



Εγχειρίδιο χρήσης GitHub   
μέσω Linux Terminals  
ROS-autom 2017 ®

Github

# προετοιμασια συστηματοσ

Επαληθευουμε οτι εχουμε τα απαραιτητα πακετα

**sudo apt-get install git  
sudo apt-get install repo**

Το πιο πιθανο ειναι να μην υπαρχει μονο το repo και να απαιτει εγκατασταση.Στη συνεχεια πρεπει να συνδεθει το github client του υπολογιστη μας,με το github στη σελιδα,ωστε να μπορω στη συνεχεια να κανω commits,pulls,pushes κτλπ.

Τρεχουμε

**git config –global user.email** [**dkatikaridis@gmail.com**](mailto:dkatikaridis@gmail.com) **(((διπλές παυλες)  
git config –global user.name mygithubname**

Tο mail και ο κωδικος πρεπει να ειναι σε πληρη αντιστοιχεια με αυτα που βλεπουμε στη σελιδα.  
τα δικα μου ειναι [dkatikaridis@gmail.com](mailto:dkatikaridis@gmail.com) με username dkati

Στη συνεχεια πρεπει να πιστοποιησουμε στο github ,τον υπολογιστη που θα κανει push   
απο το λογαριασμο μας.

**ssh-keygen -t rsa -b 4096 -C** [**dkatikaridis@gmail.com**](mailto:dkatikaridis@gmail.com)

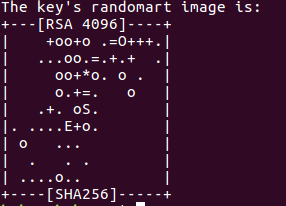
Και θα παρουμε ως επιστροφη

**Generating public/private rsa key pair.**

**Enter file in which to save the key  
παταμε enter**

**Enter passphrase   
παλι enter**

**Enter passphrase again**

**Παλι enter  
Τελικα θα μα δωσει το RSA 4096**  


Στη συνεχεια πρεπει αυτο το ssh key να γραφτεί στον ssh-agent.

**eval “$(ssh-agent -s)”**

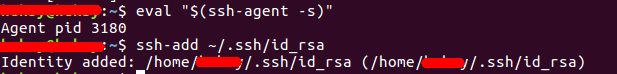
Θα παρουμε μια εξοδο σαν αυτη : **Agent pid XXXXX**

