



# GitHub

Οδηγός χρήσης GitHub σε συστήματα Windows

# Περιεχόμενα

Με links

- ❖ Ενότητα 1<sup>η</sup> : [GitHub και Git.](#)
- ❖ Ενότητα 2<sup>η</sup> : [Εισαγωγή στο GitHub και βασικές χρήσεις μέσω της ιστοσελίδας.](#)
- ❖ Ενότητα 3<sup>η</sup> : [Git Clients και λειτουργίες.](#)
- ❖ Ενότητα 4<sup>η</sup> : [Κατανόηση του GitHub flow.](#)
- ❖ Ενότητα 5<sup>η</sup> : [Permissions \(Δικαιώματα\).](#)
- ❖ Ενότητα 6<sup>η</sup> : Εντολές τερματικού.
- ❖ [Πηγές](#)

# Git & GitHub

Ποια είναι η διαφορά μεταξύ Git και GitHub?



# Git

## Εργαλείο για Developers!

- Το git είναι ένα σύστημα ελέγχου εκδόσεων (version-control system), με έμφαση στην ταχύτητα, στην ακεραιότητα των δεδομένων και στην υποστήριξη για κατανεμημένες μη-γραμμικές ροές εργασίας.
- Το git αναπτύχθηκε αρχικά από τον Linus Torvalds για την ανάπτυξη του πυρήνα Linux το 2005.
- Από τότε έχει γίνει το πιο διαδεδομένο σύστημα ελέγχου εκδόσεων για ανάπτυξη λογισμικού.



# Git και GitHub

- Όπως προαναφέρθηκε το Git είναι ένα σύστημα παρακολούθησης εκδοχών ενός προγράμματος, ένα εργαλείο για να διαχειριζόμαστε το ιστορικό των πηγαίων κωδίκων.
- Από την άλλη, το GitHub αποτελεί μια υπηρεσία hosting για Git repositories.
- Οπότε, το Git είναι ένα εργαλείο ενώ το GitHub είναι μια υπηρεσία για projects που χρησιμοποιούν Git.

# Εισαγωγικά

Εισαγωγή στο GitHub και βασικές χρήσεις μέσω της ιστοσελίδας.



# Σε αυτό το στάδιο θα δούμε...

- Πως να δημιουργούμε και να χρησιμοποιούμε ένα repository.
- Πως να ξεκινάμε και να διαχειριζόμαστε ένα branch.
- Πως να κάνουμε αλλαγές σε αρχεία και να τα προωθούμε στο GitHub ως commits.
- Πως να ανοίγουμε και να συγχωνεύουμε ένα pull request.



# Τι είναι το GitHub?

- Το GitHub είναι μια πλατφόρμα, όπου «φυλάσσεται» ένας κώδικας.
- Η «φύλαξη» του γίνεται για δύο σκοπούς:
  - Τον έλεγχο των διαφόρων εκδόσεων του λογισμικού.
  - Την συνεργασία των προγραμματιστών
- Βασικές έννοιες του GitHub: repositories, branches, commits και Pull Requests.
- Στόχος: να δημιουργήσουμε το δικό μας Hello World repository, και να μάθουμε πως λειτουργεί η ροή εργασίας των Pull Requests του GitHub.





# Δεν χρειαζόμαστε κώδικα!

9

Για να ολοκληρώσουμε αυτό το tutorial χρειαζόμαστε έναν λογαριασμό GitHub και σύνδεση στο internet.

Δεν χρειάζεται να γνωρίζουμε κώδικα, να χρησιμοποιήσουμε την γραμμή εντολών ή να έχουμε εγκατεστημένο το Git!



# Βήμα 1<sup>ο</sup>

## Δημιουργία Repository

- Συνήθως για την οργάνωση ενός project χρειαζόμαστε ένα και μόνον repository.
- Τα repositories μπορούν να περιέχουν φακέλους, αρχεία, εικόνες, βίντεο, λογιστικά φύλλα (spreadsheets) και συλλογές δεδομένων.
- Συνιστάται τα repositories να περιέχουν ένα αρχείο README, ή γενικά κάποιο αρχείο με πληροφορίες για το project.
- Το GitHub διευκολύνει την εισαγωγή ενός README, καθώς το τοποθετεί κατά την δημιουργία του repository.
- Επίσης το GitHub μπορεί να προσθέσει και άλλα αρχεία, όπως ένα license file.



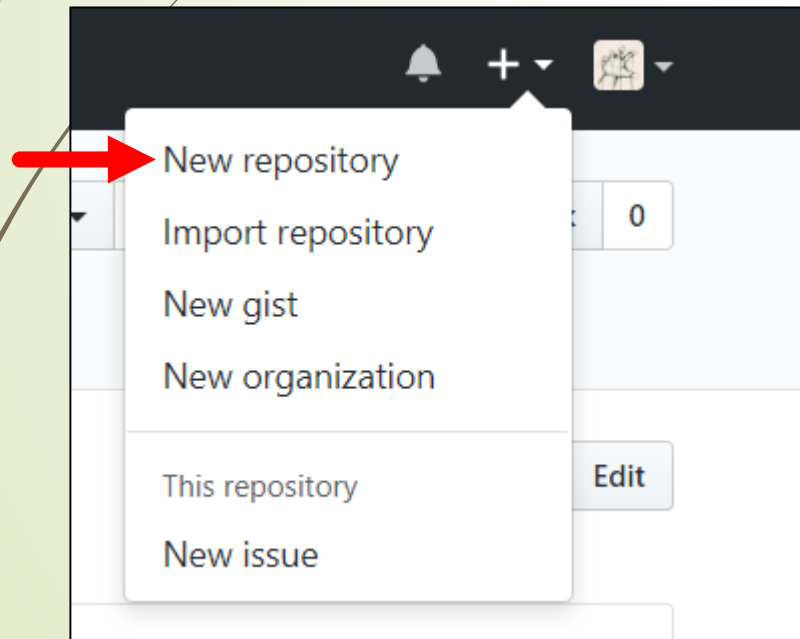
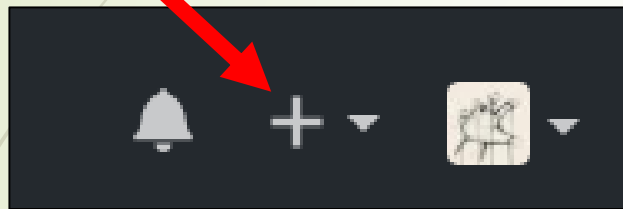
# Βήμα 1<sup>ο</sup>

## Δημιουργώντας ένα repository

1. Στην πάνω δεξιά γωνία της σελίδας, δίπλα στο εικονίδιο χρήστη, υπάρχει ένα **+**. Πατώντας το εμφανίζεται μια λίστα με επιλογές. Πατώντας **New repository**, η διαδικασία ξεκινά.
2. Μεταγόμεστε στην σελίδα δημιουργίας, όπου καλούμαστε να συμπληρώσουμε τον ιδιοκτήτη του repository, το όνομα και μια προαιρετική περιγραφή.
3. Στην συνέχεια επιλέγουμε εάν το repository μας θα είναι ιδιωτικό (private) ή δημόσιο (public).
4. Τέλος, επιλέγουμε εάν θέλουμε να προσθέσουμε το προαναφερθέν README. Στο πλαίσιο παρατηρούμε και άλλες δύο επιλογές, οι οποίες θα σχολιαστούν αργότερα.
5. Για την ολοκλήρωση της διαδικασίας, πατάμε το κουμπί **Create repository**.

# Βήμα 1<sup>ο</sup>

## Στιγμιότυπα - 1



### Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

Repository name

Act862

Choose another owner

Act862

PyLamGR

Need inspiration? How about [solid-tribble](#).

☒ Public

Anyone can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None

Add a license: None



Create repository

# Βήμα 1<sup>ο</sup>


## Στιγμιότυπα - 2

### Create a new repository


A repository contains all the files for your project, including the revision history.

Owner

Repository name

 Act862 ▾


 / 

hello 


Great repository names are short and memorable. Need inspiration? How about [solid-tribble](#).

Description (optional)

My new repository

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**


You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

 | 

Add a license: **None** ▾ 

Create repository

# Βήμα 1<sup>ο</sup>

## Στιγμιότυπα - 3

The screenshot shows a GitHub repository page for 'Act862 / hello-world'. At the top, there are navigation links for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. Below these, the repository is described as 'Just a beginner's repository <3' with an 'Edit' button. A summary bar shows '1 commit', '1 branch', '0 releases', and '1 contributor'. Below this, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button. The commit history shows 'Act862 Initial commit' as the latest commit, 20 minutes ago. Below the commit history, the 'README.md' file is displayed, containing the text 'hello-world' and 'Just a beginner's repository <3'.

Act862 / hello-world

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Just a beginner's repository <3 [Edit](#)

[Add topics](#)

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

Act862 Initial commit Latest commit 42dd20e 20 minutes ago

[README.md](#) Initial commit 20 minutes ago

[README.md](#)

# hello-world

Just a beginner's repository <3

# Βήμα 2<sup>ο</sup>

## Δημιουργία Branch - 1

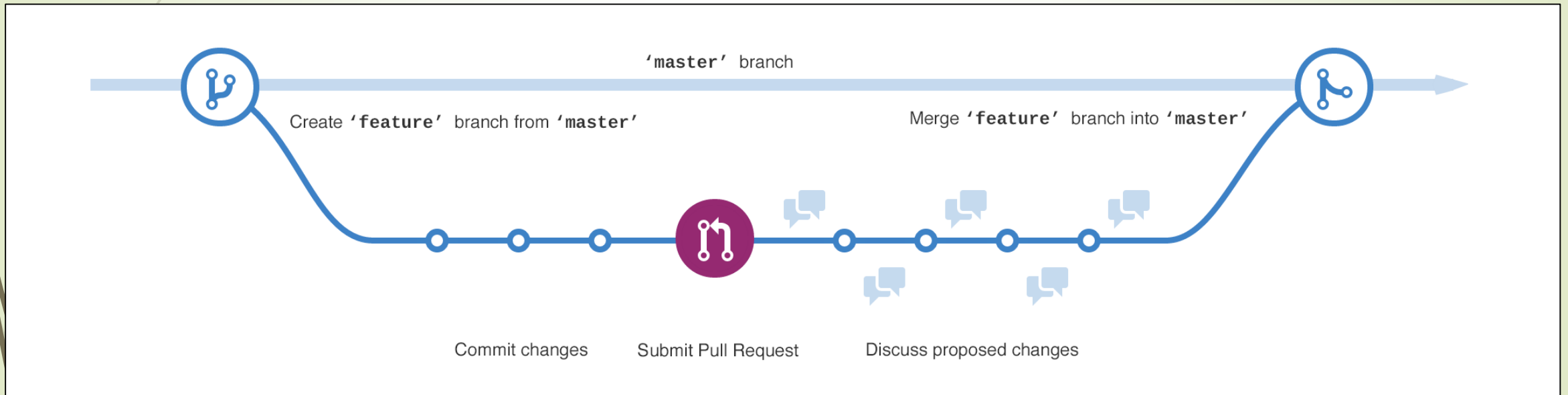
- Το branching είναι η μέθοδος με την οποία μπορούμε να δουλεύουμε την ίδια στιγμή, πάνω σε διαφορετικές εκδοχές του repository.
- Από την στιγμή της δημιουργίας του το repository έχει μόνο ένα branch το οποίο ονομάζεται **master branch**.
- Χρησιμοποιούμε branches για να πειραματιστούμε και να κάνουμε αλλαγές στον κώδικα, πριν τα αποθηκεύσουμε στο master branch.
- Όταν δημιουργούμε ένα branch από το master branch, δημιουργούμε ένα αντίγραφο του master όπως ήταν μέχρι εκείνη την στιγμή.
- Αν κάποιος άλλος έκανε αλλαγές στο master branch, όσο εμείς δουλεύαμε στο δικό μας branch, μπορούμε απλά να τραβήξουμε τις αλλαγές, και να συνεχίσουμε ανενόχλητοι.





