



# GitHub

Εγχειρίδιο χρήσης GitHub  
για Windows (7 και πάνω)

## Git και GitHub

Το **Git** είναι ένα ελεύθερο και ανοικτού κώδικα σύστημα ελέγχου έκδοσης που έχει σχεδιαστεί για να διαχειρίζεται τα πάντα από μικρά σε πολύ μεγάλα έργα προγραμματισμού και αρχείων με ταχύτητα και αποτελεσματικότητα.

Όταν είχε δημιουργηθεί το git , έτρεχε μόνο τοπικά, στον προσωπικό υπολογιστή του καθενός. Στη συνέχεια δημιουργήθηκαν προσωπικοί git servers ώστε ο προγραμματιστής/χρήστης να μπορεί να αποθηκεύει τις αλλαγές του εκεί. Όσπου δημιουργήθηκε το **GitHub**. Ένας ελεύθερος και δημόσιος χώρος διανομής repositories που δίνει τη δυνατότητα δημιουργίας κρυφών repositories χωρίς τη χρήση ιδιωτικού server.

## Γιατί Git ;

Ο έλεγχος έκδοσης είναι ο μόνος λογικός τρόπος για να παρακολουθείτε τις αλλαγές σε κώδικα, χειρόγραφα, παρουσιάσεις και έργα ανάλυσης δεδομένων. Οι περισσότεροι χρησιμοποιούν zip αρχεία με διαφορετικές ημερομηνίες και εκδόσεις. Αλλά η διερεύνηση των διαφορών είναι δύσκολη, τουλάχιστον. Αν χρησιμοποιείται το git σωστά, η παραμικρή αλλαγή θα υπάρχει αποθηκευμένη και αρχειοθετημένη.

Η συγχώνευση των αλλαγών των συνεργατών γίνεται εύκολα. Έχετε ποτέ αντιμετωπίσει έναν συνεργάτη που σας έστειλε τροποποιήσεις που διανεμήθηκαν σε πολλά αρχεία ή έπρεπε να ασχοληθεί με δύο άτομα που έχουν κάνει αλλαγές στο ίδιο αρχείο ταυτόχρονα; Επώδυνος. *git merge* είναι η απάντηση.

## Γιατί GitHub ;

Όλοι είναι εκεί. Από Samsung και Apple, μέχρι Saga Robotics και TensorFlow project. Μπορείτε να δείτε τι δουλεύουν και να διαβάσετε εύκολα τον κώδικα τους, να κάνετε προτάσεις ή αλλαγές, να δείτε τον τρόπο που δουλεύουν και να παρατηρήσετε τις μικρές αλλαγές που κάνουν (commits)

Είναι Open source. Με το github, όλος ο κώδικας ελέγχεται εύκολα, όπως και ολόκληρο το ιστορικό του. Το Github μειώνει τα εμπόδια στη συνεργασία. Είναι εύκολο να προσφέρετε προτεινόμενες αλλαγές στον κώδικα των άλλων μέσω του github.

Δεν χρειάζεται να δημιουργηθεί ένα server. Είναι εκπληκτικά εύκολο να μπει κάποιος σε αυτό και να επεξεργάζεται κώδικα.

Τέλος, το github βοηθά στην ανάπτυξη των project ακόμη και απομακρυσμένα χωρίς να χρειάζεται να χρησιμοποιούμε τον προσωπικό μας υπολογιστή.

## Δεν είμαι προγραμματιστής. Μπορώ να το χρησιμοποιήσω;

Το GitHub μπορεί να χρησιμοποιηθεί και για αποθήκευση αρχείων με ανάλογη χρήση όπως αυτή του dropbox, με τη διαφορά ότι στο github μπορείς να βλέπεις το ιστορικό των αλλαγών.

Προσοχή! Το GitHub ΔΕΝ εξυπηρετεί ως file hosting server. Πέραν του μικρού χώρου που δίνει για αποθήκευση αρχείων, η φιλοσοφία του είναι πολύ διαφορετική. Φυσικά μπορεί να γίνει αποθήκευση αρχείων εκεί.

### Τι αρχεία υποστηρίζονται στο GitHub;

Όλα, με μέγιστο μέγεθος 100mb ανά αρχείο και 5GB ανά repository. Κάποια αρχεία παρόλα αυτά αναγνωρίζονται ως αρχεία που δεν μπορείς να δεις τις αλλαγές του μέσα σε αυτά. Παρακάτω ακολουθεί ένα σενάριο χρήσης GitHub με στόχο την βελτιστοποίηση ενός project προγραμματισμού. Πριν από αυτό όμως θα παρουσιαστούν η εγκατάσταση του GitHub για Windows, και μερικές ορολογίες χρήσιμες για την κατανόηση του τρόπου λειτουργίας του.

## Εγκατάσταση

1. <https://desktop.github.com/>
2. Κατεβάζουμε το εκτελέσιμο.
3. Προχωράμε το setup το οποίο έχει αρκετά εύκολη πλοήγηση.
4. Όταν τελειώσει η εγκατάσταση προχωράμε στη δημιουργία λογαριασμού μέσω της σελίδας <https://github.com/join?source=experiment-header-dropdowns-home>
5. Ακολουθούμε τις οδηγίες όπως φαίνεται παρακάτω

# Join GitHub

The best way to design, build, and ship software.



Step 1:  
Create personal account



Step 2:  
Choose your plan



Step 3:  
Tailor your experience

## Create your personal account

Username

MyGithubNamelsHere

This will be your username. You can add the name of your organization later.

Email address

myname@certh.gr

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

Password

.....

Use at least one lowercase letter, one numeral, and seven characters.

Το όνομα θα φαίνεται δημοσίως και θα υπάρχει σε κάθε σας commit/αλλαγή που κάνετε στο project σας

Ένα οποιοδήποτε email για να την επιβεβαίωση των στοιχείων. Μπορούν να προστεθούν πολλά emails σε έναν λογαριασμό στη συνέχεια

### You'll love GitHub

Unlimited collaborators

Unlimited public repositories

✓ Great communication

✓ Frictionless development

✓ Open source community

Completed Set up a personal account	Step 2: Choose your plan	Step 3: Tailor your experience
--	-----------------------------	-----------------------------------

## Choose your personal plan

☒ Unlimited public repositories for free.

☐ Unlimited private repositories for \$7/month. [\(view in EUR\)](#)

Το iBO διαθέτει ήδη  
δωρεάν ιδιωτικά  
repositories

### Both plans include:

- ✓ Collaborative code review
- ✓ Issue tracking
- ✓ Open source community
- ✓ Unlimited public repositories
- ✓ Join any organization

Don't worry, you can cancel or upgrade at any time.

- ☐ **Help me set up an organization next**  
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.  
[Learn more about organizations](#)

- ☐ **Send me updates on GitHub news, offers, and events**  
Unsubscribe anytime in your email preferences. [Learn more](#)

Continue

Completed Set up a personal account	Step 2: Choose your plan	Step 3: Tailor your experience
--	-----------------------------	-----------------------------------

How would you describe your level of programming experience?

- ☐ Very experienced      ☐ Somewhat experienced      ☒ Totally new to programming

What do you plan to use GitHub for? (check all that apply)

- ☒ Development      ☐ Design      ☐ School projects  
☒ Research      ☒ Project Management      ☐ Other (please specify)

Επιλέγουμε ότι επιθυμούμε. Δεν είναι  
δημοσίως προσβάσιμα στοιχεία.

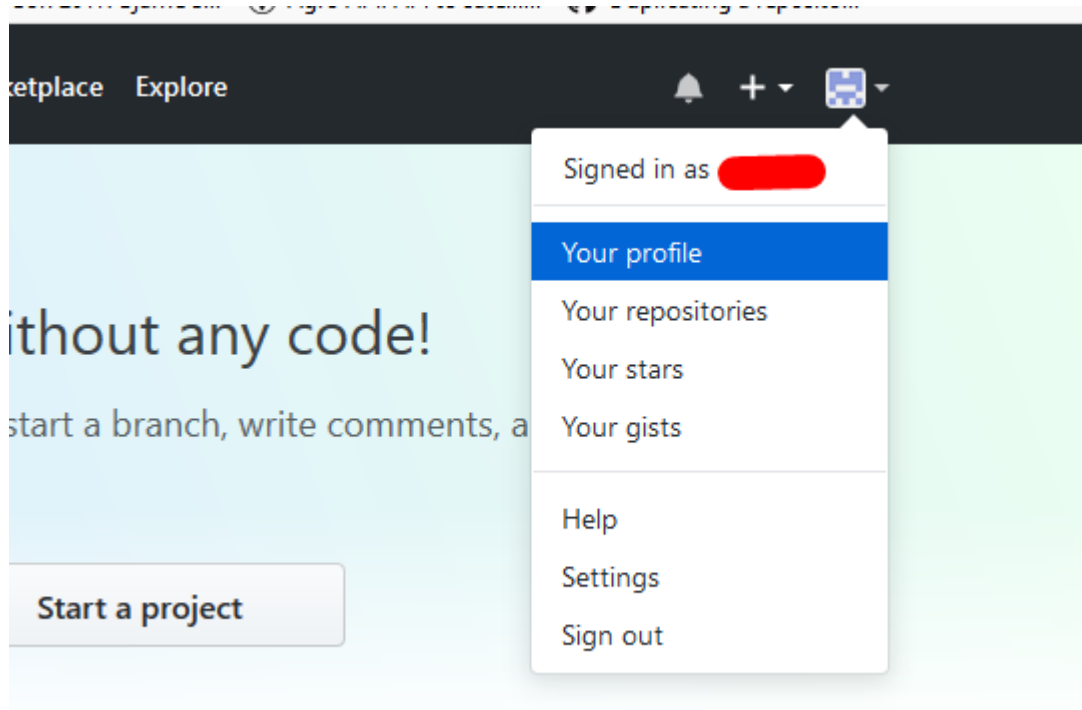
Which is closest to how you would describe yourself?