Αποθηκευουμε το ssh σε ενα τοπικο αρχειο

**ssh-add ~/.ssh/id\_rsa**

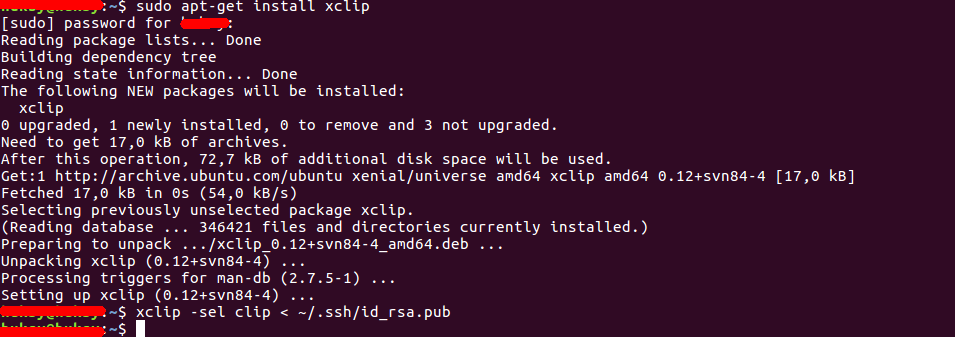
Με εξοδο

**Identity added : /home/username/.ssh/id\_rsa ( home/username/.ssh/id\_rsa )**

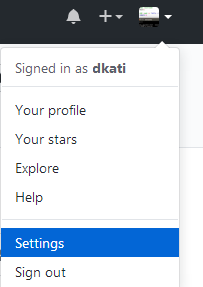


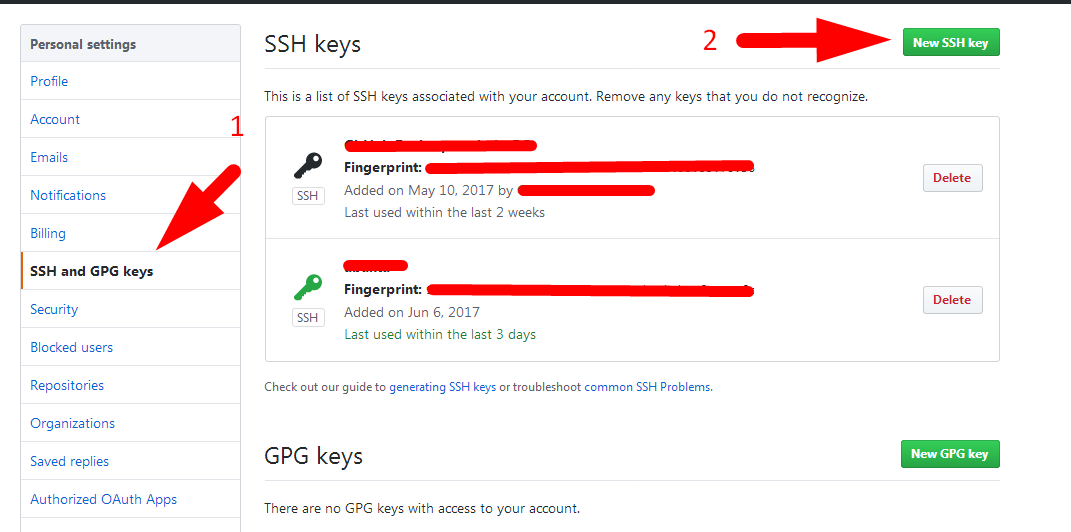
Στη συνεχεια πρεπει να αντιγραψουμε το SSH στο clipboard.Ετσι κατεβαζουμε το εργαλειο xclip   
και αντιγραφουμε το SSH μεσω αυτου,στο clipboard

**sudo apt-get install xclip && xclip -sel clip < ~/.ssh/id\_rsa.pub**

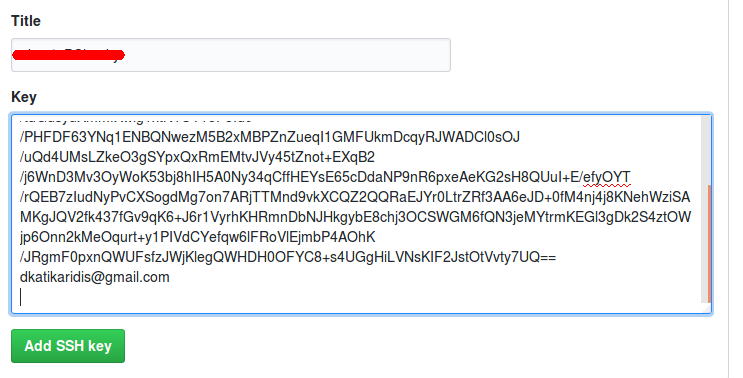


Στη συνεχεια αυτο το ssh πρεπει να προσθεθει στο λογαριασμο μας στο github. Ανοιγουμε τις ρυθμισεις του λογαριασμου μας

  
Επιλεγουμε απο τα μενου , SSH and GPG keys και παταμε στο δεξια κουμπι «New SSH key».  
Bαζουμε στο Title εναν τιτλο και στο πλαισιο key απλα παταμε δεξι κλικ/επικολληση



To τελικο αποτελεσμα θα πρεπει να ειναι ετσι



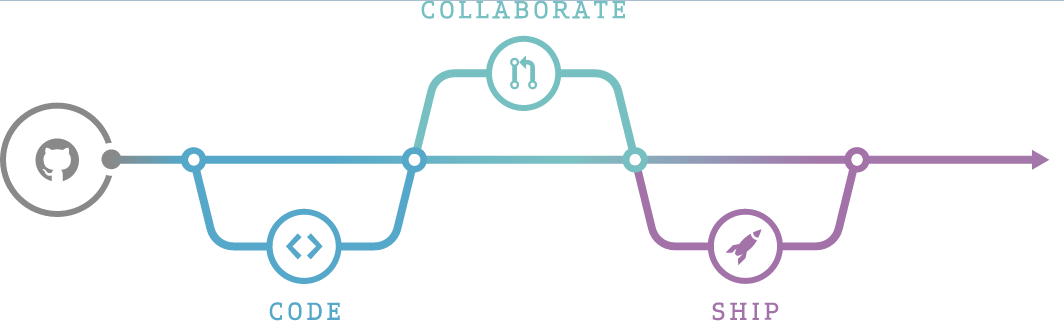
Και παταμε Add SSH key.