## Βήμα 2<sup>ο</sup>

### Χρονοδιάγραμμα





## Βήμα 2<sup>ο</sup>

### Παράδειγμα κατανόησης branches

- Έστω το αρχείο: **adder.c**
- Αρχικά το repository περιέχει τον adder.c με υποστήριξη μόνο για ακραίους. Αν κάποιος θέλει να προσθέσει λειτουργίες και για πραγματικούς, μπορεί να δημιουργήσει ένα νέο branch, με βάση το αρχικό, και να πειραματιστεί εκεί.
- Στην GitHub, οι developers χρησιμοποιούν branches για να μελετούν bugs και να εργάζονται ξεχωριστά από το master (production) branch, το οποίο είναι και αυτό που έρχεται σε επαφή με εμάς. Όταν μια αλλαγή είναι έτοιμη, τότε συγχωνεύεται στο master branch, έχουμε δηλαδή merge.

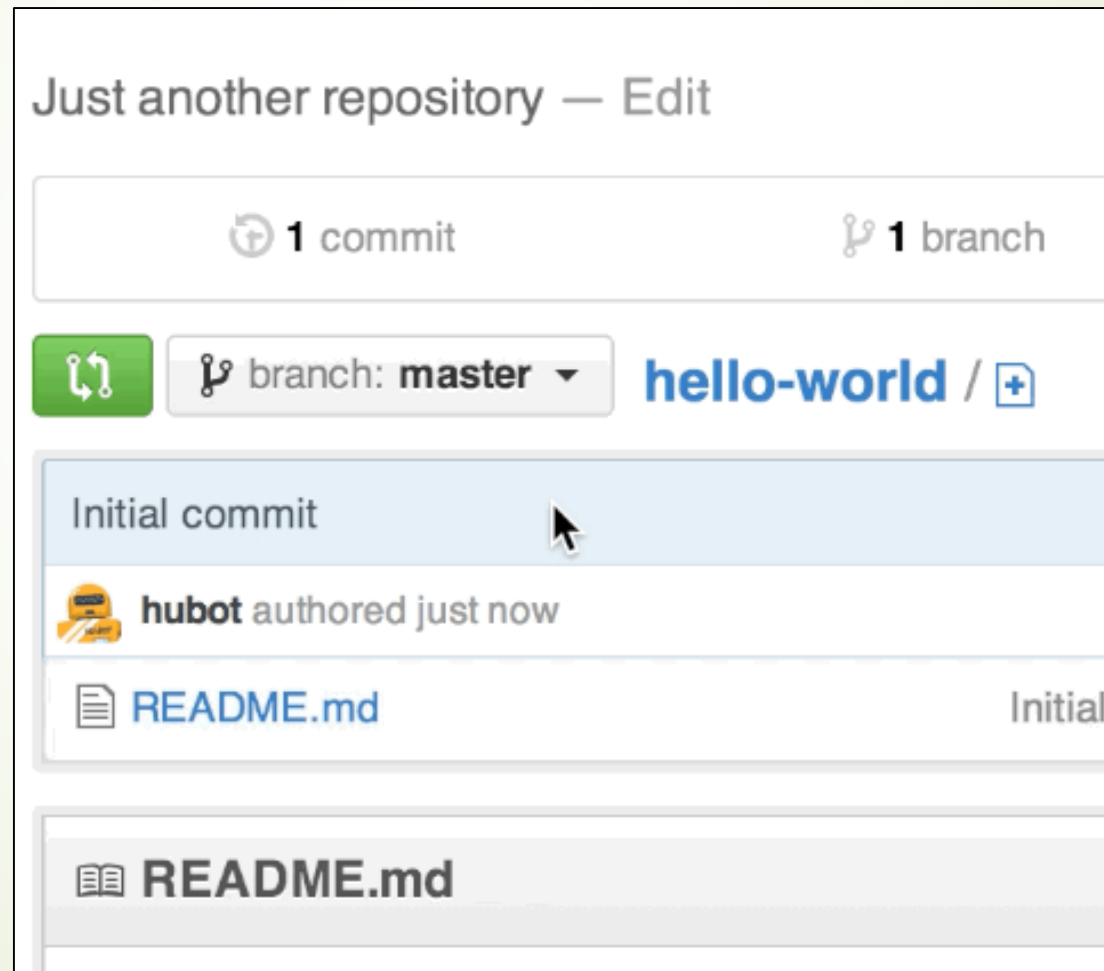
## Βήμα 2<sup>ο</sup>

### Δημιουργία branch - 2

1. Πηγαίνουμε στο repository μας, στην προκειμένη το hello-world.
  2. Επιλέγουμε την αναδιπλωμένη λίστα με τίτλο: **branch: master**.
  3. Στο πλαίσιο κειμένου γράφουμε το όνομα του branch που θέλουμε να δημιουργήσουμε.
  4. Επιλέγουμε το μπλε **Create branch** κουμπί ή πατάμε «Enter» από το πληκτρολόγιο.
- Τώρα έχουμε δύο branches. Ένα **master** και ένα **readme-edits**. Μοιάζουν το ίδιο, αλλά όχι για πολύ! Στην συνέχεια θα προσθέσουμε τις δικές μας τροποποιήσεις στο καινούργιο branch.

## Βήμα 2<sup>ο</sup>

Παράδειγμα με gif



## Βήμα 3<sup>ο</sup>

### Δημιουργία και παραπομπή αλλαγών

- Πλέον, μπορούμε να προχωρήσουμε σε αλλαγές πάνω στο `readme-edits`.
- Στο GitHub οι αποθηκευμένες τροποποιήσεις ονομάζονται **commits**.
- Κάθε commit ακολουθείται από ένα *commit message*, το οποίο είναι μια περιγραφή.
- Στις περιγραφές εξηγούνται οι λόγοι που οδήγησαν στις τροποποιήσεις.
- Τα commit messages αποτελούν το ιστορικό των τροποποιήσεων, ώστε οποιοσδήποτε τρίτος να καταλαβαίνει τι έχει αλλάξει και τι ανάγκη καλύφθηκε.

## Βήμα 3<sup>ο</sup>

Εφαρμογή αλλαγών και αποθήκευση τους.

1. Επιλέγουμε το `README.md` του branch.
  2. Επιλέγουμε το κουμπί με το **pencil icon**, το οποίο βρίσκεται στην πάνω δεξιά γωνία της προεπισκόπησης του αρχείου.
  3. Μεταγόμαστε σε έναν κειμενογράφο.
  4. Στο πλαίσιο με τίτλο `Commit changes` γράφουμε το `commit message`, περιγράφοντας τις αλλαγές που κάναμε.
  5. Πατάμε το κουμπί: **Commit changes**.
- Οι αλλαγές που κάναμε γίνονται μόνο στο `README` του `readme-edits` branch, οπότε τώρα περιέχει διαφορετικά πράγματα μέσα από το `master` branch.

## Βήμα 3<sup>ο</sup> (κατά την χρήση Git Client Services)

### Push to & Pull from

- Θεμελιώδεις λειτουργίες του Git αποτελούν οι **Push to** και η **Pull from**.
- Το **Push to** πραγματοποιεί το δικαίωμα εγγραφής ενός χρήστη (write rights).
- Αντίστοιχα το **Pull from** πραγματοποιεί το δικαίωμα ανάγνωσης (read rights).
- Δικαιώματα push και pull έχει μόνο ο ιδιοκτήτης ενός repository και οι συνεργάτες του.
- Αν και θα αναλυθούν σε βάθος αργότερα, η ροή εφαρμογής των push και pull είναι η εξής:
  - Pull από κάποιο branch → τροποποίηση → commit & commenting → push στο ίδιο branch → pull request, σε περίπτωση που δεν έχουμε δικαίωμα άμεσης εγγραφής.
- Υπάρχει και μια τρίτη θεμελιώδη λειτουργία, η οποία θα αναλυθεί επίσης αργότερα. Η **fork**, η οποία αντιστοιχεί στην διαδικασία της αντιγραφής.



## Βήμα 4<sup>ο</sup>

### Pull Request

- Εφόσον πλέον έχουμε αλλαγές σε ένα branch, εκτός του master, μπορούμε να αιτηθούμε ένα **pull request**.
- Τα Pull Requests είναι η καρδιά της συνεργασίας στο GitHub. Όταν ανοίγουμε ένα νέο pull request, προτείνουμε τις αλλαγές μας και ζητάμε από κάποιον να τις εγκρίνει.
- Αφού κριθούν, οι αλλαγές μας θα συγχωνευτούν με τα branches όσων τις αποδεχτούν.
- Τα pull requests εμφανίζουν διαφορές μεταξύ των δύο branches.
- Οι προσθήκες φαίνονται με πράσινο χρώμα.
- Οι αφαιρέσεις φαίνονται με κόκκινο χρώμα.



## Βήμα 4<sup>ο</sup>

### Discussions & self-requests

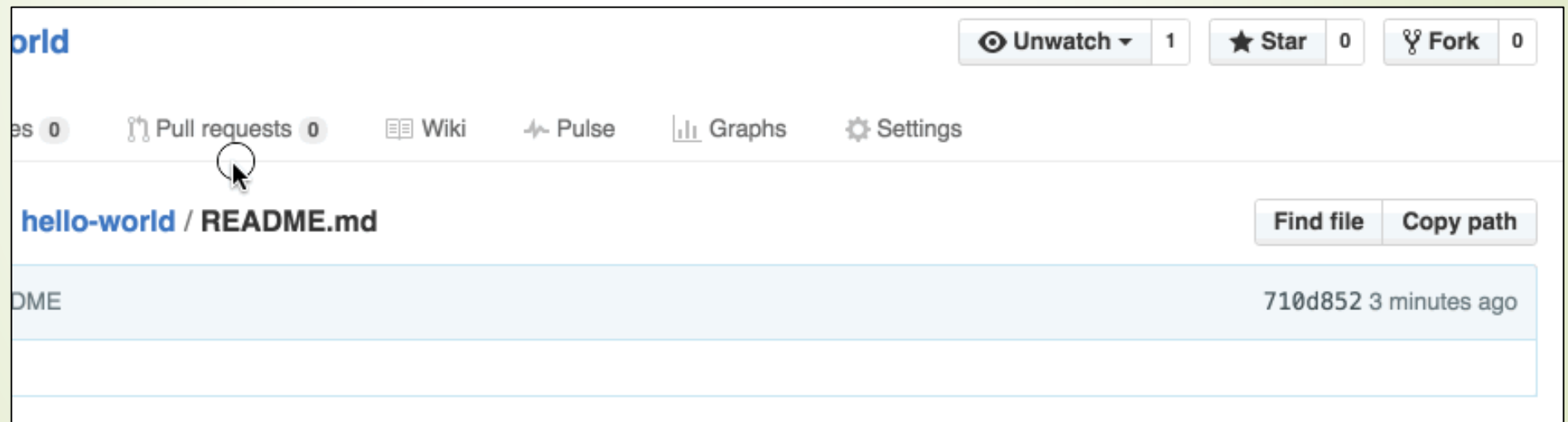
- Όταν κάνουμε το commit, μπορούμε να αιτηθούμε ένα pull request και να ξεκινήσουμε μια συζήτηση πριν ολοκληρωθεί ο κώδικας.
- Χρησιμοποιώντας το @mention system του GitHub στα Pull Request μας, ώστε να ζητήσουμε για feedback από συγκεκριμένους ανθρώπους ή μέλη της ομάδας.
- Μπορούμε ακόμη να κάνουμε pull request στο δικό μας το repository και να τις συγχωνεύσουμε.
- Η παραπάνω τακτική αποτελεί τον εύκολο τρόπο ώστε να μάθουμε πως λειτουργούν τα pull requests, πριν τα εφαρμόσουμε σε ομαδικά projects.