- ☐ I'm a professional      ☐ I'm a hobbyist      ☒ I'm a student  
☐ Other (please specify)

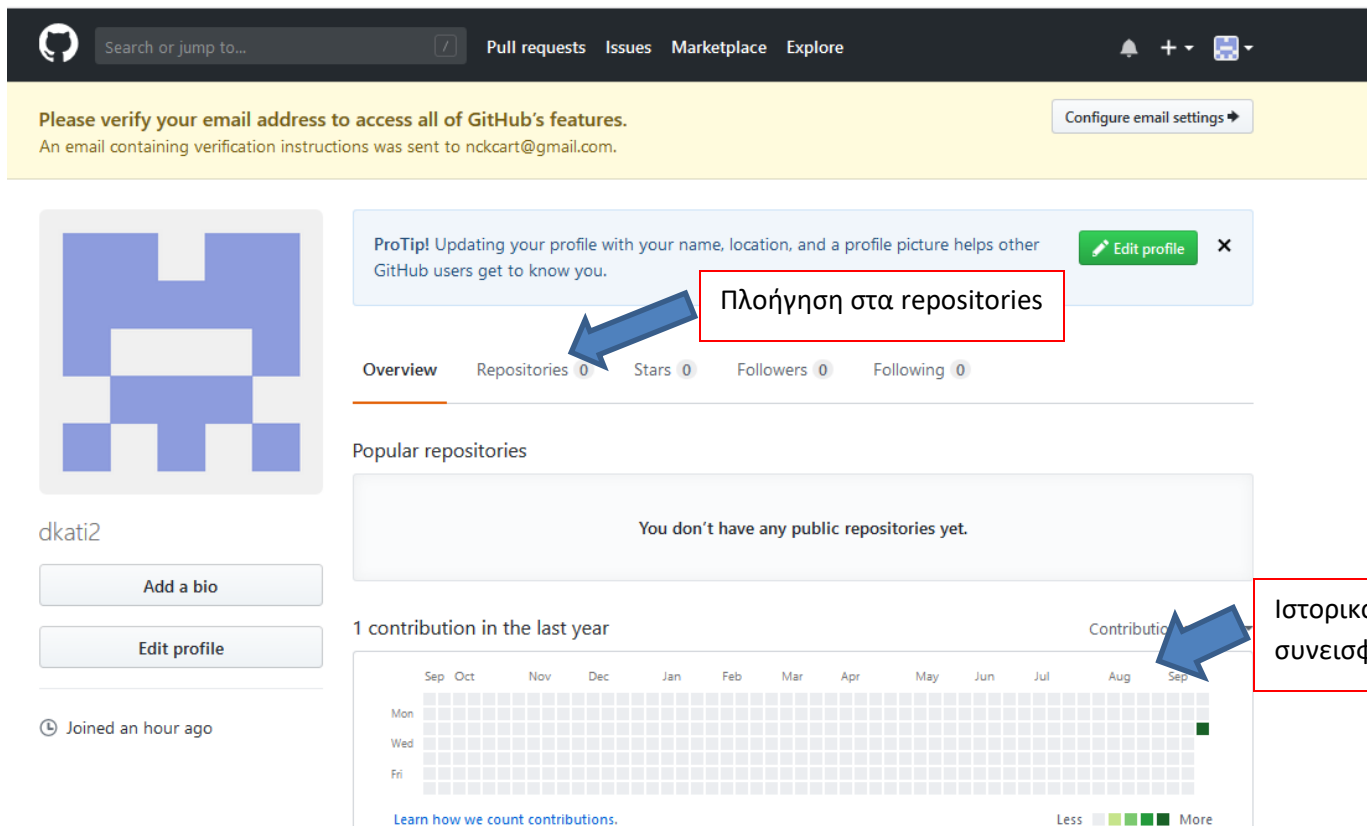
What are you interested in?

matlab ×    c++ ×    python ×    machine-learning ×

Στη συνέχεια πηγαίνουμε στο προφίλ πατώντας πάνω δεξιά στο avatar

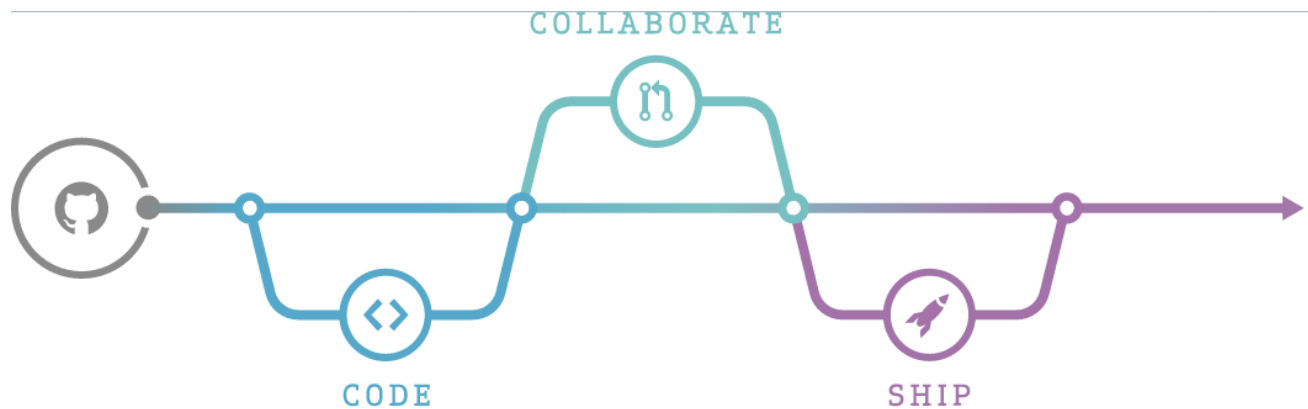


Παρακάτω παρατηρούμε την κύρια σελίδα του λογαριασμού μας.

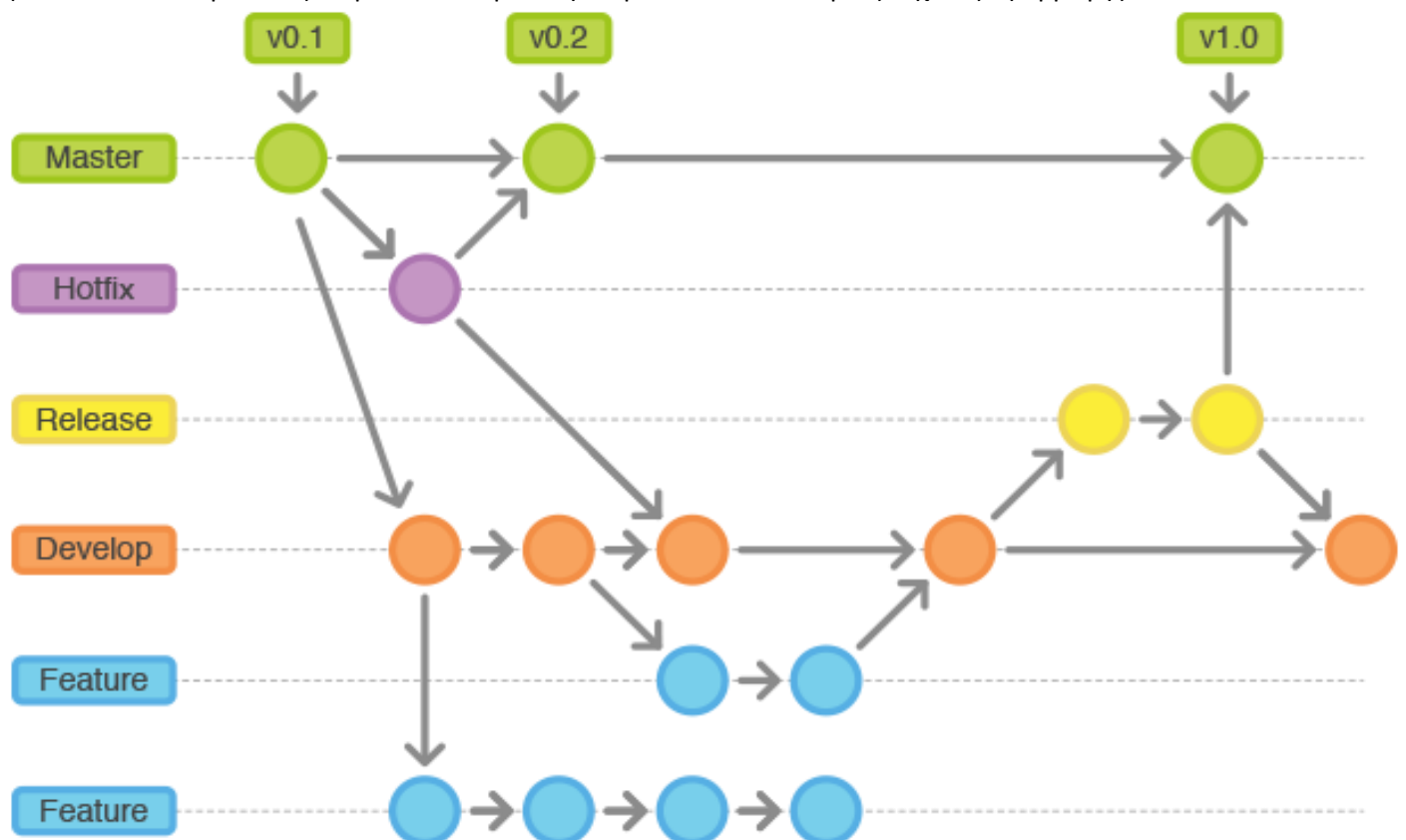


Από τη στιγμή που το προφίλ είναι έτοιμο, μπορούμε να δημιουργήσουμε οποιοδήποτε repository. Πριν από αυτό, θα προηγηθεί μία ανάλυση του τρόπου λειτουργίας του github.