Ενδεχομενως να ζητησει κωδικο προσβασης.Αν οντως ζητησει απλα τον πληκτρολογουμε.

Πλεον ο υπολογιστης ειναι συνδεδεμενος απομακρυσμενα με το λογαριασμο μας στο github.

Github

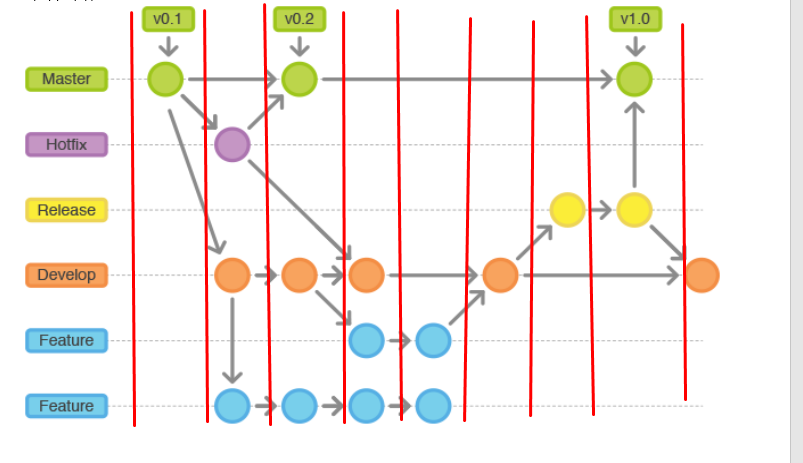
# TΡΟΠΟΣ ΛΕΙΤΟΥΡΓΙΑΣ ΚΑΙ ΦΙΛΟΣΟΦΙΑ



Τη λειτουργια του github μπορουμε να τη φανταστουμε ως ενα δεντρο (tree) με κλαδια (branches) και κυριο κορμο(master branch), πανω σε ενα χρονοδιαγραμμα.Για την ακριβεια ως ενα δεντρο με κερασια(Θα αναλυθει παρακατω).Παρακατω θα μελετησουμε ενα flow chart μιας τυχαιας εφαρμογης

Στην εικονα παρατηρουμε το εξης.

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

Το κυριο πλαισιο ειναι το master.Είναι το πρωτο branch που δημιουργειται και η κυρια ροη του προγραμματος.Τα υπολοιπα πλαισια ειναι τα δευτερευοντα branches.  
Οι διακεκομμενες γραμμες ειναι οι οριζοντιες γραμμες που μας δειχνουν την χρονικη εξελιξη των πραγματων απο αριστερα προς τα δεξια.   
Τα βελακια ροης δειχνουν τα βηματα που κανει η ροη του github το οποιο θα αναλυσουμε παρακατω.  
Tα κυκλακια αντιπροσωπευουν μια αλλαγη(ενα commit) στον κωδικα.  
Πολυ σημαντικο ειναι να παρατηρησουμε πως τα κυκλακια βρισκονται σε καθετη αντιστοιχια.  
  


Ας το αναλυσουμε αυτο.Στο πρωτο καθετο πλαισιο υπαρχει ΜΟΝΟ το κυκλακι του master branch.  
Αυτο σημαινει οτι ειναι το ΠΡΩΤΟ-initial release του source code μας στο github.Mπορουμε να φανταστουμε το master branch ως το «επισημο» source code που θα θελαμε καποιος να δει..Συνηθως ειναι η stable εκδοση του τρεχοντα κωδικα.

Στη συνεχεια παρατηρουμε 3 βελακια.  
1 προς το hotfix.1 προς το develop.1 προς το feature.  
  