## Βήμα 4<sup>ο</sup>

### Παραδείγματα με εικόνες (1/5)

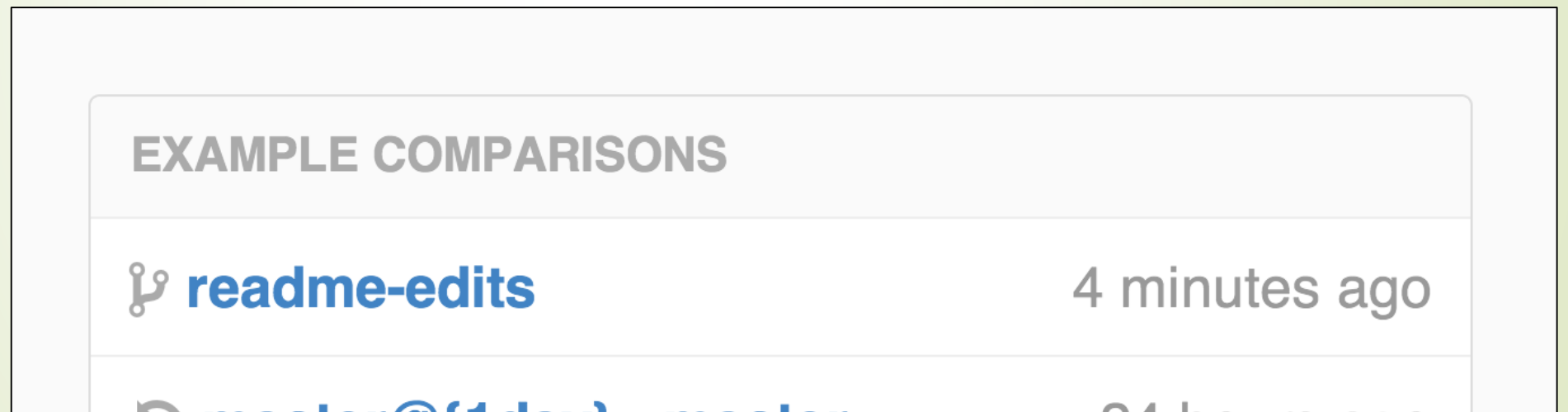
- Επιλέγουμε την καρτέλα **Pull Request**. Από την σελίδα που εμφανίζεται επιλέγουμε το πράσινο κουμπί που γράφει **New pull request**.



## Βήμα 4<sup>ο</sup>

### Παραδείγματα με εικόνες (2/5)

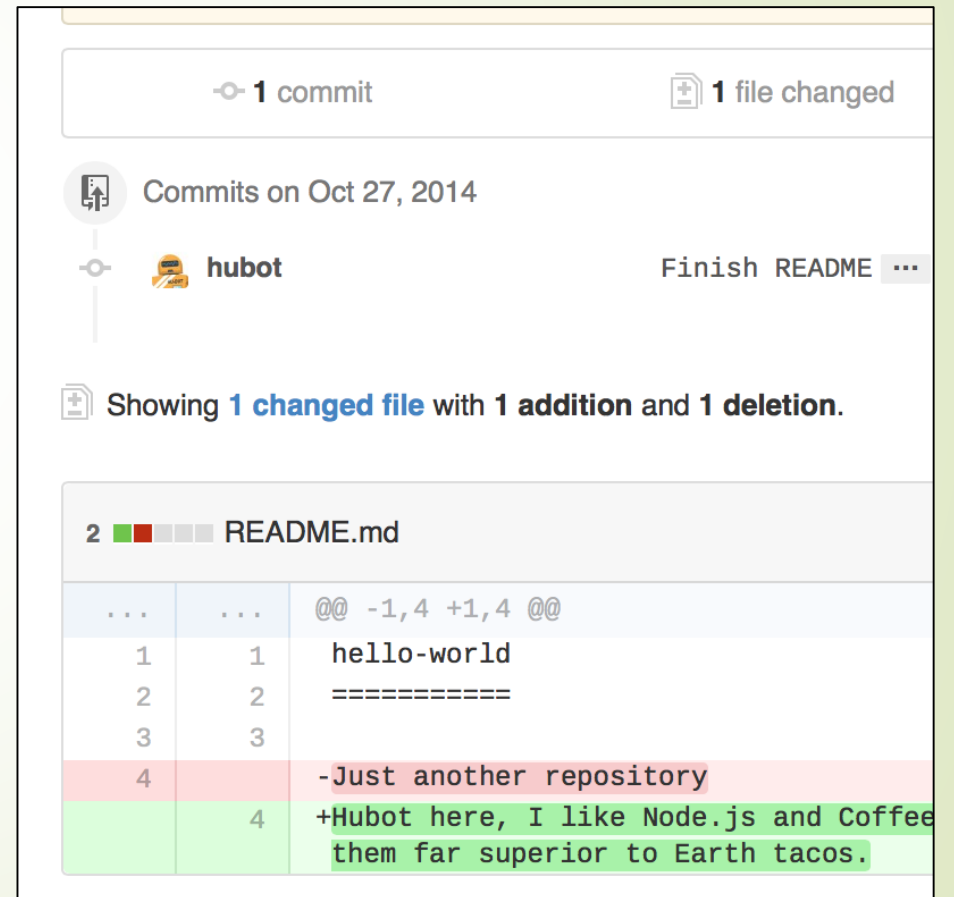
- Στο κουτί **Example Comparisons**, επιλέγουμε το branch που επιθυμούμε, όπου στην προκειμένη περίπτωση είναι το **readme-edits**. Το συγκρίνουμε με το master branch (το οποίο είναι το αρχικό).



## Βήμα 4<sup>ο</sup>

### Παραδείγματα με εικόνες (3/5)

- Ελέγχουμε τις αλλαγές που έγιναν στο αρχείο. Σε αυτό το βήμα πρέπει να είμαστε πολύ προσεκτικοί, ώστε να βεβαιωθούμε για τις αλλαγές που πρόκειται να προτείνουμε.



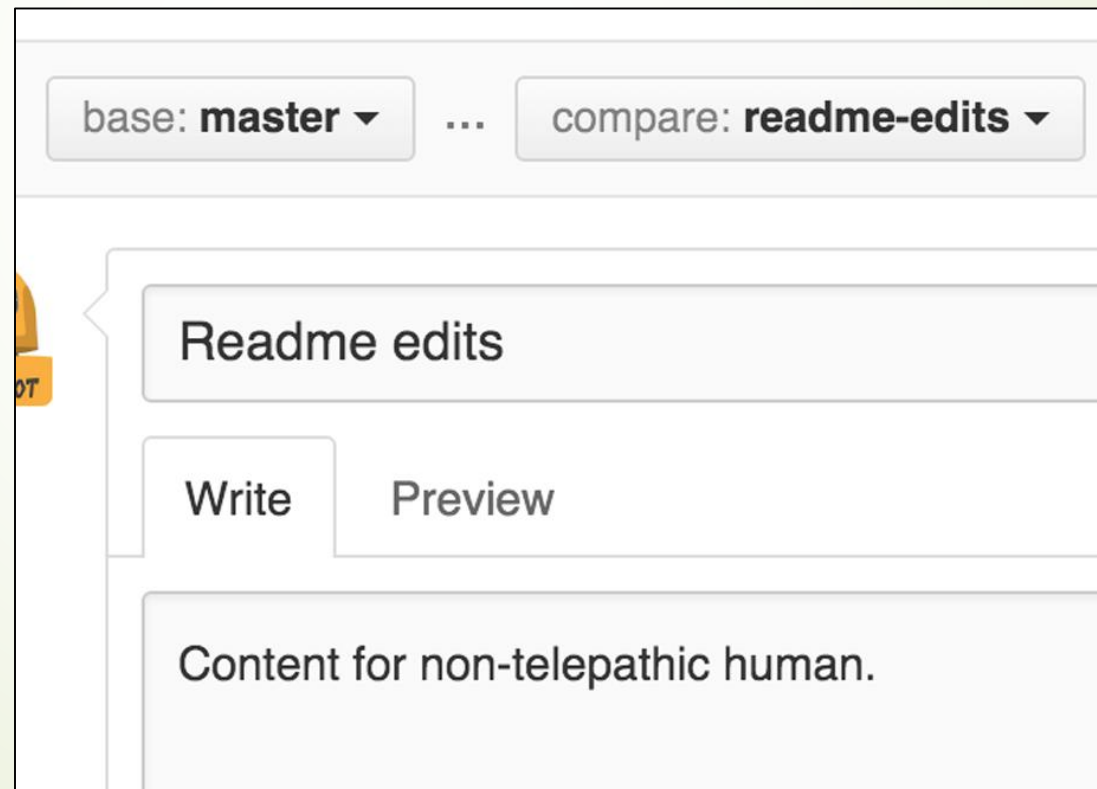
The screenshot shows a GitHub commit interface. At the top, it indicates "1 commit" and "1 file changed". Below this, it shows the commit date "Oct 27, 2014" and the user "hubot". A button labeled "Finish README" is visible. The main section shows the diff for "README.md", indicating "1 changed file" with "1 addition" and "1 deletion". The diff content is as follows:

...	...	@@ -1,4 +1,4 @@
1	1	hello-world
2	2	=====
3	3	
4		-Just another repository
	4	+Hubot here, I like Node.js and Coffee them far superior to Earth tacos.

## Βήμα 4<sup>ο</sup>

### Παραδείγματα με εικόνες (4/5)

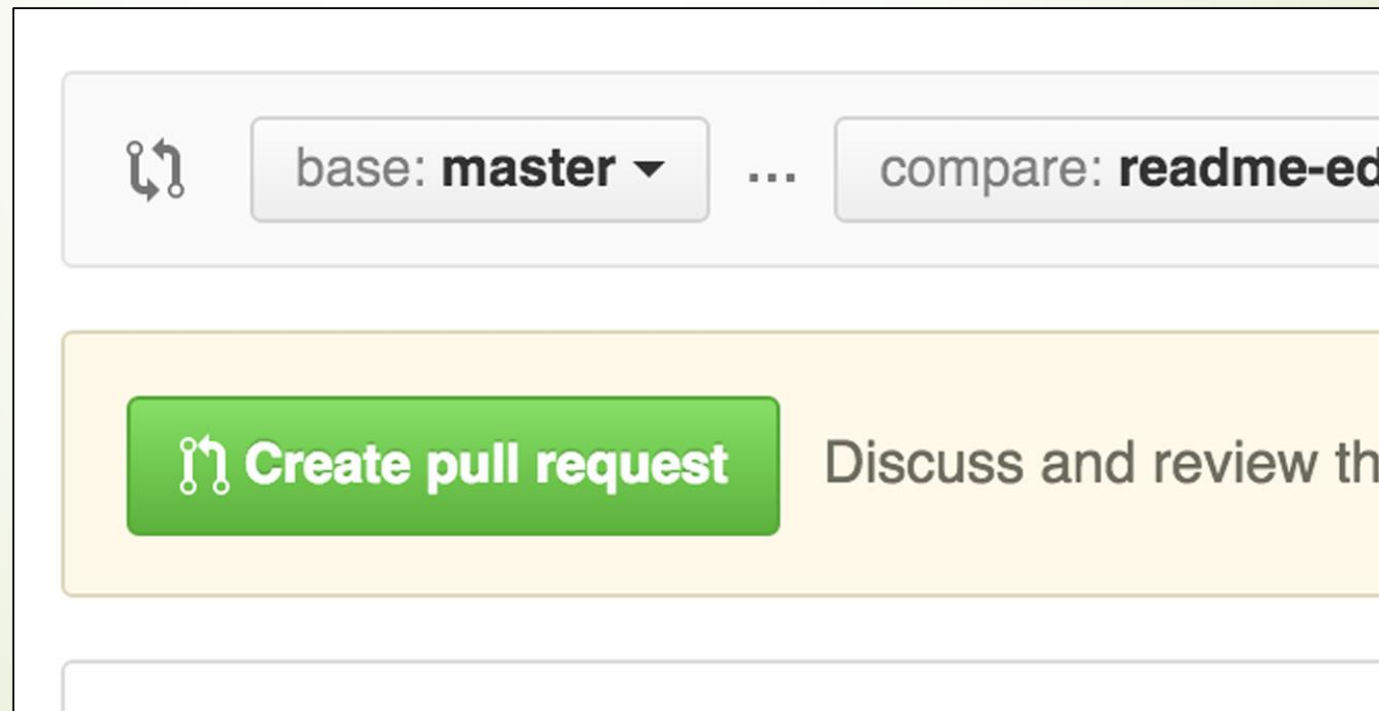
- Χάριν ομορφιάς, δίνουμε έναν τίτλο και μια σύντομη περιγραφή στις αλλαγές.