### Τρόπος λειτουργίας και η φιλοσοφία του GitHub



Τη λειτουργία του github μπορούμε να τη φανταστούμε ως ένα δέντρο (tree) με κλαδιά (branches) και κυριο κορμό (master branch), πάνω σε ένα χρονοδιάγραμμα. Για την ακρίβεια ως ένα δέντρο με κερασία (Θα αναλυθεί παρακάτω). Παρακάτω θα μελετήσουμε ένα flowchart μιας τυχαίας εφαρμογής



Στην εικόνα παρατηρούμε το εξής.

- 1 πλαίσιο με ονομα master
- 1 πλαίσιο με ονομα hotfix
- 1 πλαίσιο release
- 1 πλαίσιο develop
- 2 πλαίσια feature
- Καποια κυκλακια
- 6 διακεκομμενες γραμμες
- Βελακια ροης
- 3 κουτακια που αναφερουν την εκδοση της εφαρμογης καθε φορα

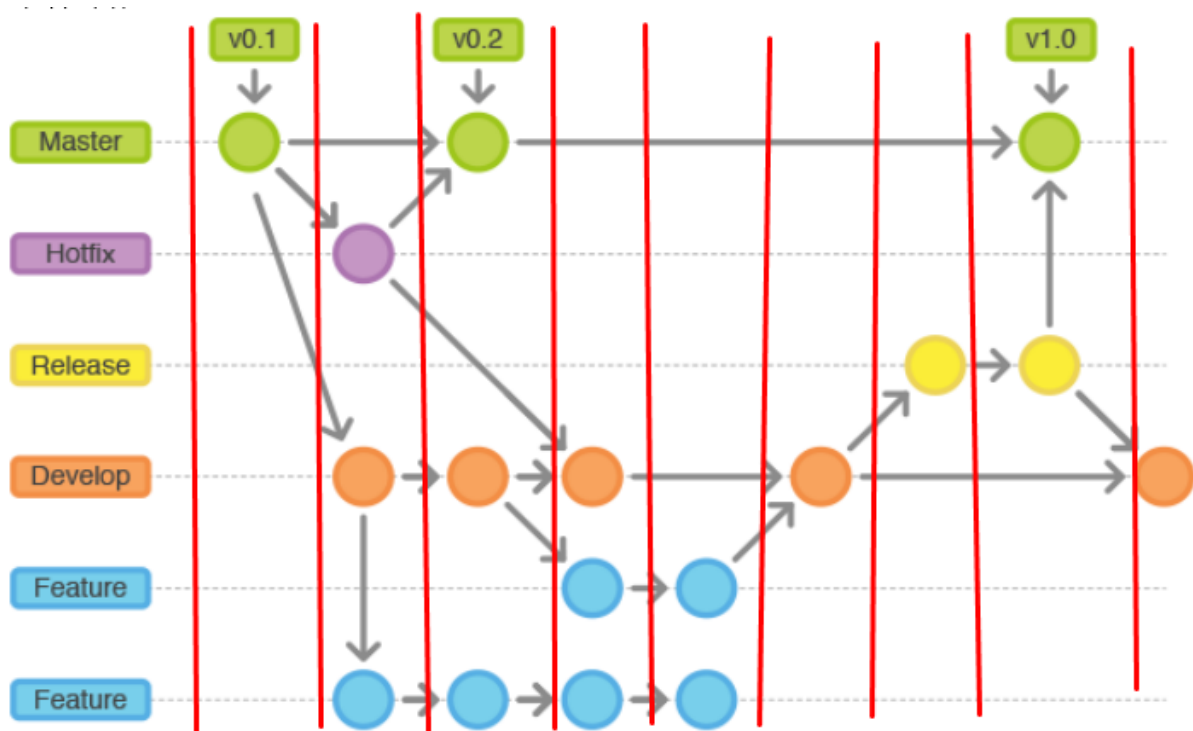
Το κυριο πλαίσιο είναι το master. Είναι το πρώτο branch που δημιουργείται και η κυρία ροή του προγράμματος. Τα υπολοιπα πλαίσια είναι τα δευτερευοντα branches.

Οι διακεκομμενες γραμμες είναι οι οριζοντιες γραμμες που μας δειχνουν την χρονικη εξελιξη των πραγματος απο αριστερα προς τα δεξια.

Τα βελακια ροης δειχνουν τα βηματα που κανει η ροή του github το οποιο θα αναλυσουμε παρακατω.

Τα κυκλακια αντιπροσωπευουν μια αλλαγη(ενα commit) στον κωδικα.

Πολυ σημαντικό είναι να παρατηρησουμε πως τα κυκλακια βρισκονται σε καθετη αντιστοιχια.



Ας το αναλυσουμε αυτο. Στο πρώτο καθετο πλαίσιο υπαρχει ΜΟΝΟ το κυκλακι του master branch. Αυτο σημεινει οτι είναι το ΠΡΩΤΟ-initial release του source code μας στο github. Μπορούμε να

φανταστούμε το master branch ως το «επίσημο» source code που θα θέλαμε κάποιος να δει. Συνήθως είναι συνήθως η stable έκδοση του τρεχόντα κωδικα. Φυσικά το κύριο branch μας μπορεί να ονομάζεται και διαφορετικά.

Στη συνέχεια παρατηρούμε 3 βελακία.

1 προς το hotfix. 1 προς το develop. 1 προς το feature.

Αυτό σημαίνει ότι από το master φτιάξαμε άλλα 3 branches. Βλέπουμε πως τα κυκλακία αυτά δεν είναι στην ίδια καθετο με το master. Αυτό σημαίνει πως έχει γίνει κάποιο commit-κάποια αλλαγή στον κωδικα.

Συνεπώς βλέπουμε ότι ο προγραμματιστής του κωδικα εφτιάξε 3 ακόμα branch για να μπορεί να προσθέσει μια αλλαγή.

*-Και γιατί δεν βάζει την αλλαγή κατευθείαν στο master branch?*

*-Για να μπορεί να την ελέγξει. Αν δουλεύει σωστά τότε την προσθέτει στο master branch που θα δούμε παρακάτω*

Στην τρίτη κάθετη γραμμή βλέπουμε το hotfix να μπαίνει στο master. Αυτό σημαίνει πως το hot fix ήταν πιθανόν ένα bug fix, οπότε ο προγραμματιστής το προς' ορμωσε στο κύριο branch. Επίσης ο προγραμματιστής ανεβάζει την έκδοση του προγράμματος σε v0.2

Στην ίδια τρίτη κάθετη γραμμή παρατηρώ και τα άλλα κυκλακία από develop και feature. Κάθε κυκλακία είναι και ένα commit (μια αλλαγή) στον κωδικα. Αυτό σημαίνει πως αν το κυκλακί που υπάρχει μέσα σε μια κάθετη γραμμή, υπάρχει και σε άλλο branch, τότε **το ΙΔΙΟ ακριβώς** κομμάτι κωδικα υπάρχει και σε αυτό το branch. Με την ίδια λογική προχωράμε στο διαγραμμα και όπου υπάρχουν βελακία που πάνε διαγώνια, σημαίνει πως από εκείνο το σημείο (commit), φτιάχνω νέο branch. Θα τα αναλύσουμε όλα αυτά και παρακάτω

## Παρατηρήσεις

- Το github κρατάει ιστορικό των commits. Αν κάτι γραφτεί στο ιστορικό, ΔΕΝ ΔΙΑΓΡΑΦΕΤΑΙ.
- Το ιστορικό του github μπορεί να αναιρεθεί (Να γίνει reset) ή να γίνει Revert
  - Revert σημαίνει να γυρίσω τον κωδικα πως ήταν πριν.  
Αν πχ, εσβησα ένα declaration μιας μεταβλητής, τότε με το revert τη δηλώνει ξανά
- Ολοκληρω το source code με τα όλα τα branches και το commit history ονομάζεται repository (repo)
- Το ότι κάνω push μια αλλαγή/ένα commit δεν σημαίνει πως αλλάζω branch. Ο λόγος που αλλάζω branch είναι για να μπορώ να κάνω δοκιμές στον κωδικα χωρίς να επηρεάζω τον βασικό κωδικα που θεωρώ stable
- Μέσα σε ένα repository μπορούμε να προσθέσουμε contributors και να έχουν δικαίωμα να αλλάξουν τον κωδικα. Οπότε την επομένη φορά που θα θέλω να κάνω κάποιες αλλαγές, θα πρέπει να «τραβήξω» τις αλλαγές του άλλου contributor




### Ξεκινώντας νέο repository για να ανεβάσουμε τον κώδικα ενός project που έχουμε ήδη γράψει

Μεταφερομαστε στο προφιλ μας στο github.com. Αφου εχουμε ηδη κανει login , επιλεγουμε το + απο πάνω δεξια και επιλεγουμε New repository. Εκει μπορουμε να συμπληρωσουμε τα στοιχεια του αρχικου repository. Αρχικα θα ειναι αδειο και μετα θα κανουμε προσθηκη του κωδικα μας ως initial release. Γραφουμε το ονομα του repository , μια περιγραφη εαν θελουμε, Δημοσιο repository και επιλεγουμε και το Initialize this repository with a README για να δημιουργησει το πρωτο μας αρχειο. Αργότερα θα προσθεσουμε και το license.