Αυτο σημαινει οτι απο το master φτιαξαμε αλλα 3 branches.Βλεπουμε πως τα κυκλακια αυτα δεν ειναι στην ιδια καθετο με το master.αυτο σημαινει πως εχει γινει καποιο commit-καποια αλλαγη στον κωδικα.  
Συνεπως βλεπουμε οτι ο προγραμματιστης του κωδικα εφτιαξε 3 ακομα branch για να μπορει να προσθεσει μια αλλαγη.  
-Και γιατι δεν βαζει την αλλαγη κατευθειαν στο master branch?  
-Για να μπορει να την ελεγξει.Αν δουλευει σωστα τοτε την προσθετει στο master branch που θα δουμε παρακατω  
Στη τριτη καθετη γραμμη βλεπουμε το hotfix να μπαινει στο master.Αυτο σημαινει πως το hotfix ηταν πιθανον ενα bug fix ,οποτε ο προγραμματιστης το προσαρμοσε στο κυριo branch.Επισης ο προγραμματιστης ανεβαζει την εκδοση του προγραμματος σε v0.2

Στην ιδια τριτη καθετη γραμμη παρατηρω και τα αλλα κυκλακια απο develop και feature.

Καθε κυκλακι ειναι και ενα commit(μια αλλαγη) στον κωδικα.Αυτο σημαινει πως αν το κυκλακι που υπαρχει μεσα σε μια καθετη γραμμη ,υπαρχει και σε αλλο branch ,τοτε **το ΙΔΙΟ ακριβως** κομματι κωδικα υπαρχει και σε αυτο το branch.

Με την ιδια λογικη προχωραμε στο διαγραμμα και οπου υπαρχουν βελακια που πανε διαγωνια ,σημαινει πως απο εκεινο το σημειο(commit) ,φτιαχνω νεο branch.Θα τα αναλυσουμε ολα αυτα και παρακατω

**Παρατηρησεις**

* Το github κραταει ιστορικο των commits .Αν κατι γραφτει στο ιστορικο ,ΔΕΝ ΔΙΑΓΡΑΦΕΤΑΙ.
* Το ιστορικο του github μπορει να αναιρεθει (Να γινει reset) ;ή να γινει Revert
  + Revert σημαινει να γυρισω τον κωδικα πως ηταν πριν.  
    Αν πχ,εσβησα ενα declaration μιας μεταβλητης,τοτε με το revert τη ξαναδηλωνει
* Ολοκληρο το source code με τα ολα τα branches ονομαζεται repository (repo)
* Το οτι κανω push μια αλλαγη/ενα commit δεν σημαινει πως αλλαζω branch.Ο λογος που αλλαζω branch ειναι για να μπορω να κανω δοκιμες στον κωδικα χωρις να επιρρεαζω τον βασικο κωδικα που θεωρω stable
* Μεσα σε ενα repository μπορουμε να προσθεσουμε contributors και να εχουν δικαιωμα να αλλαξουν τον κωδικα.Οποτε την επομενη φορα που θα θελω να κανω καποιες αλλαγες,θα πρεπει να «τραβηξω» τις αλλαγες του αλλου contributor

Github

# Linux terminal και git εντολες

Το github αρχικα δημιουργηθηκε με μοναδικο τροπο χρησης ,τα linux terminals.Αργοτερα δημιουργηθηκε και η desktop εφαρμογη για Windows.Εδω θα αναλυθει η χρηση μεσω linux terminal.

Η εξοικειωση με το github μεσω terminal θα βοηθησει και αυτον που θα θελει να χρησιμοποιησει την windows desktop εφαρμογη.**Oχι ομως το αντιστροφο**(Η εφαρμογη για windows περιεχει επισης terminal ,για advanced χρηστες)

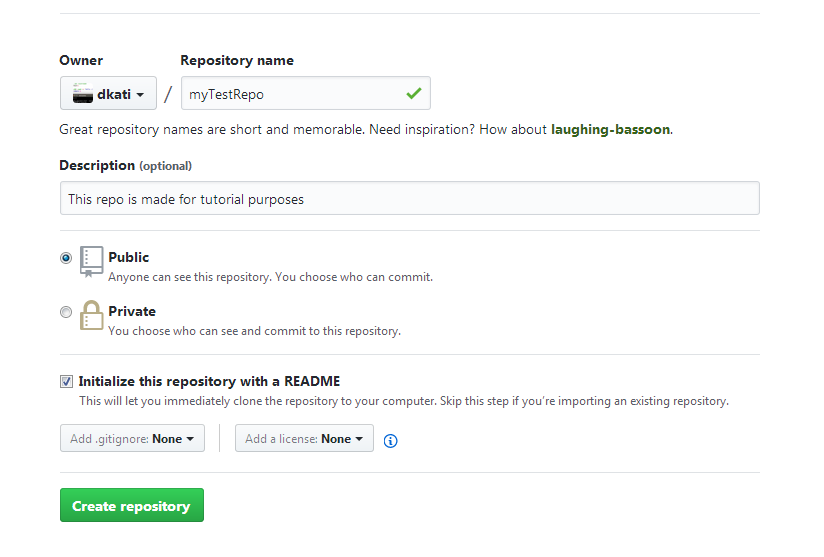
Οι τροποι που μπορω να ανεβασω/κατεβασω τον κωδικα μου στο/απο το github ειναι δυο.