## Βήμα 4<sup>ο</sup>

Παραδείγματα με εικόνες (5/5)

- Εφόσον είμαστε ικανοποιημένοι με τις αλλαγές που παρουσιάστηκαν, πατάμε το **Create Pull Request**.



# Βήμα 5<sup>ο</sup>

## Συγχωνεύοντας ένα Pull Request

- Σε αυτό το τελικό σημείο, είναι ώρα να συγκεντρώσουμε όλες τις αλλαγές σε ένα μέρος, συγχωνεύοντας το readme-edits branch στο master branch.
- Η διαδικασία είναι απλή:
  1. Πατάμε την επιλογή **Merge pull request**, για να συγχωνεύσουμε τις αλλαγές στο master.
  2. Πατάμε το **Confirm merge**. Προσοχή!! Πρέπει να είμαστε σίγουροι πως οι αλλαγές δεν πρόκειται να προκαλέσουμε ανεπανόρθωτη ζημιά στον κώδικα μας.
  3. Αφού ολοκληρωθεί η συγχώνευση μπορούμε άφοβα να διαγράψουμε το branch, αφού οι αλλαγές έχουν ενσωματωθεί στο master branch.

## Βήμα 5<sup>ο</sup>

### Παραδείγματα με εικόνες

- Το GitHub μας ενημερώνει εάν το merge μπορεί να γίνει με ασφάλεια.



**This branch has no conflicts with the base branch**

Merging can be performed automatically.



**Merge pull request**

You can also [open this in GitHub Desktop](#) or view [command line instructions](#).



## Βήμα 5<sup>ο</sup>

### Παραδείγματα με εικόνες

- Επίσης μας ενημερώνει, μετά την ολοκλήρωση του merge σχετικά με την δυνατότητα να διαγράψουμε το branch.



#### **Pull request successfully merged and closed**

You're all set—the `readme-edits` branch can be safely deleted.

 **Delete branch**



# Τέλος εισαγωγής

## Ολοκλήρωση

- Έχοντας ολοκληρώσει αυτό το tutorial μάθαμε πως να δημιουργούμε ένα project και να κάνουμε pull requests.
- Επίσης μάθαμε:
  1. Πως δημιουργείται ένα repository ανοικτού κώδικα.
  2. Πως να ξεκινάμε και να διαχειριζόμαστε ένα branch.
  3. Πως να επεμβαίνουμε σε αρχεία και να ενσωματώνουμε τις αλλαγές αυτές στο GitHub μέσω των commits.
  4. Πως να κάνουμε και να αποδεχόμαστε ένα Pull Request.
- Όμως δεν τελειώσαμε εδώ, διότι αυτά ήταν μόνο τα εισαγωγικά!



# Git Clients

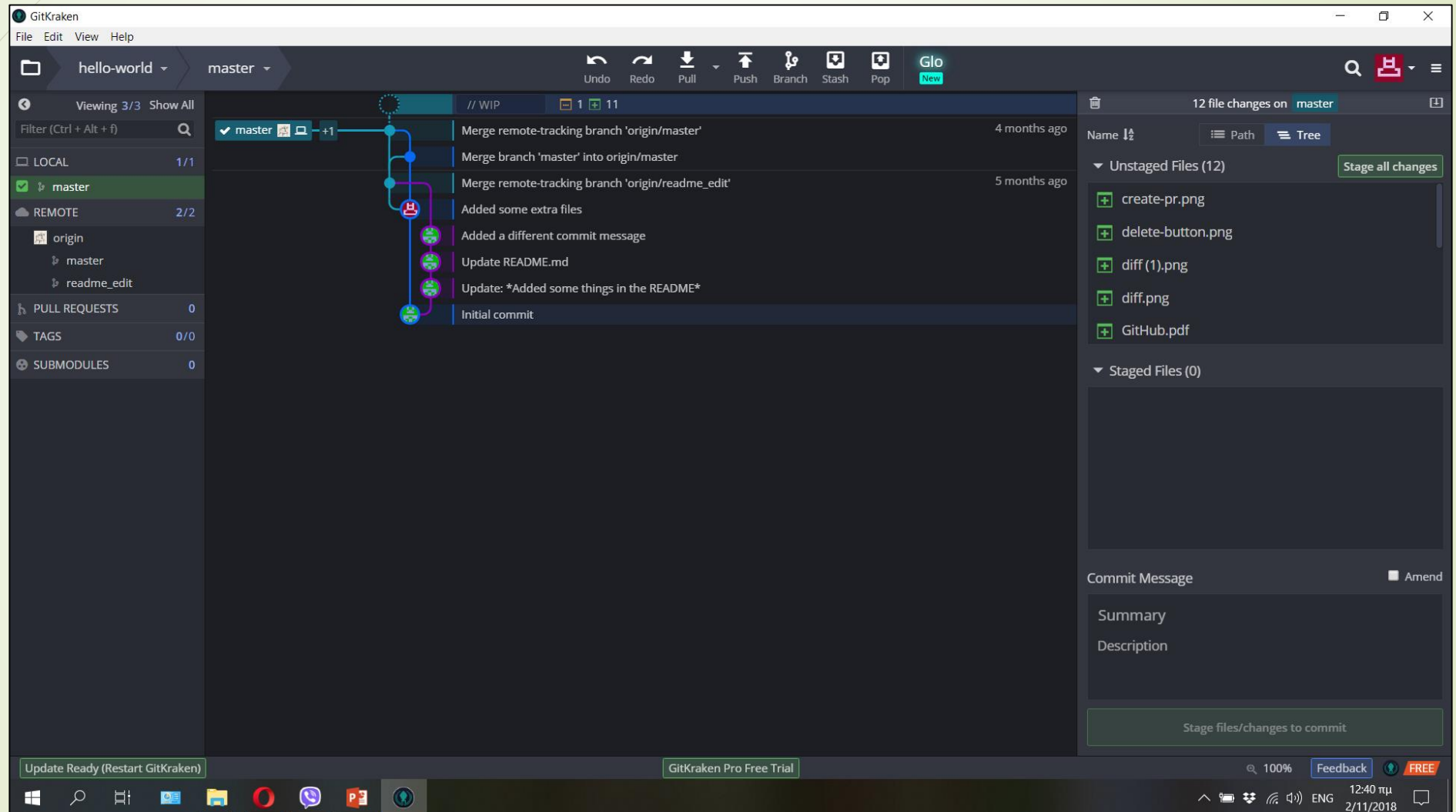
Χρήσεις Git clients και βασικές λειτουργίες.



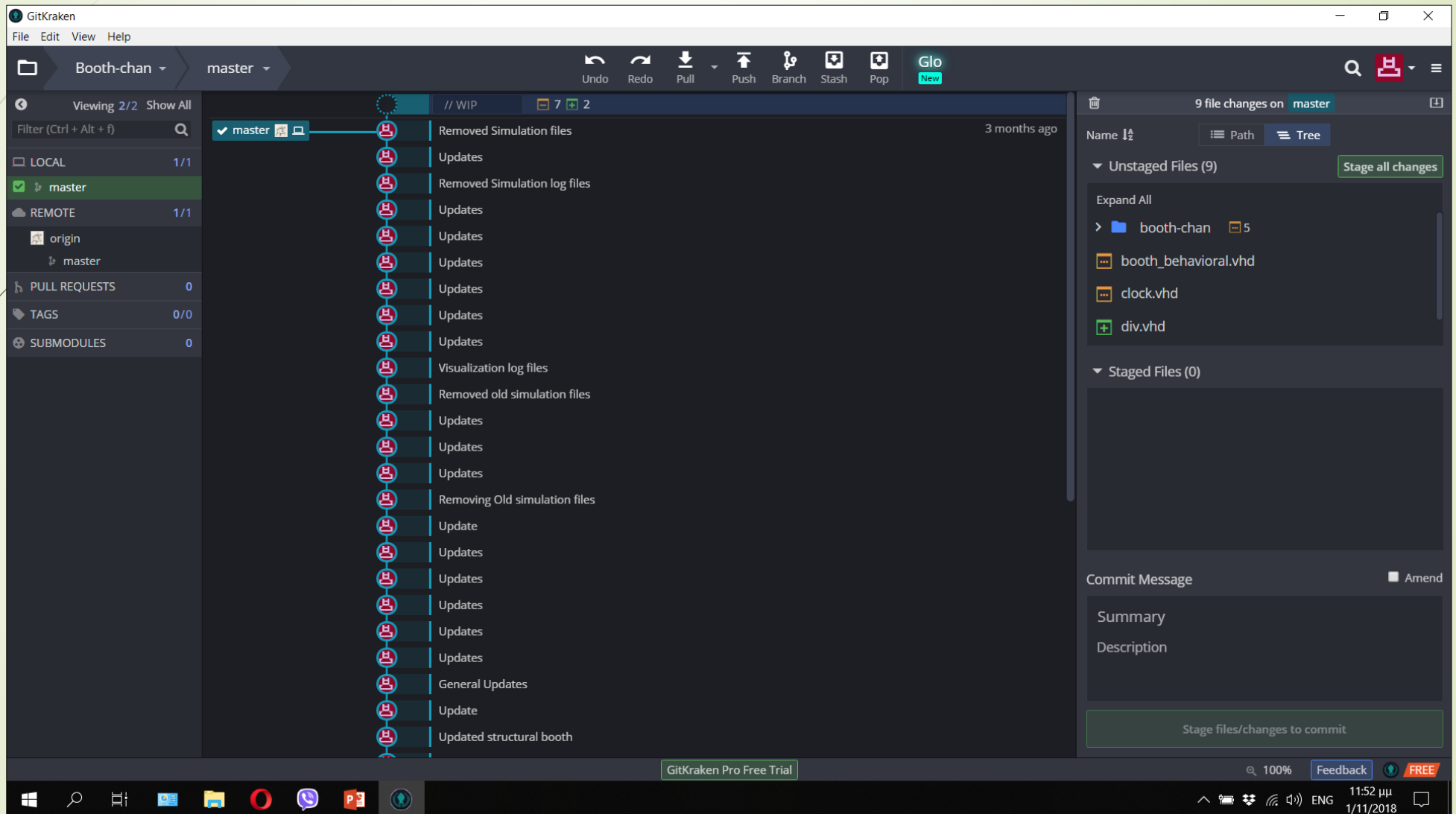
# GUI clients

- Το Git ως **εργαλείο** περιέχει προ εγκατεστημένα εργαλεία GUI για τις λειτουργίες του commit (git-gui) και του browsing (gitk).
- Υπάρχουν και διάφορα εργαλεία από τρίτους, για χρήστες που προτιμούν την χρήση του Git μέσω κάποιας πλατφόρμας.
- Αυτά ονομάζονται Git GUI-clients. Μερικά από αυτά είναι:
  - SourceTree,
  - GitHub Desktop,
  - **GitKraken**,
  - TortoiseGit,
  - SmartGit, etc.
- Τα παραπάνω προτείνονται για συστήματα **Windows**
- Περισσότερες πληροφορίες: <https://git-scm.com/download/gui/windows>

# Παράδειγμα Git Client – GitKraken – 1



# Παράδειγμα Git Client – GitKraken – 2



# Λειτουργίες/Εντολές

- Στο Git υπάρχουν αρκετές λειτουργίες/εντολές.
- Για μεγαλύτερη απλότητα θα αναλύσουμε μερικές από αυτές.
- Οι λειτουργίες που θα δούμε είναι:
  - **Stage**
  - **Push**
  - **Pull**
  - **Merge & Rebase**
  - **Fork**
- Για περισσότερες λεπτομέρειες πάνω σε όλες τις λειτουργίες/εντολές συμβουλευτείτε το διαδίκτυο.