---

Owner

Repository name

 dkati ▾

 / 


myTestRepo ✓

Great repository names are short and memorable. Need inspiration? How about **laughing-bassoon**.


Description (optional)

This repo is made for tutorial purposes

---

☒  **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

Αφου πατησουμε Create repository θα δουμε την εικονα αυτη

dkati / myTestRepo

Unwatch1Star0Fork0

<> CodeIssues0Pull requests0Projects0WikiSettingsInsights

This repo is made for tutorial purposes

Edit

Add topics

1 commit1 branch0 releases1 contributor

Branch: masterNew pull requestCreate new fileUpload filesFind fileClone or download

Initial commitLatest commit 5a6f92a just now

README.mdInitial commitjust now

README.md

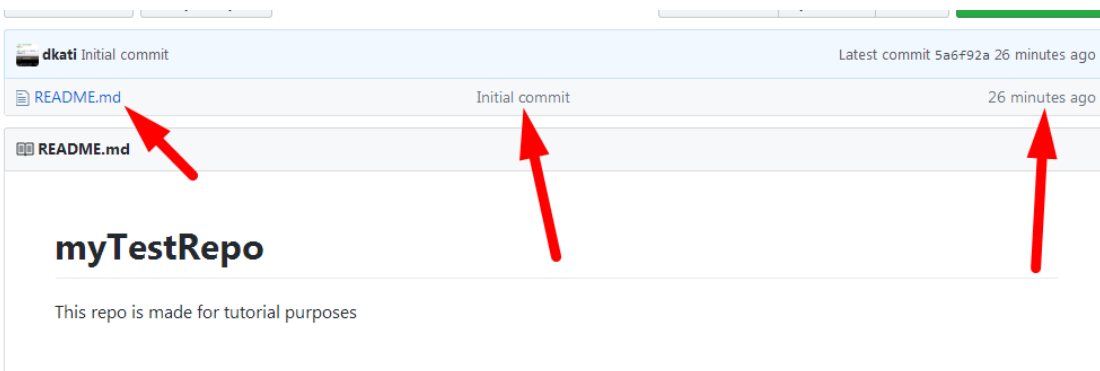
# myTestRepo

This repo is made for tutorial purposes

Παρατηρούμε τα εξής στοιχεία

- Πάνω αριστερά βλέπουμε το όνομα του χρήστη και το όνομα του repository.  
dkati/myTestRepo
- Από κάτω υπάρχει ένα μενού.
  - Code : Το κύριο και πιο σημαντικό μενού
  - Issues : Στη καρτέλα αυτή μπορεί οποιοσδήποτε χρήστης (ακόμη και αυτός που δεν είναι contributor) να κάνει αναφορά κάποιων θεμάτων-σφαλμάτων, και οι contributors να τα δουν και να απαντήσουν/λυσουν τα ζητήματα
  - Pull requests : Στη καρτέλα αυτή είναι μαζεμένες κάποιες προτάσεις που κάποιος τρίτος χρήστης κάνει, σχετικά με τον κώδικα. Οι contributors βλέπουν τις αλλαγές και αν θέλουν με το πατήμα ενός κουμπιού επιτρέπουν το τρίτο χρήστη να προσθέσει τα commit του στο repository μας, χωρίς να είναι μέλος αυτού
  - Wiki : Το παραδοσιακό wiki όπου οι contributors δίνουν κάποιες οδηγίες σχετικά με το source code
  - Settings : Ρυθμίσεις του repository (Όχι του source code)
- Παρακάτω είναι το description που έχουμε βάλει στο repository
- Στη συνέχεια υπάρχει άλλο ένα μενού

- 1 Commit : Πατώντας πάνω στο μενού αυτό μας εμφανίζει όλα τα commits που έχουν γίνει με χρονολογική σειρά. Λεπτομέρειες για τα commits θα αναφερθούν αργότερα
- 1 branch : εμφανίζει τα branches
- 0 releases : αφορά τα releases που κάνουν οι contributors
- 1 contributor : εμφανίζει όλους όσους συνεισφέρει στον κώδικα είτε είναι μέλη του repository είτε χρησιμοποιήσαμε τον κώδικα του
- Παρακάτω βλέπουμε ένα κουμπι-μενού που λέει branch: master. Από εδώ μπορώ να αλλάζω branches και να βλέπω τον αντίστοιχο κώδικα και τα αντίστοιχα commits.
- New pull request: Εάν θέλω ως τρίτος χρήστης να προσθέσω κώδικα
- Create new file/Upload files : Χειροκίνητη δημιουργία/δημοσίευση αρχείου (**Δεν συνιστάται**)
- Clone/Download : Κατέβασμα του source code σε zip μορφή (Δεν συνιστάται! Ένας λόγος που δεν συνιστάται είναι ότι στα Linux περιβάλλοντα, το zip αρχείο μπορεί να διαγράψει τυχόν symlinks κατά το extract
- **dkati** Initial commit : Αναφέρεται το τελευταίο commit που έχει γίνει. Το github αυτόματα κάνει ένα commit όταν δημιουργούμε το repository και προσθέτουμε readme. Η μορφή του τίτλου του commit είναι  
<githubusername><Τίτλος commit>
- Όλα τα υπόλοιπα από κάτω είναι τα αρχεία του συγκεκριμένου directory μαζί με τις λεπτομέρειες



Το αριστερό βελάκι μας δείχνει τα αρχεία που έχει το συγκεκριμένο directory

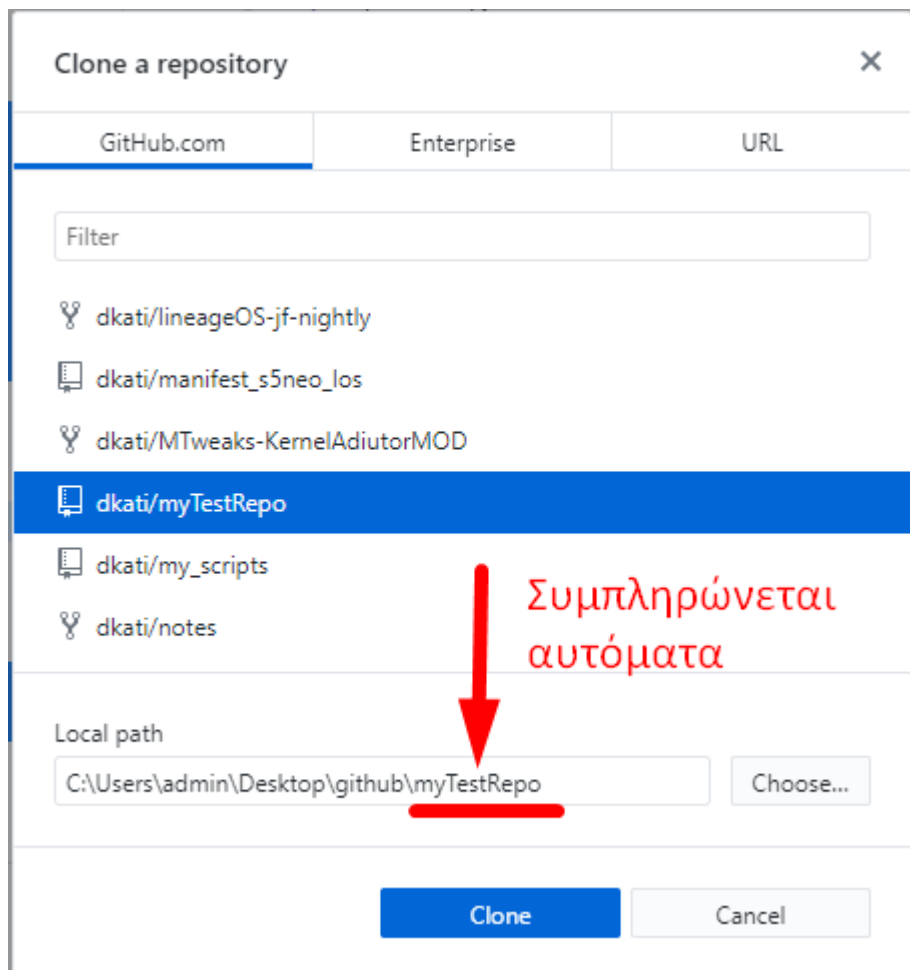
Το μεσαίο βελάκι δείχνει το τελευταίο commit που έχει γίνει **και έχει επηρεάσει το συγκεκριμένο αρχείο**

Το δεξιά βελάκι δείχνει την ώρα που έχει περάσει από το τελευταίο commit που επηρέασε το αντίστοιχο αρχείο.