* Ανεβασμα/Κατεβασμα τον κωδικα σε συγκεκριμενο branch ,στο github μου, και επεξεργασια.Το μειονεκτημα εδω ειναι πως ορισμενες φορες πρεπει να κατεβαζουμε ΟΛΟΚΛΗΡΟ τον κωδικα απο το github και οχι μονο τα τελευταια commits που καποιος αλλος contributor εκανε στο project
* Δημιουργια manifest που εχει κατεβασμενα ολα τα branch σε .tmp φακελο και ειναι ΠΑΝΤΑ προσβασιμα χωρις πιθανοτητα διαγραφης.Το μειονεκτημα εδω ειναι πως ο τροπος αυτος ειναι λιγο πιο περιπλοκος

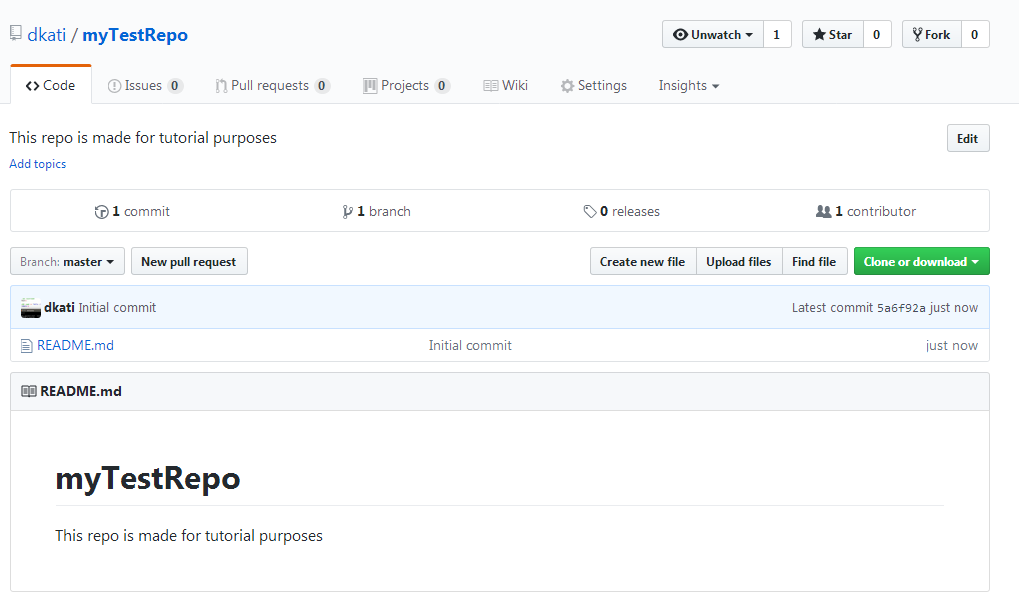
Παρακατω θα αναλυθει ο **δευτερος** τροπος τον οποιο θεωρουμε πιο ασφαλες.Παρολαυτα θα αναφερθουμε και στον πρωτο τροπο ο οποιος μας βοηθαει στο να παρουμε ολοκληρα repository απο αλλους

# Ξεκινωντας νεο repository για να ανεβασουμε τον κωδικα ενοσ project που εχουμε γραψει

Μεταφερομαστε στο προφιλ μας στο github.com.Αφου εχουμε ηδη κανει login , επιλεγουμε το **+** απο πανω 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.Αργοτερα θα προσθεσουμε και το license.