# Staging

- Η διαδικασία του **staging** είναι απλώς η προετοιμασία του αρχείου για να γίνει ένα commit.
- Το Git επιτρέπει να κάνουμε commit μόνο συγκεκριμένα σημεία των αλλαγών που έγιναν από το τελευταίο commit.
- Παράδειγμα:
  - Δουλεύουμε σε ένα πρόγραμμα. Έχουμε δυο λειτουργίες, από τις οποίες η μία δουλεύει ολοκληρωμένα, αλλά η άλλη είναι ατελής με κάθε έννοια. Θα θέλαμε να κάνουμε ένα commit αλλά δεν θα θέλαμε να συμπεριλάβουμε τον κώδικα της δεύτερης λειτουργίας, ο οποίος είναι ακόμη ατελής. Οπότε κάνουμε stage τα σημεία που ξέρουμε ότι ανήκουν στην πρώτη λειτουργία, και τα κάνουμε commit.
- Πρακτικά το staging αποτελεί μια **ενδιάμεση κατάσταση**, ανάμεσα στο working local directory και το git repository.

# Παράδειγμα

The screenshot shows the GitKraken application interface. The main window displays a commit history on the right, a file explorer on the left, and a commit message editor at the bottom. A red error message is visible in the top right corner.

**GitKraken**  
File Edit View Help

hello-world master

Undo Redo Pull Push Branch Stash Pop Glo New

Viewing 3/3 Show All  
Filter (Ctrl + Alt + f)

LOCAL 1/1  
REMOTE 2/2

origin  
master  
readme\_edit

PULL REQUESTS 0  
TAGS 0/0  
SUBMODULES 0

Commit History:

- Merge remote-tracking branch 'origin/master' 4 months ago
- Merge branch 'master' into origin/master
- Merge remote-tracking branch 'origin/readme\_edit' 5 months ago
- Added some extra files
- Added a different commit message
- Update README.md
- Update: \*Added some things in the README\*
- Initial commit

**Staging paths failed**  
Could not stage paths for the following reason: Error: could not open 'D:/PyLam/hello-world/-\${GitHub.pptx}': Δεν ήταν δυνατή η προσπέλαση του αρχείου από τη διεργασία, επειδή χρησιμοποιείται ήδη από κάποια άλλη διεργασία.

Unstaged  
create-p  
delete-b  
diff (1).p  
diff.png  
GitHub.pdf

Staged Files (0)

Commit Message  
Amend

Summary  
Description

Stage files/changes to commit

Update Ready (Restart GitKraken) GitKraken Pro Free Trial 100% Feedback FREE

12:44 πμ 2/11/2018



# Push – 1

## Το δικαίωμα “εγγραφής” του git

- Η εντολή **push** χρησιμοποιείται για να ανεβάσουμε το περιεχόμενο ενός τοπικού repository (working local directory) σε ένα απομακρυσμένο repository (GitHub repository).
- Με την χρήση του push μεταφέρουμε τα commits από τον τοπικό χώρο στο απομακρυσμένο repository.
- Επίσης, μια άλλη λειτουργία είναι πως το push εξάγει τα commits σε κάποιο branch που έχουμε ορίσει. Προσοχή, το **default** δεν είναι πάντα το master, και ειδικά στους **Git Clients** καθώς αποθηκεύουν το τελευταίο push target.
- Το pushing πρέπει να γίνεται με ιδιαίτερη προσοχή καθώς έχει την δυνατότητα να επικαλύψει αλλαγές, οπότε δημιουργείται ο **κίνδυνος απώλειας κώδικα**.



# Push – 2

## Επιπλέον χαρακτηριστικά

- Μπορούμε να κάνουμε push ένα συγκεκριμένο branch στο remote repository, μαζί με όλα τα commits.
- Αν το branch δεν υπάρχει, δημιουργείται στον τοπικό φάκελο, και εμφανίζεται στον Git Client.
- Για την αποφυγή της επικάλυψης των commits, το Git δεν επιτρέπει να εκτελεστεί push όταν οδηγεί σε **non-fast-forward** merge στον φάκελο προορισμού (remote repository).
- Non-fast-forward: περίπτωση που προκύπτει αν το commit που γίνεται push δεν βασίζεται στην τρέχων ακμή του remote branch. Πιθανές αιτίες είναι να έχει αλλάξει το repository (branch) από τότε που ξεκινήσαμε να δουλεύουμε ή να εκτελούμε push σε λάθος project.



# Push – 3

## Επιπλέον χαρακτηριστικά

- Οι αλλαγές στο remote branch μπορεί να έχουν προέλθει από άλλους **contributors**. Εκτελώντας push παρακάμπτουμε το **code review**. Έτσι το push απορρίπτεται με το μήνυμα σφάλματος “non-fast forward merge”.
- Επίλυση του σφάλματος:
  - Επανατοποθέτηση του commit στο branch.
  - Merge το commit στο branch ή merge το branch στο local και μετά commit.
- Μετά από οποιαδήποτε από τις δύο επιλογές, το push θα γίνει με **επιτυχία**.
- Το push είναι ένα από τα συστατικά της **διαδικασίας συγχρονισμού του Git**.
- Το push θεωρείται ως το “upload command”, ενώ το **fetch** και το **pull** μπορούν να θεωρηθούν ως τα “download commands”.
- Μόλις τα σύνολα αλλαγών έχουν μεταφερθεί χρησιμοποιώντας είτε upload είτε download, το merge μπορεί να εκτελεστεί για να αφομοιωθούν οι αλλαγές στο προορισμό.

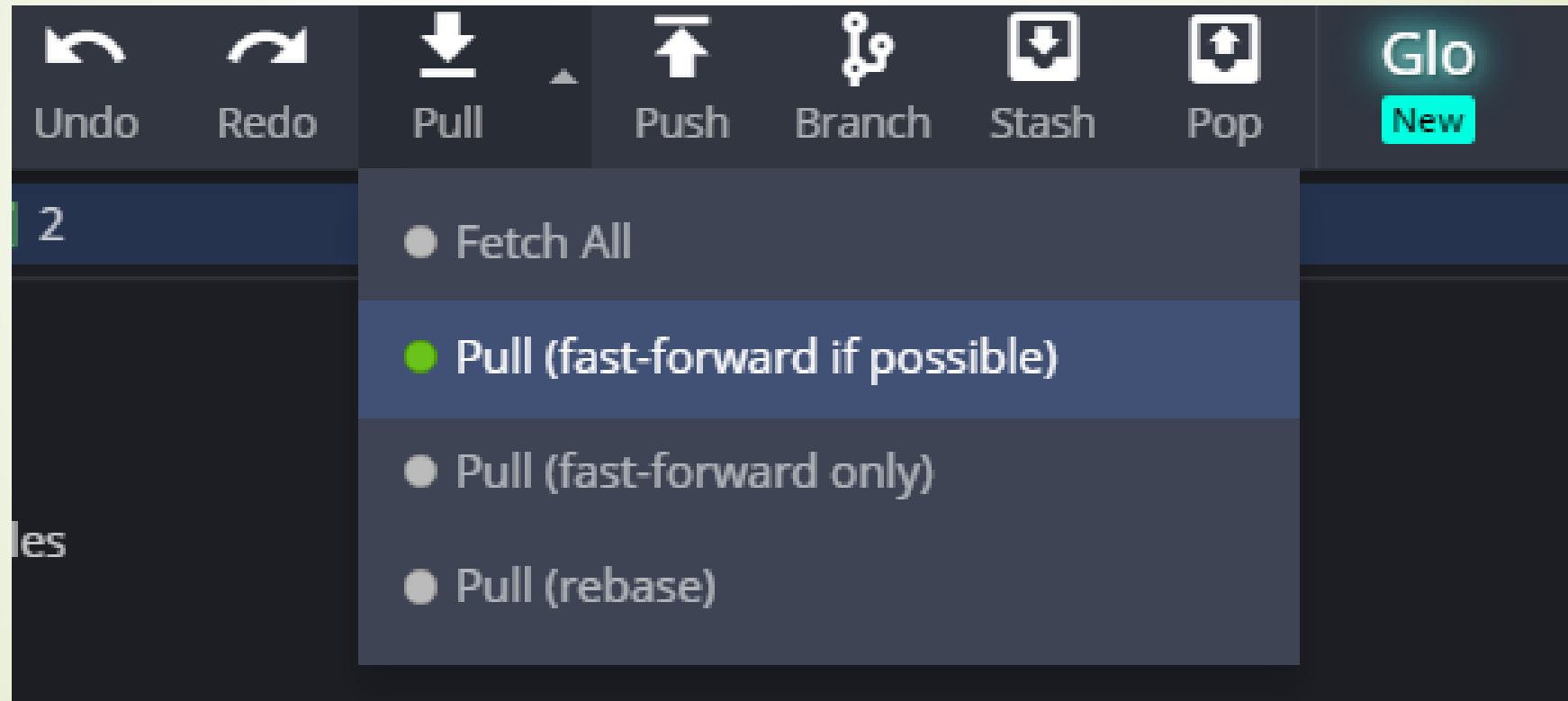
# Pull – 1

Το δικαίωμα “ανάγνωσης” του Git.

- Η εντολή/λειτουργία **pull** χρησιμοποιείται για να κάνουμε **fetch** και **download** από ένα απομακρυσμένο repository και να **ανανεώσουμε** το τοπικό repository άμεσα, ώστε να ακολουθεί το περιεχόμενο του remote.
- Το pull συντίθεται από δύο άλλες εντολές: το **fetch** και το **merge**.
- Σε πρώτη φάση το git pull θα εκτελέσει ένα fetch με στόχο το local branch όπου δείχνει το head. Στα Git Clients το Head αποτελεί το μέρος που αναγράφεται το όνομα του branch. Αφού το περιεχόμενο κατέβει, το git pull θα ενεργοποιήσει την ροή merge. Το merge θα ανανεώσει το branch έως το πιο πρόσφατο commit.
- Όπως και το push, έτσι και το pull έχει διάφορες παραλλαγές, οι οποίες αναλύονται στην επόμενη διαφάνεια.

# Pull – 2

Διαφορετικές μορφές pull



## Pull – 2

### Διαφορετικές μορφές pull

- **Απλό pull:** Fetch το branch που επιθυμούμε από το remote repository και άμεσο merge στο τοπικό αντίγραφο του branch.
- **Pull no-commit ή fetch all:** Παρόμοιο με το απλό pull, όμως σε αυτή την κλήση δεν δημιουργείται κανένα merge commit.
- **Pull (rebase):** Αντί για την χρήση του merge, καλείται η εντολή rebase. Το rebase επικεντρώνεται στην επανεγγραφή του ιστορικού του branch. Θα εκκινήσει μια εις βάθος ανάγνωση του ιστορικού του branch και θα επανατοποθετήσει commits με σωστή χρονική σειρά.



# Merge & Rebase – 1

Οι τεχνικές συγχώνευσης/ενσωμάτωσης

- Αρχικά, και οι δυο εντολές επιλύουν **ακριβώς το ίδιο** πρόβλημα. Και οι δυο εντολές είναι σχεδιασμένες να συγχωνεύουν αλλαγές από το ένα branch στο άλλο, αλλά **με διαφορετικό τρόπο**.
- **Merge**: Δημιουργία ενός merge commit από το source branch στο destination branch. Το commit επισυνάπτει τα ιστορικά των δύο branches δημιουργώντας ένα ενιαίο ιστορικό.
- **Rebase**: Επανατοποθετεί ολόκληρο το **source branch** στην κορυφή του **destination branch** (ίσως του master), ενσωματώνοντας όλα τα commits στο destination branch. Όμως αντί να χρησιμοποιήσει merge commits, το rebasing **επανεγγράφει το ιστορικό** του project, δημιουργώντας **καινούργια** commits για κάθε commit στο αρχικό branch.
- Θετικό του rebase είναι ότι έχει ως αποτέλεσμα ένα καθαρό project history.