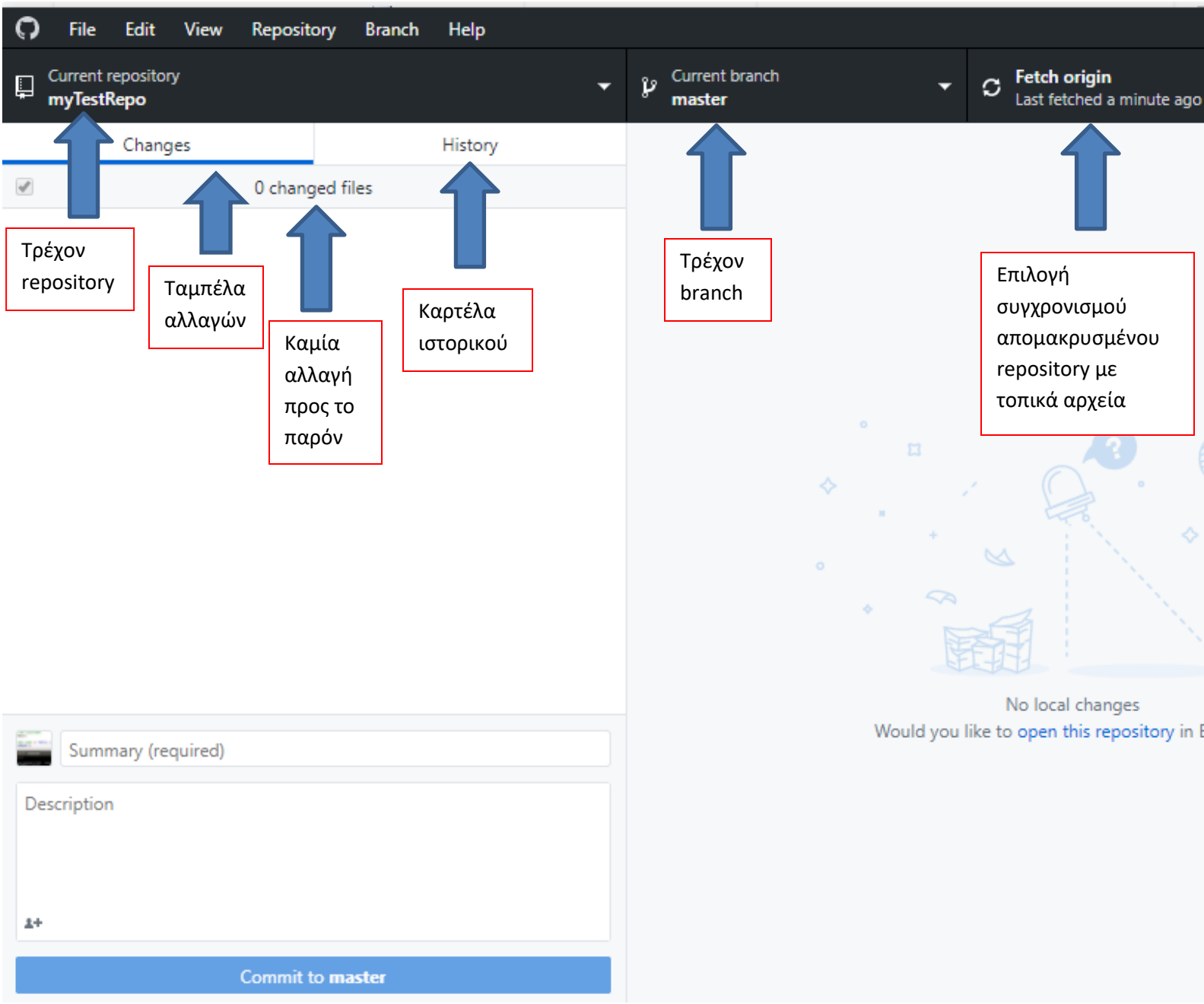
Γενικότερα οι πληροφορίες αυτές μας βοηθούν να έχουμε μια τάξη μέσα σε τεραστίου μεγέθους projects.

## Συνδέοντας το repository με τα τοπικά μας αρχεία

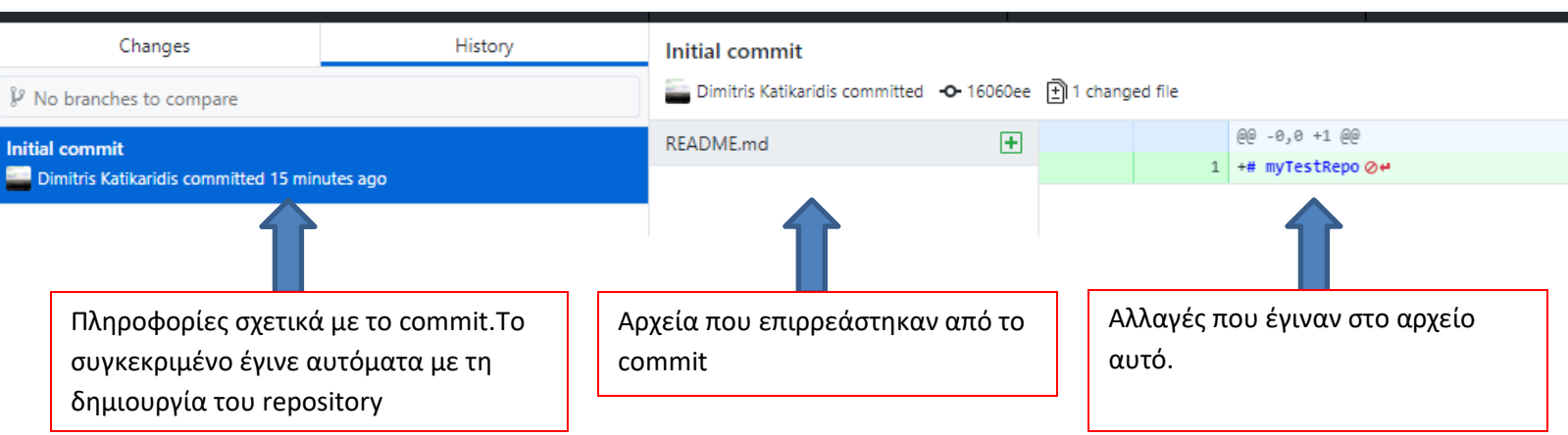
Από τη στιγμή που το απομακρυσμένο repository έχει δημιουργηθεί, το μόνο που απομένει, είναι ο συγχρονισμός του με τα τοπικά μας αρχεία. Εκτελούμε την εφαρμογή που έχουμε κάνει εγκατάσταση. Συνδέουμε το λογαριασμό μας και προχωράμε. Επιλέγουμε από το μενού επιλογών File->Clone repository. Στο path που ζητάει καλό θα είναι να έχουμε έναν φάκελο κάπου τοπικά ώστε να μπορούμε να έχουμε όλα τα github project μας μαζεμένα σε ένα σημείο. Κατά το clone θα δημιουργηθεί ένας φάκελος μέσα στο path που θα επιλέξουμε. Συνεπώς δε χρειάζεται να φτιάξουμε και άλλον φάκελο μέσα σε αυτό για κάθε ξεχωριστό project. Επιλέγουμε το repository που μόλις φτιάξαμε.



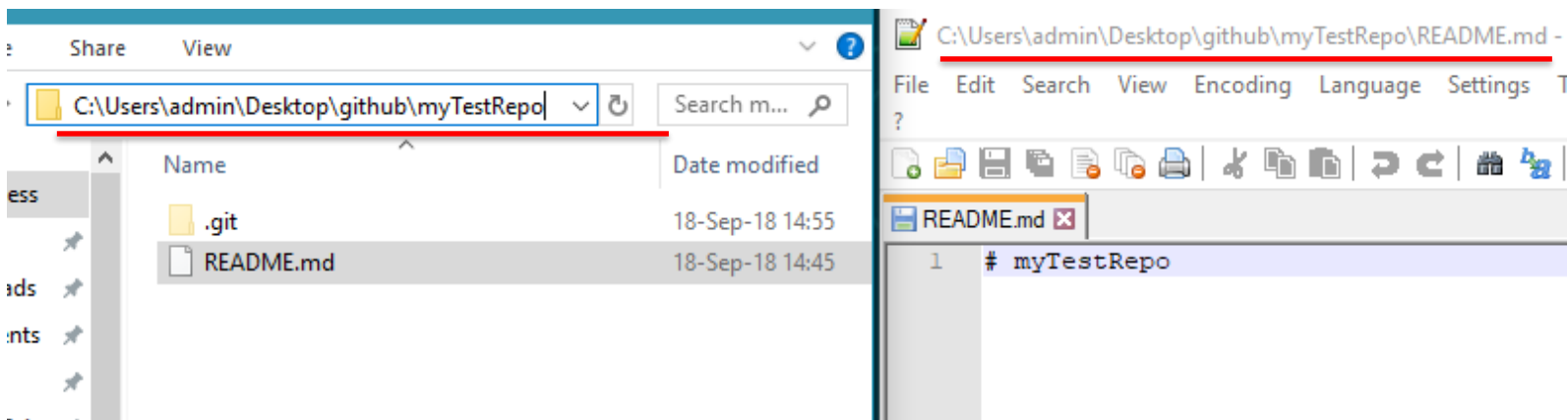
Στη συνέχεια πατάμε clone και περιμένουν να κατέβει το απομακρυσμένο repository. Όταν ολοκληρωθεί, θα έχουμε μία εικόνα σαν την παρακάτω.