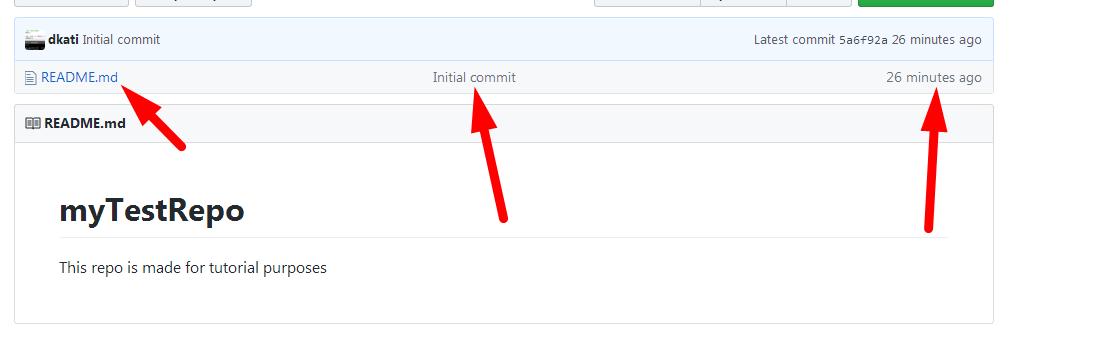


Αφου πατησουμε Create repository θα δουμε την εικονα αυτη



Παρατηρουμε τα εξης στοιχεια

* Πανω αριστερα βλεπουμε το ονομα του χρηστη και το ονομα του repository.  
  dkati/myTestRepo
* Απο κατω υπαρχει ενα μενου.
  + Code : Το κυριο και πιο σημαντικο μενου
  + Issues : Στη καρτελα αυτη μπορει οποιοσδηποτε χρηστης (ακομη και αυτος που δεν ειναι contributor) να κανει αναφορα καποιων θεματων-σφαλματων,και οι contributors να τα δουν και να απαντησουν/λυσουν τα ζητηματα
  + Pull requests : Στη καρτελα αυτη ειναι μαζεμενες καποιες προτασεις που καποιος τριτος χρηστης κανει,σχετικα με τον κωδικα.Οι contributors βλεπουν τις αλλαγες και αν θελουν με το πατημα ενος κουμπιου επιτρεπουν το τριτο χρηστη να προσθεσει τα commit του στο repository μας,χωρις να ειναι μελος αυτου
  + Wiki : Το παραδοσιακο wiki οπου οι contributors δινουν καποιες οδηγιες σχετικα με το source code
  + Settings : Ρυθμισεις του repository(Οχι του source code)
* Παρακατω ειναι το description που εχουμε βαλει στο repository
* Στη συνεχεια υπαρχει αλλο ενα μενου
  + 1 Commit : Πατωντας πανω στο μενου αυτο μας εμφανιζει ολα τα commits που εχουν γινει με χρονολογικη σειρα.Λεπτομερειες για τα commits θα αναφερθουν αργοτερα
  + 1 branch : εμφανιζει τα branches
  + 0 releases : αφορα τα release που κανουν οι contributors
  + 1 contributor : εμφανιζει ολους οσους συνεισφερει στον κωδικα ειτε ειναι μελη του repository ειτε χρησιμοποιησαμε τον κωδικα του
* Παρακατω βλεπουμε ενα κουμπι-μενου που λεει branch: master.Απο εδω μπορω να αλλαζω branches και να βλεπω τον αντιστοιχο κωδικα και τα αντιστοιχα commits.
* New pull request : Εαν θελω ως τριτος χρηστης να προσθεσω κωδικα
* Create new file/Upload files : Χειροκινητη δημιουργια/δημοσιευση αρχειου **(Δεν συνισταται)**
* Clone/Download : Κατεβασμα του source code σε zip μορφη (Δεν συνισταται!Ενα λογος που δεν συνισταται ειναι οτι στα linux περιβαλλοντα ,το zip αρχειο μπορει να διαγραψει τυχον symlinks κατα το extract
* **dkati** Initial commit : Αναφερεται το τελευταιο commit που εχει γινει.Το github αυτοματα κανει ενα commit οταν δημιουργουμε το repository και προσθετουμε readme.Η μορφη του τιτλου του commit ειναι   
  <github username> <Τιτλος commit>
* Ολα τα υπολοιπα απο κατω ειναι τα αρχεια του συγκεκριμενου directory μαζι με τις λεπτομερειες



Το αριστερο βελακι μας δειχνει τα αρχεια που εχει το συγκεκριμενο directory

Το μεσαιο βελακι δειχνει το τελευταιο commit που εχει γινει **και εχει επιρρεασει το συγκεκριμενο αρχειο**

Το δεξια βελακι δειχνει την ωρα που εχει περασει απο το τελευταιο commit που επιρρεασε το αντιστοιχο αρχειο.