# Merge & Rebase – 2

## Θετικά και αρνητικά του Rebase

- Εξαλείφει τα περιττά merge commits που απαιτούνται από την merge.
- Έχει ως αποτέλεσμα ένα τέλεια γραμμικό ιστορικό. Αυτό μας διευκολύνει στην παρακολούθηση και την μελέτη της ιστορίας του project από την πρώτη μέρα.
- Αρνητικά σημεία εμφανίζονται στους τομείς της ασφάλειας και της ιχνηλασιμότητας. Αν δεν ακολουθηθούν συγκεκριμένοι **κανόνες**, η επανεγγραφή του ιστορικού μπορεί να προβεί μοιραίο για την συνεργασία και την ροή της ομάδας.
- Επίσης, η απουσία των merge commits δεν παρέχει πληροφορίες για τον ακριβή χρόνο που ενσωματώθηκαν οι αλλαγές στο branch/repository.
- Χρυσός κανόνας: **Πότε μην χρησιμοποιείτε rebase σε public branches!!**



# Merge & Rebase – 2

## Κίνδυνοι του Rebase

- Το rebase μεταφέρει όλα τα commits στο master, στην κορυφή του destination branch. Το πρόβλημα είναι ότι αυτό συνέβη μόνο στο master repository!
- Οι υπόλοιποι developers εργάζονται ακόμη πάνω στο αρχικό master. Και εφόσον το rebasing ξαναδημιουργεί τα commits, και το ιστορικό ανασυντάσσεται, το Git θα νομίζει ότι το master branch αποκλίνει από τα versions των υπολοίπων.
- Ο μόνος τρόπος να επανασυγχρονιστούν τα branches είναι να εκτελεστεί ένα merge ώστε να συγχωνευθούν σε ένα. Αυτό οδηγεί σε ένα γιγαντιαίο merge commit, που περιέχει δύο σετ από commits με τις ίδιες αλλαγές (τις original και τις αλλαγές από το rebased branch).
- Οπότε πριν εκτελέσουμε το rebase πρέπει να αναρωτηθούμε ποιοι συνεργάτες μας παρακολουθούν το branch που πάμε να πειράξουμε.



# Fork

Το δικαίωμα “αντιγραφής” του Git.

- Η εντολή/λειτουργία fork δεν διαφέρει από την αντίστοιχη της στις γλώσσες προγραμματισμού.
- Στις γλώσσες προγραμματισμού η fork() δημιουργεί μια νέα διεργασία-παιδί από την αρχική (πατρική). Ο κώδικας του παιδιού είναι ο κώδικας από την κλήση της fork() μέχρι το return().
- Ομοίως λοιπόν η fork στο git αποτελεί εντολή δημιουργίας ενός κλώνου του repository.
- Κάνοντας fork το repository μπορούμε να πειραματιζόμαστε ελεύθερα πάνω σε αλλαγές χωρίς να επηρεάζουμε το αρχικό repository.
- Πιο συχνά τα fork χρησιμοποιούνται είτε για να προτείνουμε αλλαγές στο repository κάποιου άλλου, είτε να χρησιμοποιήσουμε ένα ξένο repository σαν γραμμή εκκίνησης για μια δική μας ιδέα (πάντα με την συγκατάθεση του ιδιοκτήτη).

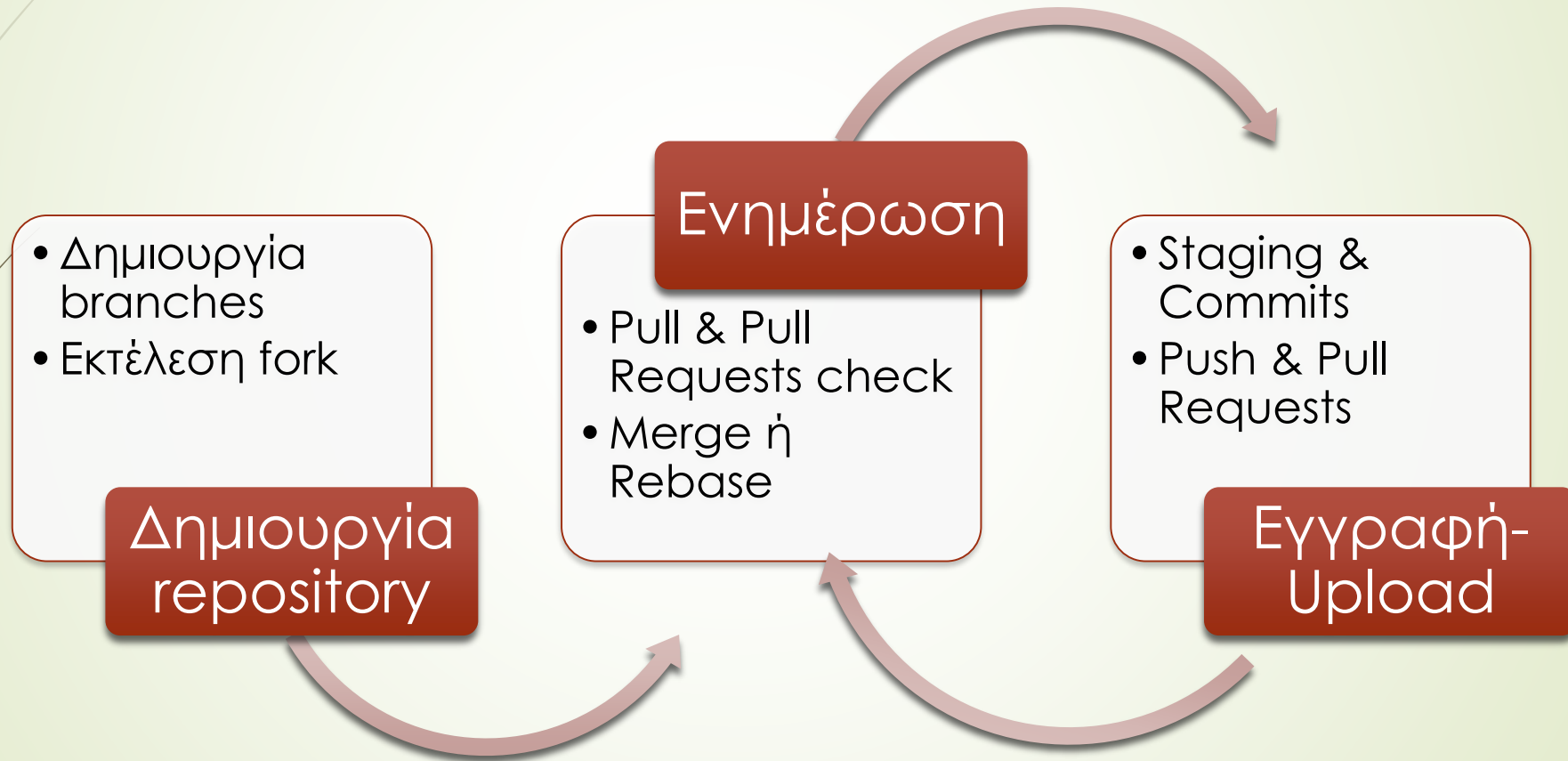
# Fork

## Η ροή χρήσης του fork

- Όπως είπαμε, τα fork μπορούν να χρησιμοποιηθούν στην πρόταση αλλαγών σε ξένα repository.
- Ως εκ τούτου, αποτελούν μέθοδο bug fixing:
  1. Εκτελούμε fork σε ένα repository.
  2. Προχωράμε στην διόρθωση.
  3. Καταθέτουμε ένα pull request στον ιδιοκτήτη του repository/project.
- Αν ο ιδιοκτήτης μείνει ευχαριστημένος ίσως και αποφασίσει να εκτελέσει pull στην διόρθωση μας και να την ενσωματώσει (merge) στο project του.
- Το fork πρέπει να είναι πάντα ενημερωμένο με βάση το αρχικό project στο GitHub. Πρέπει τακτικά να ελέγχονται τα pull requests και να γίνονται pulls πριν προβούμε σε οποιαδήποτε αλλαγή.
- **Κίνδυνος** να γίνει pull request από εμάς, ενώ δουλεύουμε σε παλιά έκδοση!

# Git Workflow!

Δουλεύοντας σε μια ομάδα







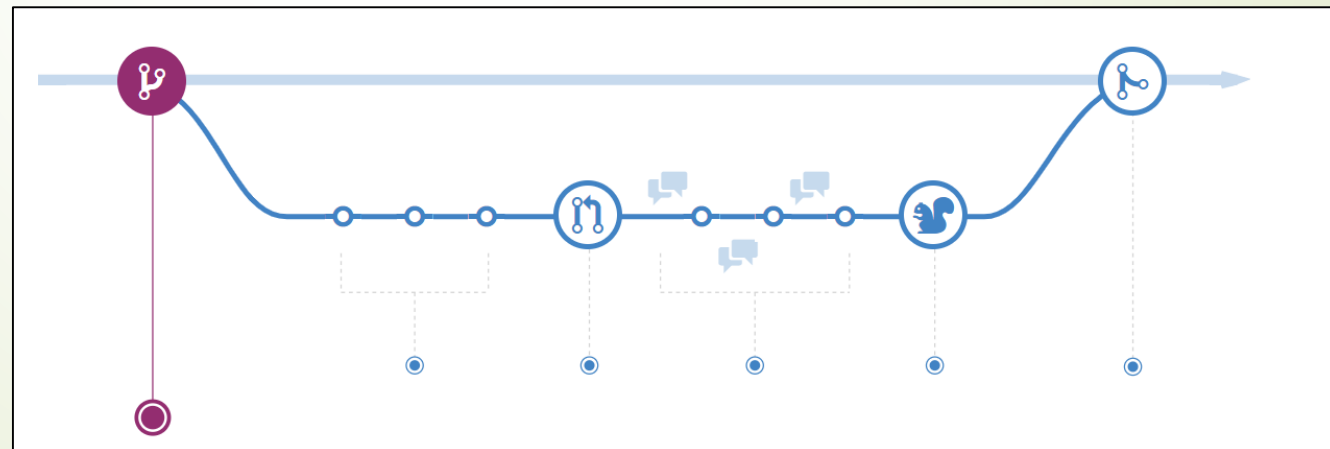
# GitHub flow

Παρουσίαση και κατανόηση σε βάθος.



# Δημιουργία ενός branch

- Όταν δουλεύουμε σε ένα project, θα δούμε πως η έμπνευση για νέες ιδέες και λειτουργίες είναι άπειρη. Πολλές από αυτές τις ιδέες είναι έτοιμες για ανάπτυξη, άλλες όμως χρειάζονται λίγο παραπάνω σκέψη. Το Branching υπάρχει έτσι ώστε να διαχειριζόμαστε αυτή την ροή εργασίας.
- Όταν δημιουργούμε ένα branch στο project μας, δημιουργούμε ένα περιβάλλον όπου μπορούμε να δοκιμάσουμε νέες ιδέες. Οι αλλαγές που κάνουμε σε ένα branch δεν επηρεάζουν το αρχικό project. Είμαστε εξασφαλισμένοι πως οι αλλαγές δεν θα ενσωματωθούν στο project μέχρι κάποιος συνεργάτης μας, να τις εγκρίνει.





# Συμβουλές branching

- Το branching είναι ένα από τα βασικά στοιχεία του Git, και ολόκληρο το GitHub flow είναι βασισμένο σε αυτό.
- Μοναδικός κανόνας είναι ότι, οποιοδήποτε είδος κώδικα υπάρχει στο master branch είναι πάντα αναπτυσσόμενο.
- Ως εκ τούτου, είναι σημαντικό, κάθε νέο branch να δημιουργείται από το master, όταν εργαζόμαστε πάνω σε μια νέα λειτουργία ή διόρθωση.
- Το όνομα του branch πρέπει να αντιστοιχεί στο θέμα της εργασίας που μελετάται σε αυτό, ώστε να διευκολύνονται οι συνεργάτες μας.
- Παραδείγματα ονομάτων: refactor-authentication, renaming-basic-functions, reducing-memory-usage, user-content-cache-key, etc.

