

Sourav Ray
Mar 16

Terminal

Developer's Toolkit: 4 linux command-line tools for debugging everyday issues

Andy Macdonald
Apr 15

Software Development