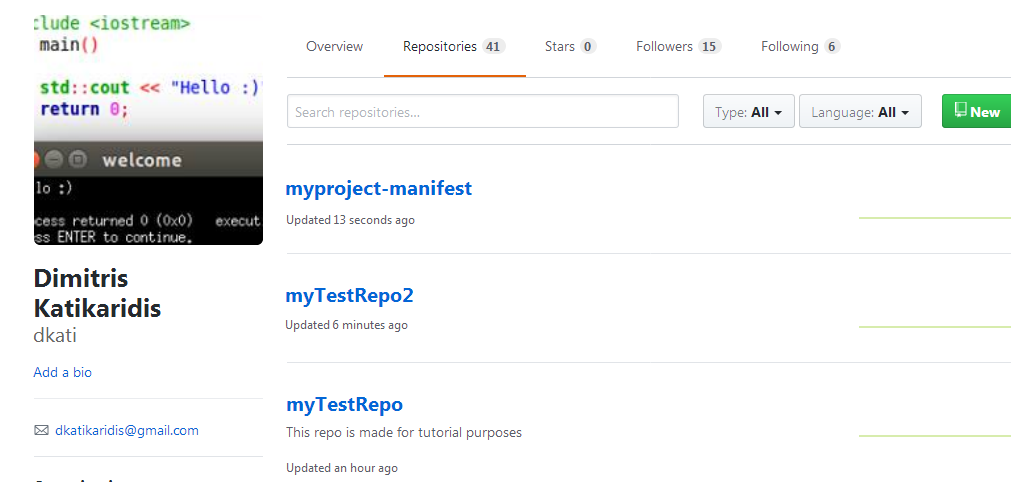
Γενικοτερα οι πληροφοριες αυτες μας βοηθουν να εχουμε μια ταξη μεσα σε τεραστιου μεγεθους κωδικες

Δεδομενου λοιπον οτι καταλαβαμε πως ειναι η δομη του repository παμε να φτιαξουμε **ΑΛΛΟ ΕΝΑ repository με ονομα myTestRepo2** το οποιο θα ειναι το 2ο εργαλειο μας και στη συνεχεια θα εξηγησουμε και θα φτιαξουμε το manifest που τα συνδεει ολα αυτα μαζι.

Manifest ειναι ενα ειδικο repository που περιεχει ενα ή και παραπανω αρχεια xml τα οποια περιεχουν τα github links απο τα projects που χρειαζομαστε για να στησουμε ενα ολοκληρο source code.  
Εχουμε παρατηρησει οτι πολλα source code απο επαγγελματικα προγραμματα ειναι χωρισμενα σε διαφορα μερη ,που καθε μερος ειναι ενα εργαλειο.Λογου χαρειν,στο ROS εχουμε διαφορα εργαλεια (Gazebo,ROS core,RVIZ) τα οποια θελουμε να δουλεουν ολα μαζι.  
Ετσι λοιπον εμεις θα φτιαξουμε εστω 2 repositories για καθε ενα εργαλειο που θελουμε να κανουμε τις δικες μας/custom αλλαγες.  
Η δουλεια του manifest ειναι να οριζει στο git service ,ποια sources πρεπει να κατεβουν.  
Αντι λοιπον να κατεβαζουμε ενα-ενα τα project μας ,τα τοποθετουμε σε ενα directory και το χωριζουμε σε κομματια μεσω του manifest για καλυτερη οργανωση.

# Δημιουργια ενοσ manifest

Δημιουργουμε ενα νεο δημοσιο repository με ονομα myproject-manifest ,με readme.  
Πριν προχωρησουμε ας δουμε λιγο την αρχικη οθονη του προφιλ μας.