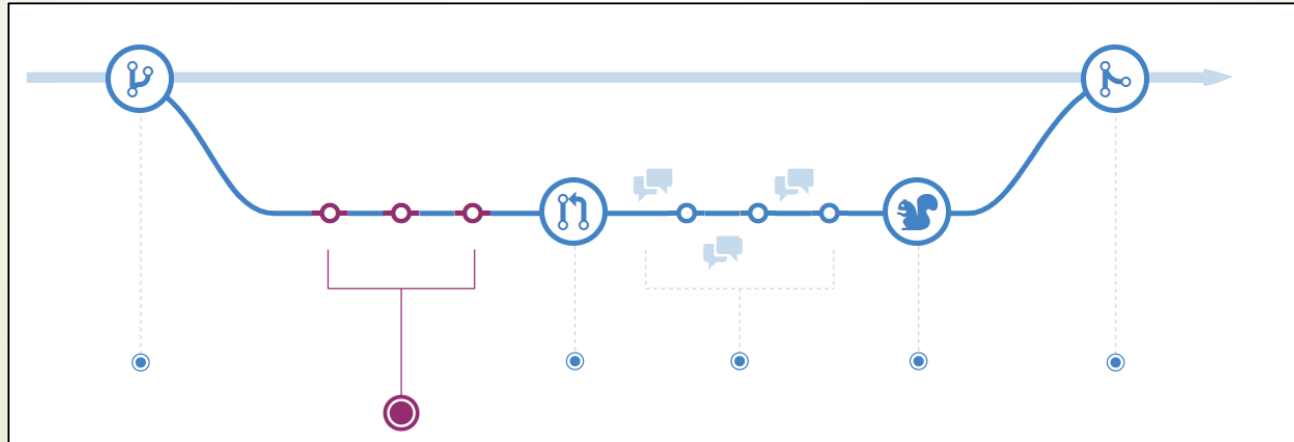


# Commits

- Όταν το branch μας έχει δημιουργηθεί είναι η στιγμή να αρχίσουμε τις αλλαγές.
- Commit θεωρείται οποιαδήποτε ενέργεια πάνω στο αρχείο, είτε είναι add, edit ή delete.
- Τα commits προστίθενται στο branch, και μας επιτρέπουν να παρακολουθούμε την πορεία ανάπτυξης.
- Δημιουργείτε ένα ιστορικό προσθηκών, επιτρέποντας σε τρίτους να καταλάβουν τον τρόπο με τον οποίο αναπτύξαμε τον κώδικα μας.
- Κάθε commit ακολουθείται από ένα σχετικό commit message, το οποίο περιγράφει τους λόγους που έγινε η συγκεκριμένη αλλαγή.
- Κάθε commit θεωρείται μια διαφορετική μονάδα, που έφερε αλλαγές στον κώδικα.
- Αυτό μας βοηθά στην περίπτωση που βρεθεί bug, να ανατρέξουμε πίσω και να το διορθώσουμε.

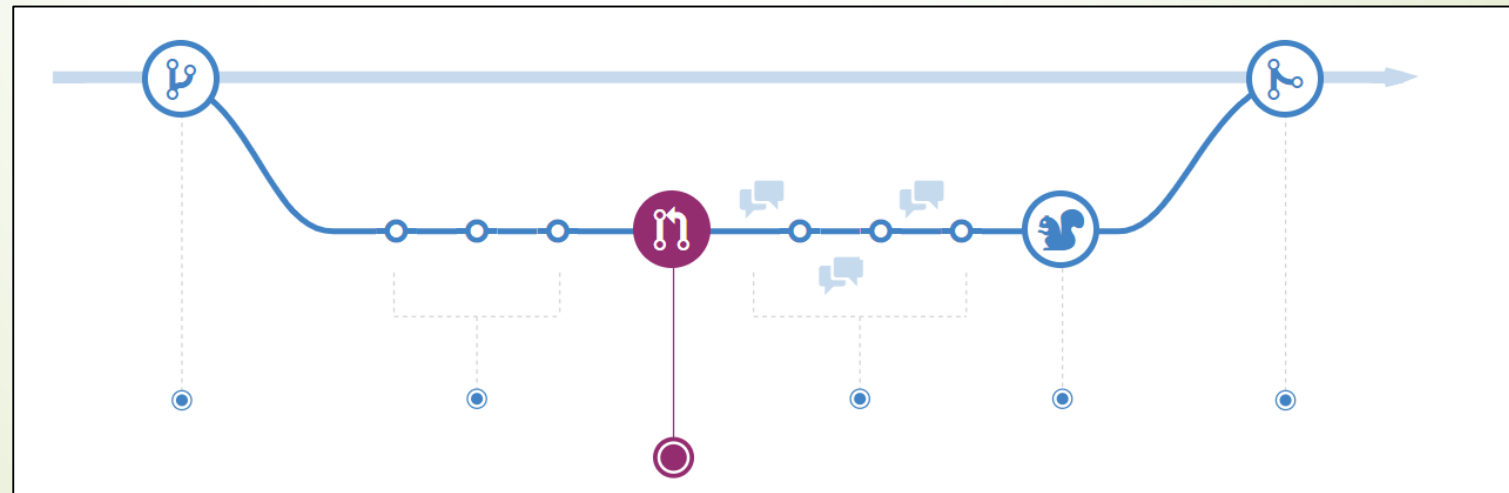
# Συμβουλές για τα commits

- Τα commit messages είναι σημαντικά, εφόσον το Git παρακολουθεί τις αλλαγές και τις εμφανίζει ως commits όταν τις κάνουμε push στο server.
- Γράφοντας καθαρά και κατανοητά commit messages, διευκολύνουμε τους υπολοίπους χρήστες στην καλύτερη παρακολούθηση του κώδικα.
- Έτσι, θα είναι ευκολότερο να λάβουμε feedback καθ' όλη την πορεία ανάπτυξης κώδικα.



# Pull Requests

- Τα pull requests εκκινούν συζητήσεις για τα commits που γίνονται.
- Μέσω των pull requests ο καθένας μπορεί να δει τι αλλαγές θα ενσωματωθούν στον κώδικα του εάν αποδεχθεί το request.
- Pull request μπορεί να γίνει οποιαδήποτε στιγμή κατά την διάρκεια της ανάπτυξης του project.



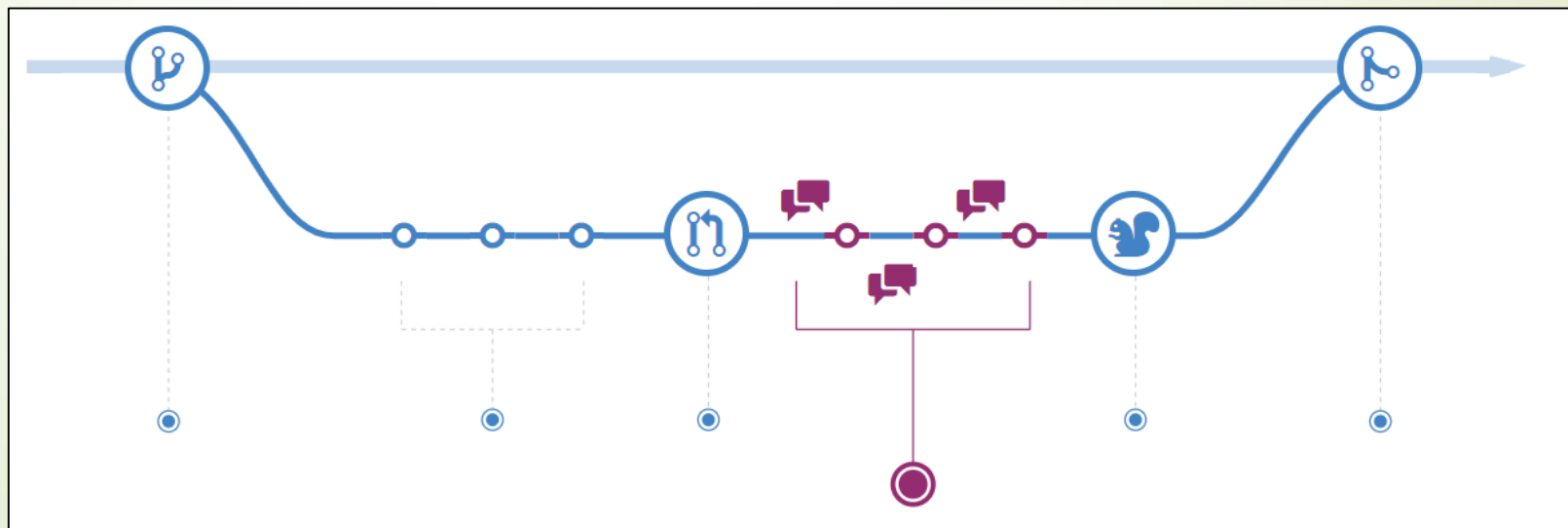
# Pull Requests

## Κυριότερες χρήσεις

- Τα pull requests είναι χρήσιμα για την συνεισφορά σε project ανοικτού κώδικα και για την διαχείριση των αλλαγών σε κοινόχρηστα repositories.
- Αν χρησιμοποιείται το μοντέλο Fork & Pull, τα pull requests αποτελούν τρόπο για να ενημερώσουμε τους συντηρητές για τις αλλαγές που θέλουμε να προτείνουμε.
- Αν χρησιμοποιείται το μοντέλο Shared Repository, τα pull requests βοηθούν στην εκκίνηση κριτικών για τους κώδικες και συζητήσεις πάνω σε προτεινόμενες αλλαγές, πριν ενσωματωθούν στο master branch.

# Συζητήσεις και κριτικές

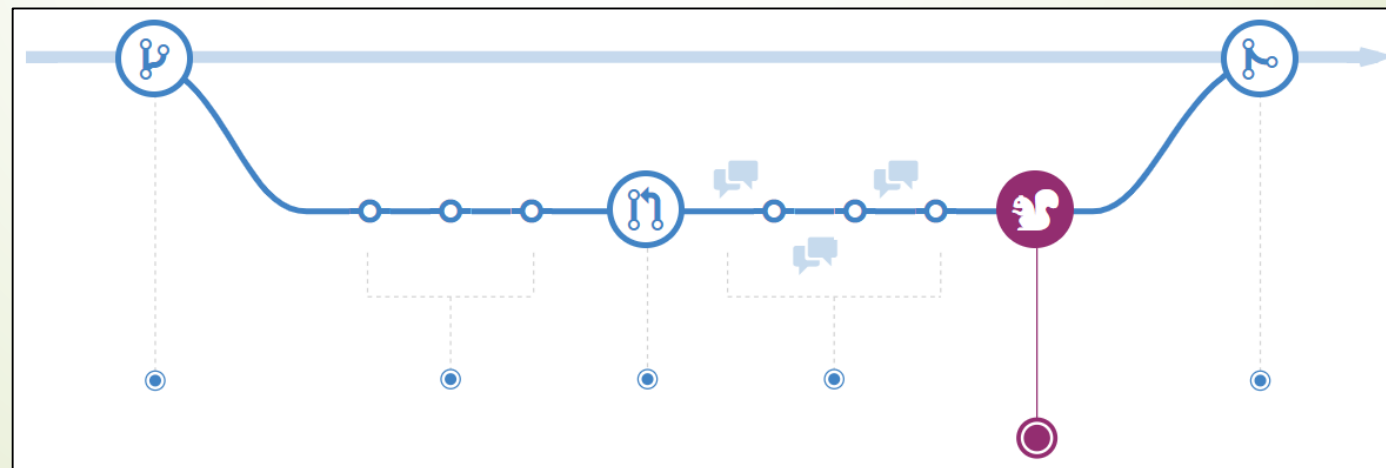
- Έχοντας γίνει ένα Pull request, ο χρήστης αναμένει τους υπόλοιπους υπεύθυνους του repository για κάποιου είδους απάντηση.
- Οι συνεργάτες ίσως εκφράσουν κάποιες ερωτήσεις ή σχόλια.
- Τα pull request είναι σχεδιασμένα ώστε να πυροδοτούν συζητήσεις πάνω σε ερωτήσεις ή σχόλια.