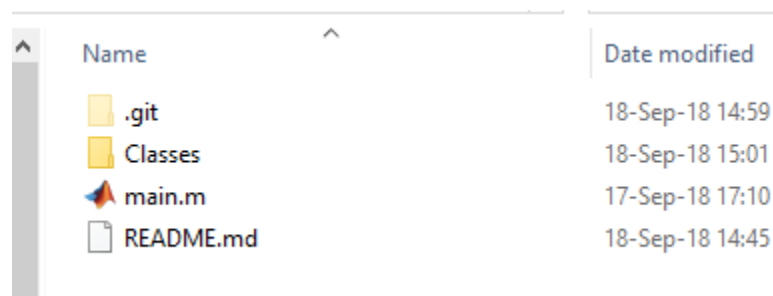
Πατώντας στο ιστορικό βλέπουμε την παρακάτω εικόνα



Αν πάμε στο path που δώσαμε κατά το clone του repository θα δούμε τα αρχεία από το απομακρυσμένο repository.

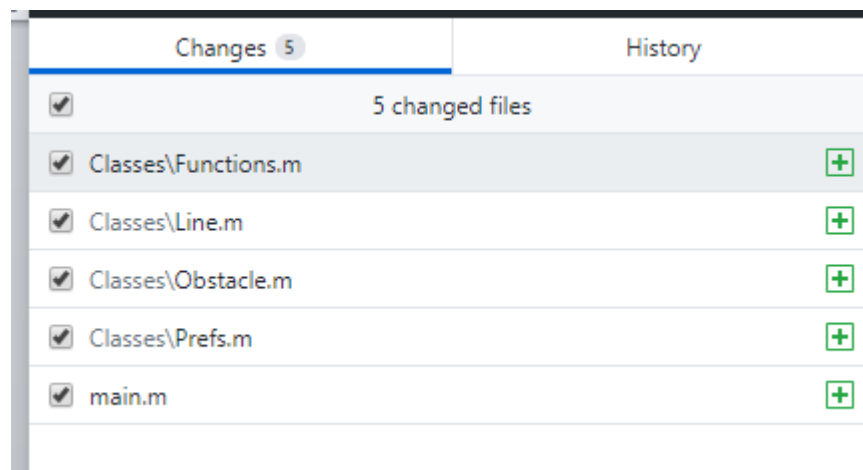


Έστω λοιπόν ότι θέλουμε να ανεβάσουμε στο repository το μέχρι τώρα project μας. Προσθέτουμε τα αρχεία μας μέσα στο φάκελο **myTestRepo** και μεταφερόμαστε στην εφαρμογή του GitHub.



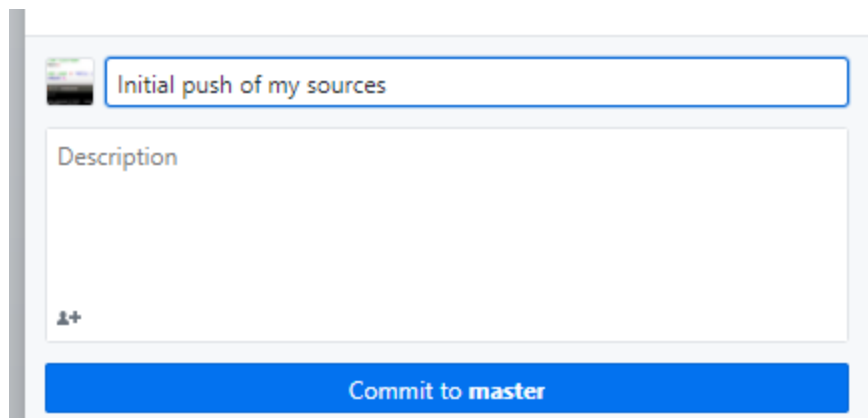
Ας υποθέσουμε ότι το project μας περιέχει έναν φάκελο με όνομα **Classes** και το αρχείο MatLab **main.m**. Πηγαίνοντας στην εφαρμογή του GitHub, στην καρτέλα «Changes», παρατηρούμε ότι η εφαρμογή έχει αναγνωρίσει τις αλλαγές μας όπως φαίνεται

παρακάτω.



Επιλέγοντας κάθε αρχείο, βλέπουμε το τι έχει προσθεθεί. Φυσικά, από τη στιγμή που ανεβάζουμε ολόκληρο αρχείο, το github «αντιλαμβάνεται» ότι ανεβάζουμε καινούριες σειρές. Συνεπώς όλο το αρχείο είναι στην ουσία πολλές προσθήκες νέων σειρών.

Στη συνέχεια βάζουμε κάτω αριστερά ένα commit title και πατάμε το κουμπί «commit to master» .

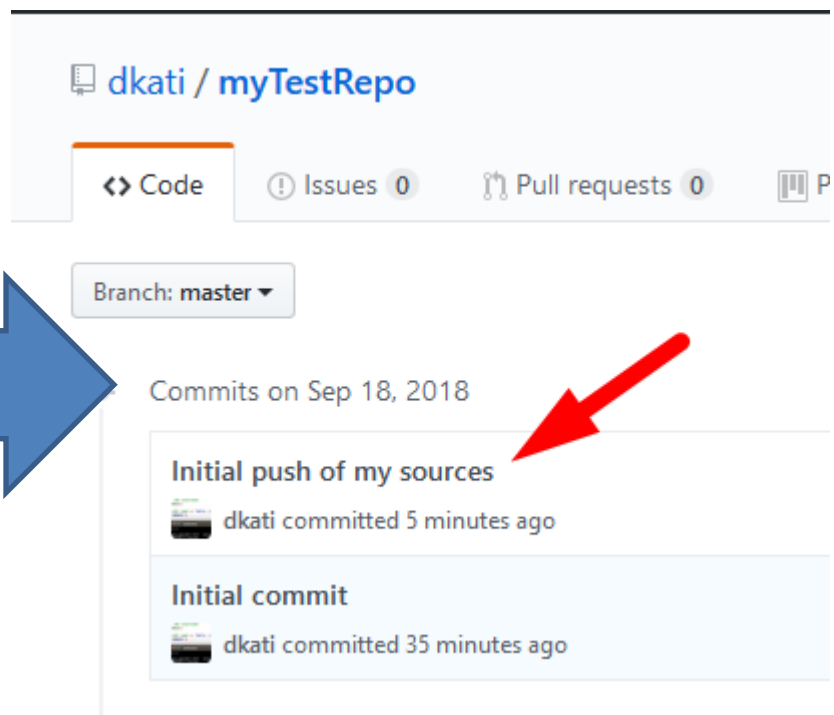
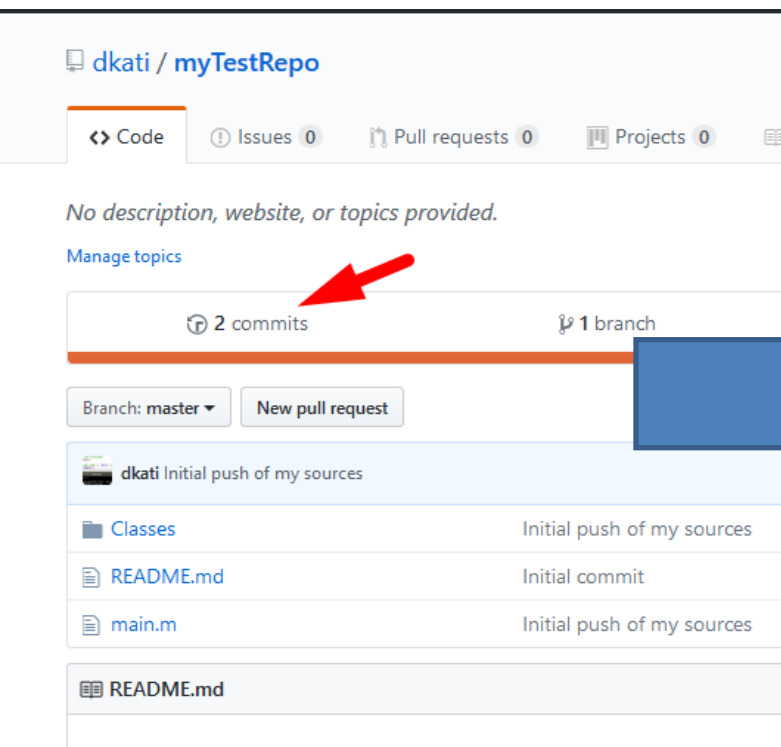


Φυσικά στην περιγραφή μπορούμε να βάλουμε ότι θέλουμε ή και τίποτα. Μόλις πατήσουμε το κουμπί, θα παρατηρήσουμε ότι αφενώς έχουν εξαφανιστεί οι αλλαγές από την εφαρμογή και αφετέρου το η επιλογή από την μπάρα επιλογών έχει αλλάξει από «fetch origin» σε «push origin».

Επίσης παρατηρούμε και ένα βελάκι με τον αριθμό '1'. Αυτό σημαίνει πως έχουμε 1 ΤΟΠΙΚΟ commit το οποίο πρόκειται να κάνουμε push στο απομακρυσμένο repository.

**Best practice : Κάνουμε push , ένα-ένα τα commits για να αποφύγουμε conflicts**

Πατώντας το push origin η εφαρμογή θα αποστείλει την αλλαγή μας στο GitHub. Όπως είδαμε και πριν, επιλέγουμε την καρτέλα “X commits” από τη σελίδα του repository για να δούμε το commit.



Πατώντας πάνω στο commit βλέπουμε τις αλλαγές που προκαλέσαμε. Όταν μπορούμε μέσα στο commit επιλέγουμε το 'X files changed' για να δούμε ποια αρχεία επηρεάστηκαν.

The screenshot shows a GitHub commit page for user 'dkati'. The commit message is 'Initial push of my sources'. The commit hash is '1c0b24b1fd9d807f09ee89aed75e65ccf99a7a38'. The page shows 5 changed files with 358 additions and 0 deletions. The files listed are:

- Classes/Functions.m (+172 -0)
- Classes/Line.m (+44 -0)
- Classes/Obstacle.m (+16 -0)
- Classes/Prefs.m (+16 -0)
- Classes/Functions.m (+110 -0)

The file 'Classes/Functions.m' is selected, showing a diff with 172 additions and 0 deletions. The diff content is:

```
@@ -0,0 +1,172 @@
1 + classdef Functions
2 +     methods (Static)
3 +         function printGoogle(table_)
4 +             latRow1=[];
5 +             lonRow1=[];
```

Annotations on the page:

- Top navigation bar: Code, Issues (0), Pull requests (0), Insights, Settings.
- Commit message: 'Initial push of my sources'.
- Commit hash: '1c0b24b1fd9d807f09ee89aed75e65ccf99a7a38'.
- Summary: 'Showing 5 changed files with 358 additions and 0 deletions.'.
- File list: 'Classes/Functions.m', 'Classes/Line.m', 'Classes/Obstacle.m', 'Classes/Prefs.m'.
- Diff view: 'Classes/Functions.m'.

Annotations (in red boxes):

- Top navigation bar: 'Όνομα commit', 'Περιγραφή (αν υπάρχει)', 'Ποιο branch', 'Ποιος', 'Πριν πόση ώρα'.
- Summary: 'Αριθμός νέων και αφαιρούμενων σειρών κώδικα'.
- File list: 'Το σύμβολο + σημαίνει «νέο αρχείο». Αντίστοιχα '-' σημαίνει «διαγραφή αρχείου» και κίτρινη τελεία σημαίνει «τροποποίηση αρχείου»'.
- Diff view: 'Αριθμός νέων και αφαιρούμενων σειρών κώδικα του εκάστοτε αρχείου'.

Με αυτόν τον τρόπο έχουμε πλέον όλα τα τοπικά μας αρχεία , συγχρονισμένα με το απομακρυσμένο repository.



## Το πρώτο πραγματικό commit και push

Το πιο σημαντικό σημείο είναι να καταλάβουμε τι θέλουμε να κάνουμε push και τότε θέλουμε να το κάνουμε push. Ένα commit πρέπει να είναι καθαρό. Αυτό σημαίνει ότι το κάθε commit πρέπει

- Να έχει ξεκάθαρο τίτλο που θα περιγράφει τι ακριβώς κάνει
- Αν είναι απαραίτητο να έχει μια περιγραφή με λεπτομέρειες
- **Να μην περιεχει αλλαγες σε αρχια που δεν αφορουν την κυρια αλλαγη και σκοπο του commit.**
- Να μην είναι αντιγραφή από άλλο commit. Αν θελω το commit καποιου τριτου θα το κανω με τον νόμιμο τρόπο που θα δούμε παρακάτω.

Ας εξηγήσουμε την 3<sup>η</sup> περίπτωση.

Κατά την εξοικίωση μας με το github, κάνουμε push πράγματα τα οποία δεν πρέπει να γίνουν push.

### Παράδειγμα-Σενάριο

*Σε ένα αρχείο αλλάζω 2 μεταβλητές από **false** σε **true**  
και στο ίδιο αρχείο αλλάζω το όνομα μιας συνάρτησης.*

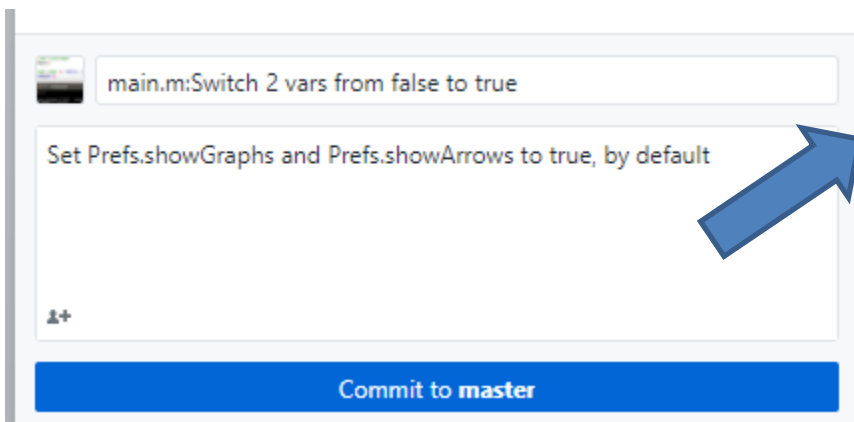
Στο commit title βάζω **“main.m:Switch 2 vars from false to true”** με περιγραφή **«Set Prefs.showGraphs and Prefs.showArrows to true, by default»**. Παρόλαυτα υπάρχει και η αλλαγή του ονόματος της συνάρτησης και θα γίνει και αυτό push. Κατά το οποίο δεν θα θέλαμε καθώς είπαμε πως το commit title πρέπει να περιγράφει ΠΛΗΡΩΣ την αλλαγή που προκάλεσε το commit.

Η σωστή κίνηση θα είναι να κάνω πρώτα την αλλαγή στις 2 μεταβλητές, να κάνω το push και στη συνέχεια, σε δεύτερο commit, να αλλάξω όνομα στη συνάρτηση.

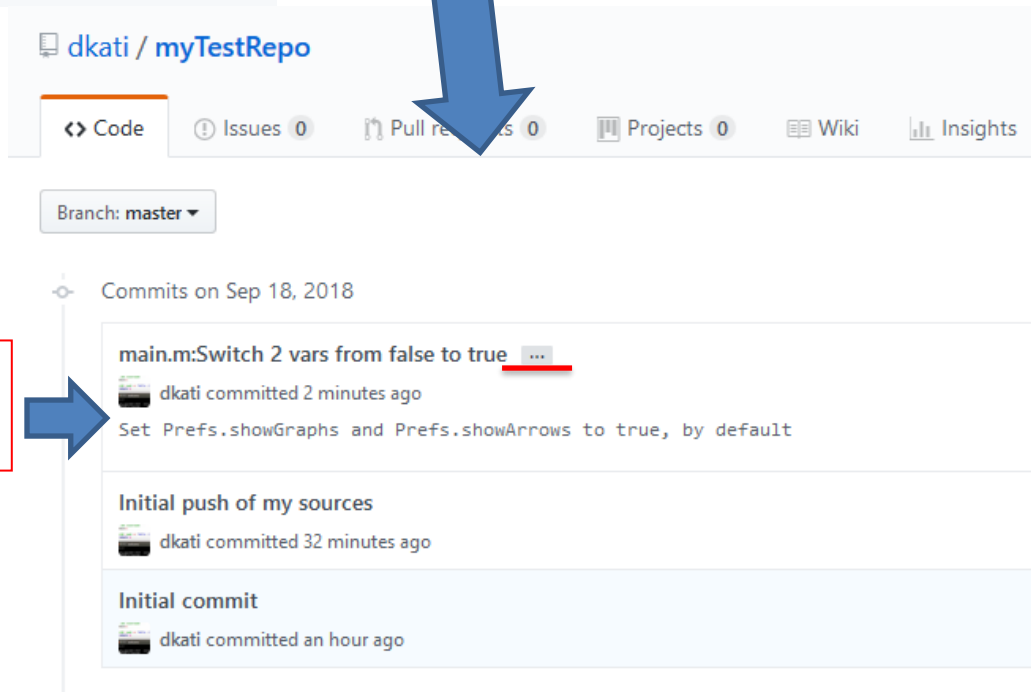
Κάνουμε την αλλαγή στα αρχεία μας και μεταφερόμαστε στην εφαρμογή του github.