# Deploy

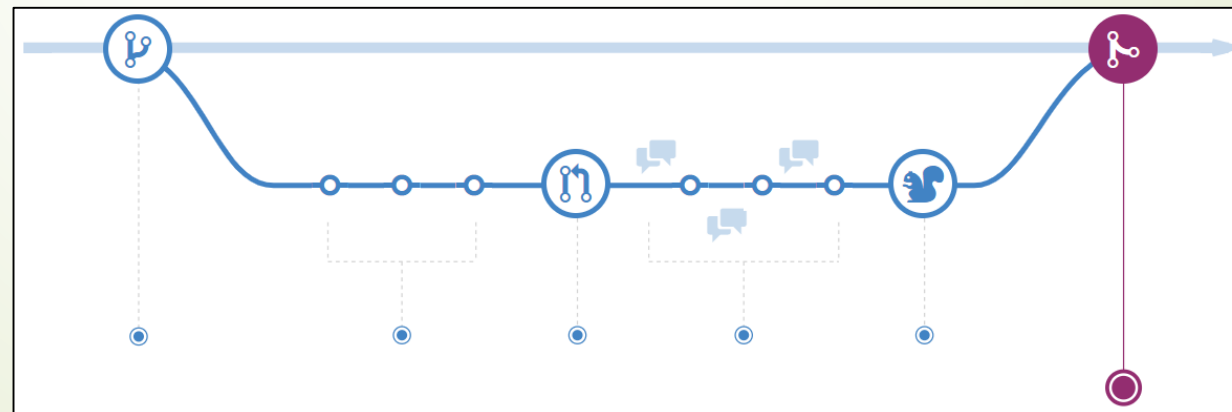
- Με το GitHub, μπορούμε να παρατάξουμε ένα branch για τελικό έλεγχο παραγωγής πριν το ενσωματώσουμε στο master branch.
- Μόλις το pull request έχει κριθεί και το branch περάσει τα απαραίτητα tests, μπορούμε να προωθήσουμε τις αλλαγές έτσι ώστε να τις επαληθεύσουμε στην παραγωγή.
- Αν το branch δημιουργεί προβλήματα, μπορούμε να ανατρέξουμε και να προωθήσουμε το ήδη υπάρχον master branch.





# Merge

- Εφόσον οι αλλαγές μας έχουν επικυρωθεί στην παραγωγή, είναι πλέον η ώρα να τις ενσωματώσουμε στον κώδικα του master branch.
- Όταν έχουμε ενσωματώσει τις αλλαγές, τα pull requests σχηματίζουν ένα ιστορικό αλλαγών του κώδικα μας.
- Το ιστορικό υποστηρίζει αναζήτηση. Οπότε επιτρέπει σε κάποιον να μελετήσει την πορεία ανάπτυξης του κώδικα στο repository, και την λογική πίσω από τις ενημερώσεις.





# Permissions

Δικαιώματα χρηστών σε repositories.



# Ομάδες

- Όπως είναι γνωστό, η χρήση του GitHub συνίσταται κατά την ανάπτυξη ομαδικών project.
- Ως εκ τούτου, είναι φυσικό σε μια ομάδα να υπάρχουν διαφορετικά αξιώματα. Κάποια από αυτά θα μπορούσαμε να τα ονομάσουμε **team leader, member, tech-support** κλπ..
- Ως γενίκευση, το GitHub προσφέρει τα παρακάτω αξιώματα:
  - **Owner** (Ιδιοκτήτης)
  - **Collaborator** (Συνεργάτες)
  - **Organization Members** (Μέλη)
- Γενικά ένα repository που ανήκει σε ένα λογαριασμό έχει δυο επίπεδα δικαιωμάτων
  - **Owner level** (Επίπεδο ιδιοκτήτη)
  - **Collaborator level** (Επίπεδο συνεργάτη)

# Owners (Ιδιοκτήτες) - 1

## Owner level permissions

- Ο ιδιοκτήτης ενός repository έχει πλήρη έλεγχο πάνω σε αυτό.
- Ο ιδιοκτήτης είναι μοναδικός.
- Μερικά από τα δικαιώματα είναι τα παρακάτω:
  - Πρόσκληση συνεργατών (collaborators).
  - Αλλαγή της διαφάνειας του repository. Από public σε private και το αντίθετο.
  - Περιορισμός αλληλεπιδράσεων με το repository.
  - Συγχώνευση ενός pull request σε ένα protected branch, ακόμη και αν δεν υπάρχουν κριτικές αποδοχής.
  - Διαγραφή του repository.
  - Διαχείριση της θεματολογίας ενός repository.
  - Επιλογή **CODEOWNERS** για ένα repository.
  - Αρχειοθέτηση ενός repository.

## Owners (Ιδιοκτήτες) – 1.2

### Περιορισμός αλληλεπιδράσεων με το repository

- Χρήστες με δικαιώματα ιδιοκτήτη σε κάποιο public repository μπορούν, προσωρινά, να περιορίσουν συγκεκριμένους χρήστες από το να σχολιάσουν, να ξεκινήσουν ζητήματα ή να κάνουν κάποιο pull request.
- Ουσιαστικά επιβάλλουμε μια περίοδο περιορισμού για κάποιους χρήστες.
- Η διάρκεια του περιορισμού διαρκεί 24 ώρες. Αφού παρέλθει το χρονικό όριο, οι χρήστες αποκτούν ξανά την ελευθερία τους.
- Περισσότερες λεπτομέρειες στο: <https://help.github.com/articles/limiting-interactions-in-your-repository/>

# Owners (Ιδιοκτήτες) – 1.3

## Αρχειοθέτηση repository

- Με την αρχειοθέτηση ενός repository επισημαίνουμε πως το project στο οποίο αναφερόταν το συγκεκριμένο repository δεν συντηρείται πλέον ενεργά.
- Προτείνεται πριν την αρχειοθέτηση να κλείνονται όλα τα ζητήματα και τα pull requests, καθώς και να ενημερώνεται αντίστοιχα το README και η περιγραφή του repository.
- Όταν ολοκληρωθεί η αρχειοθέτηση, δεν είναι πλέον δυνατή η προσθήκη ή αφαίρεση συνεργατών ή ομάδων. Συνεργάτες με πρόσβαση στο repository μπορούν μόνο να εκτελέσουν fork ή star.
- Κατά την αρχειοθέτηση, όλα τα στοιχεία του repository (issues, pull requests, code, labels, milestones, commits, tags, branches, comments κτλ.) γίνονται read-only. Για να γίνουν αλλαγές πρέπει να επαναφερθεί το repository.
- Περισσότερες λεπτομέρειες: <https://help.github.com/articles/about-archiving-repositories/>



# Collaborators (Συνεργάτες) - 1

## Collaborator level permissions

- Οι συνεργάτες πρέπει να έχουν προσκληθεί από τον ιδιοκτήτη του repository.
- Στα πλαίσια της συνεργασίας, τους παραχωρούνται τα εξής δικαιώματα:
  1. Push to (write), pull from (read) και fork (copy) ενός repository.
  2. Δημιουργία, εφαρμογή και διαγραφή labels και milestones.
  3. Άνοιγμα, κλείσιμο, άνοιγμα εκ νέου και ανάθεση ζητημάτων.
  4. Παρέμβαση και διαγραφή σε σχόλια πάνω σε commits, pull requests και ζητήματα.
  5. Επισήμανση ενός ζητήματος ή pull request ως διπλότυπο.
  6. Άνοιγμα, συγχώνευση και κλείσιμο pull requests.
  7. Εφαρμογή προτεινόμενων αλλαγών σε pull requests που έχουν γράψει ή τους έχουν ανατεθεί.
  8. Αποστολή pull request από forks του repository.



# Collaborators (Συνεργάτες) - 2

## Collaborator level permissions

- Συνέχεια δικαιωμάτων συνεργατών:
  - 9. Αφαίρεση των δικαιωμάτων τους ως συνεργάτες.
  - 10. Κατάθεση κριτικής πάνω σε ένα pull request, η οποία μπορεί να επηρεάσει την συγχωνευσιμότητα του.
  - 11. Συμπεριφορά ως διορισμένοι **CODEOWNERS**.
  - 12. Κλείδωμα μιας συνομιλίας.
  - 13. Αναφορά υβριστικού περιεχομένου στο GitHub Support ή στο GitHub Premium Support.
- Φυσικά, πρέπει να μας είναι αντιληπτό ότι όλα τα δικαιώματα των συνεργατών αποτελούν και δικαιώματα του ιδιοκτήτη (ακόμη και αν δεν αναφέρθηκαν πρωτύτερα).

# Collaborators (Συνεργάτες) - 3

## CODEOWNERS

- Ο όρος CODEOWNERS αναφέρεται στους χρήστες, οι οποίοι λαμβάνουν δικαιώματα εγγραφής πάνω σε ένα repository από τον ιδιοκτήτη.
- Ως δικαιώματα εγγραφής αναφερόμαστε στην λειτουργία **Push to repo**.
- Οι CODEOWNERS ζητούνται αυτόματα όταν κάποιος εκτελεί ένα pull request, το οποίο αφορά κώδικα όπου οι ίδιοι έχουν οριστεί ως «ιδιοκτήτες».
- Επίσης όταν κάποιος με δικαιώματα admin ή owner έχει ενεργοποιήσει την απαίτηση για κριτικές κατά τα pull request, οι CODEOWNERS πρέπει να δώσουν την συγκατάθεση τους πριν γίνει δεκτό ένα Pull request.
- Πληροφορίες για τον ορισμό των CODEOWNERS δίνονται στο link:  
<https://help.github.com/articles/about-codeowners/>

# Collaborators (Συνεργάτες) - 4

## Πρόσκληση συνεργατών

- Η πρόσκληση συνεργατών γίνεται μέσω μερικών απλών βημάτων:
  1. Μέσω GitHub, πηγαίνουμε στην κεντρική σελίδα του repository.
  2. Κάτω από τον τίτλο του repository πατάμε το κουμπί των **settings**.
  3. Στην αριστερή μπάρα πατάμε την επιλογή **collaborators**.
  4. Κάτω από το collaborators μπορούμε να πληκτρολογήσουμε το username.
  5. Επιλέγουμε το username από το μενού που θα εμφανιστεί.
  6. Τέλος, πατάμε το **add collaborator**.
  7. Ο χρήστης που επιλέξαμε θα λάβει ένα e-mail που τους προσκαλεί στο repository. Με την αποδοχή της πρόσκλησης λαμβάνουν και τα δικαιώματα συνεργάτη πάνω στο repository.

# Collaborators (Συνεργάτες) - 5

## Επιπλέον πληροφορίες

- Για περισσότερες οδηγίες σχετικά με:
  - Τα γενικά περί συνεργατών: <https://help.github.com/articles/permission-levels-for-a-user-account-repository/>
  - την αφαίρεση συνεργατών: <https://help.github.com/articles/removing-a-collaborator-from-a-personal-repository>
  - την παραίτηση συνεργατών: <https://help.github.com/articles/removing-yourself-from-a-collaborator-s-repository>
  - την οργάνωση ομάδας: <https://help.github.com/articles/organizing-members-into-teams>

# Πηγές

- Θεωρία: <https://www.atlassian.com/git/tutorials>
- Θεωρία: <https://help.github.com/articles/github-flow/>
- Θεωρία: <https://help.github.com/articles/fork-a-repo/>
- Θεωρία: <https://git.eclipse.org/r/Documentation/error-non-fast-forward.html>
- Θεωρία & Εικόνες: <https://guides.github.com/activities/hello-world/>
- Θεωρία & Εικόνες: <https://guides.github.com/introduction/flow/>