Changes 1		History		main.m	
1 changed file				8	8 @@ -8,8 +8,8 @@ addpath('./Grid');
main.m				9	9 %=====Global settings=====
				10	10 Prefs=Prefs();
				11	11 -Prefs.showGraphs = false;
				12	12 -Prefs.showArrows = false;
				11	+Prefs.showGraphs = true;
				12	+Prefs.showArrows = true;
				13	13 Prefs.showEdges = true;
				14	14 %=====
				15	15

Παρατηρούμε ότι έχει αναγνωρίσει τις αλλαγές μας. Όποτε συμπληρώνουμε σωστά το commit title και description επιλέγουμε το «commit to master» και κάνουμε το push πατώντας το κουμπι push πάνω από την εργαλειοθήκη.



Φυσικά το commit φαίνεται και στη σελίδα του github.



Πατώντας στις 3 γκρι τελείες μας εμφανίζεται το commit description.

Συνεπώς από εκεί και πέρα μπορούμε να κάνουμε μετανομασία τη συνάρτηση που θέλουμε και να κάνουμε ένα ακόμη commit με τίτλο έστω «**main.m:Rename function1 to myfunction1**»

Αντιμετώπιση του παραπάνω σεναρίου σε περίπτωση που κάνουμε καταλάθος ΚΑΙ την αλλαγή του ονόματος συνάρτησης μέσα σε ένα commit

Έστω ότι έχει γίνει αλλαγή και των μεταβλητών αλλά και μετανομασία της συνάρτησης.

The screenshot shows a Git commit interface with the following components:

- Changes 1**: 1 changed file (main.m)
- History**: (empty)
- main.m**: The diff view showing changes to the file.
- Commit Message**: main.m:Switch 2 vars from false to true
- Commit Action**: Commit to master

The diff for main.m shows the following changes:

Line	Original	Modified
8	8	@@ -8,8 +8,8 @@ addpath('./Grid');
9	9	%=====Global settings=====
10	10	Prefs=Prefs();
11	-Prefs.showGraphs = true;	+Prefs.showGraphs = false;
12	-Prefs.showArrows = true;	+Prefs.showArrows = false;
13	Prefs.showEdges = true;	Prefs.showEdges = true;
14	%=====	%=====
15	@@ -94,7 +94,7 @@ if ( ~isPossible )	@@ -94,7 +94,7 @@ if ( ~isPossible )
94	end	end
95	% creation of spare matrix DG	% creation of spare matrix DG
96	-[a,b]=size(E);	+ [a,b]=getSize(E);
97	WeightMATRIX=ones(1,a);	WeightMATRIX=ones(1,a);
98	Node1MATRIX=E(:,1);	Node1MATRIX=E(:,1);
99	Node2MATRIX=E(:,2);	Node2MATRIX=E(:,2);
100		

Παρατηρούμε ότι οι αλλαγές περιέχουν και την μετανομασία. Επιλέγοντας τις 2 σειρές (που έχουν τον αριθμό 97) τότε τις βγάζω εκτός του συγκεκριμένου commit. Με αυτόν τον τρόπο μπορούμε να αφαιρέσουμε αυτήν την αλλαγή απο το συγκεκριμένο commit. Όταν γίνει το push τότε τις επιλέγω ξανά ώστε να τις κρατήσει στο νέο-δεύτερο commit που θα κάνω για την μετανομασία.

Repository → Αποθετήριο κώδικα.περιέχει τα αρχεία μας μαζί με το ιστορικό αλλαγών και τα branches

Branch → Η διακλάδωση του project. Υπάρχει για να μπορούμε να έχουμε πολλές εκδόσεις του ίδιου project

Public/private repository → Public ονομάζεται το repository που είναι διαθέσιμο προς όλο τον κόσμο. Μπορεί να δει (όχι να επεξεργαστεί) τον κώδικα μας και το ιστορικό του. Private ονομάζεται το repository που **ΔΕΝ** είναι διαθέσιμο προς όλο τον κόσμο. Αφορά μονάχα τα άτομα που έχει προσκαλέσει ο δημιουργός του.

Organization → Η ομπρέλα κάτω από την οποία μπορούμε να βάλουμε πολλά repositories σαν ομάδα.

Push → Μεταφόρτωση μιας αλλαγής από τα τοπικά αρχεία μας, στο απομακρυσμένο repository

Pull → Μεταφόρτωση μιας αλλαγής από το απομακρυσμένο repository ,στα τοπικά μας αρχεία.Αυτο συνήθως συμβαίνει όταν παραπάνω από ένας developer επιρρεάζει αρχεία από το repository. Όλοι οι υπόλοιποι πρέπει να κάνουν pull

Clone → Μεταφόρτωση ΟΛΟΚΛΗΡΟΥ repository στον υπολογιστή μας

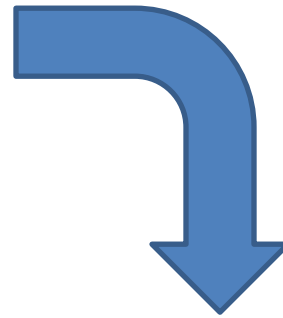
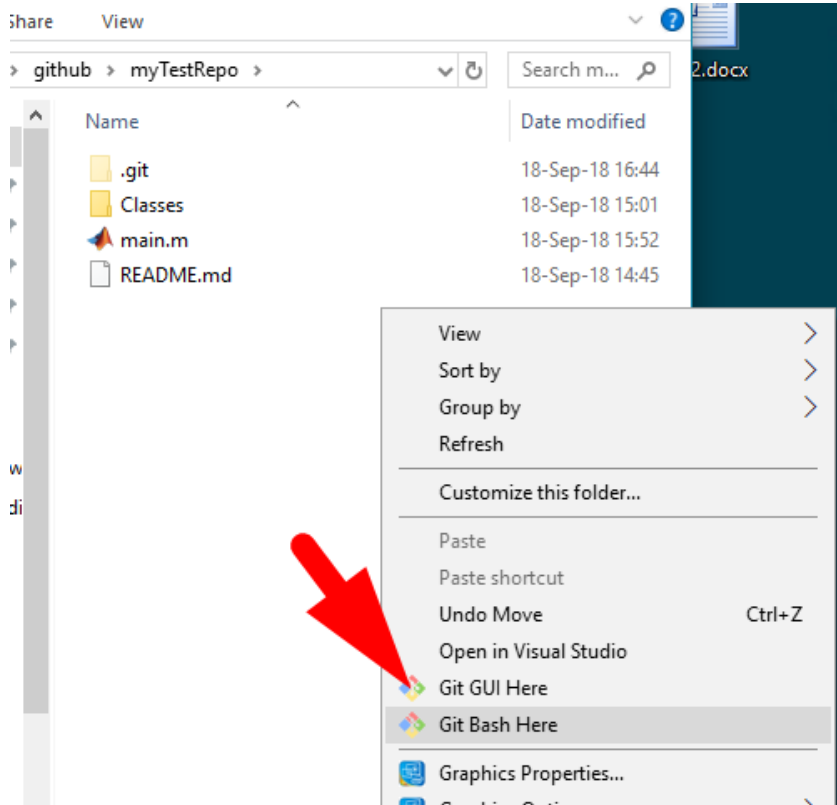
Revert → Αναστροφή ενός commit. Πρακτικά, αναστρέφει το αποτέλεσμα ενός commit

Origin → Το github application, στο υπόβαθρο, δουλεύει με χρήση ssh. Origin ονομάζεται ο μεσολαβητής της επικοινωνίας του Η/Υ μας υπολογιστή με το απομακρυσμένο repository

Fetch ή sync → Μεταφόρτωση όλων των αλλαγών από το απομακρυσμένο repository στα τοπικά αρχεία μας.

## Προχωρημένες εντολές και χρήση τερματικού

Σε αντίθεση με το τερματικό του Linux, η εφαρμογή του Windows GitHub έχει περιορισμένες δυνατότητες. Η πλήρης χρήση του github απαιτεί εντολές από το git bash (<https://gitforwindows.org/> )



```
MINGW64:/c:/Users/admin/Desktop/github/Thesis/Thesis_Katikaridis
admin@DESKTOP-1NDRQBL MINGW64 ~/Desktop/github/Thesis/Thesis_Katikaridis (master)
$ |
```

Παρακάτω αναγράφονται οι εντολές που χρειαζόμαστε

# GIT CHEAT SHEET

presented by TOWER › Version control with Git - made easy



## CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repo.git
```

Create a new local repository

```
$ git init
```

## LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

*Don't amend published commits!*

```
$ git commit --amend
```

## COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

Show changes over time for a specific file

```
$ git log -p <file>
```

Who changed what and when in <file>

```
$ git blame <file>
```

## BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

## UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository, named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD

```
$ git pull <remote> <branch>
```

Publish local changes on a remote

```
$ git push <remote> <branch>
```

Delete a branch on the remote

```
$ git branch -dr <remote/branch>
```

Publish your tags

```
$ git push --tags
```

## MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

*Don't rebase published commits!*

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
```

```
$ git rm <resolved-file>
```

## UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit

...and discard all changes since then

```
$ git reset --hard <commit>
```

...and preserve all changes as unstaged changes

```
$ git reset <commit>
```

...and preserve uncommitted local changes

```
$ git reset --keep <commit>
```