Παρατηρουμε οτι πλεον εχουμε 3 repositories.Το ενα ειναι το myTestRepo που ενδεχομενως να ειναι το ενα απο τα εργαλεια που source code μου ,το δευτερο ειναι το myTestRepo2 το οποιο ειναι καποιο 2 εργαλειο μου και το αλλο ειναι το manifest

Παταμε πανω στο myproject-manifest και επιλεγουμε απο δεξια «Create new file»   
Στην επιλογη ονοματος πληκτρολογουμε «default.xml» και ως περιεχομενο βαζουμε

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<manifest>

<remote name=**"github"**

fetch=**"https://github.com/"** />

<project path=**"myTestRepoLocalDir"** name=**"dkati/myTestRepo"** remote=**"github"** revision =**"master"** />

<project path=**"myTestRepo2LocalDir"** name=**"dkati/myTestRepo2"** remote=**"github"** revision =**"master"** />

</manifest>

Ας εξηγησουμε τον κωδικα

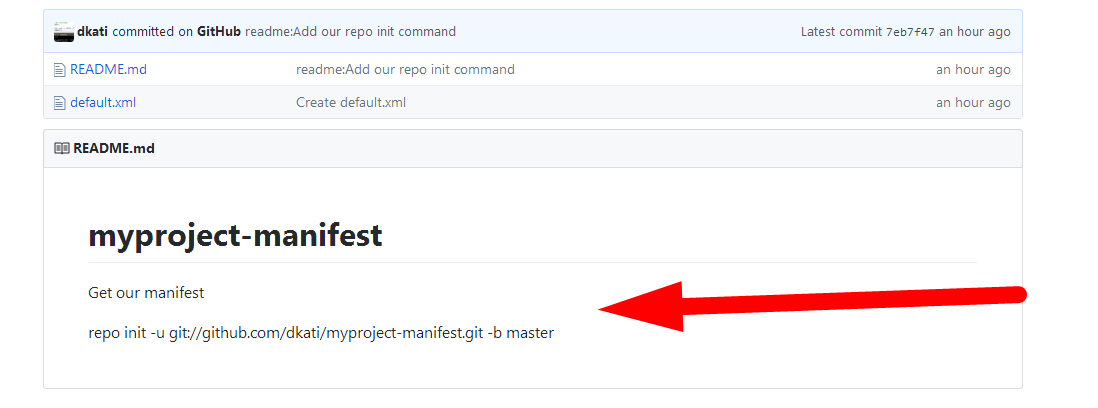
Η πρωτη σειρα αφορα την μορφη του XML.  
remote ειναι το μελος του xml που οριζει απο που θα τραβαει το git service ,ολα τα sources.  
το μελος path του project ,ειναι το τοπικο directory που θελουμε να παει το κατεβασμενο source code  
To name ειναι το github name του repository  
Revision ειναι το branch το οποιο θα εμφανιζεται στον τοπικο φακελο μας και το branch το οποιο θα κατεβαζουμε/κανουμε push.

Κατω απο την επικεφαλιδα Commit new file οριζουμε το Commit title.  
Εδω οριζουμε τον τιτλο του commit που θα φαινεται στα commits.Γενικοτερα πρεπει να προσεχουμε τα commit title που κανουμε καθως ειναι αυτα που καποιος τριτος βλεπει.Οποτε η εικονα που βγαινει προς τα εξω πρεπει να ειναι προσεγμενη.Ενα τυπικο προτυπο commit title ειναι το εξης:  
  
<dir></subdir> : <Τι εχω αλλαξει>  
Πχ αν εκανα αλλαγες μεσα σε μια class σε ενα αρχειο στο directory mysource/src/libs/mylib.cpp  
Τοτε στο commit title μπορω να γραψω -> src/lib: Do not expose \_var from mylib  
ή κατι παρομοιο.Γενικοτερα προσπαθουμε να κανουμε ευστοχα commit titles χωρις μεγαλη εκταση  
Στο description βαζουμε την περιγραφη του commit,αν θελουμε να εξηγησουμε κατι,και παταμε commit new file  
Mετα απο τη δημιουργια του αρχειου μπαινουμε στο αρχειο readme.md και επιλεγουμε το δεξια μολυβι ωστε να το επεξεργαστουμε.Μεσα στο αρχειο θα γραψουμε την κυρια εντολη που μας κατεβαζει το manifest στον υπολογιστη και το ρυθμιζει.Η εντολη ειναι η   
  
repo init -u git://github.com/dkati/myproject-manifest.git -b master

Στη συνεχεια πρεπει να συνταξουμε ενα commit title.  
Ενας αποδεκτος τιτλος θα μπορουσε να ειναι -> **readme:Add our repo init command**

Aφου κανουμε το commit επιστρεφουμε στο myproject-manifest

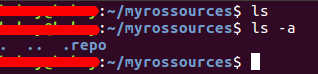
Παρατηρουμε οτι το επεξεργασμενο readme φαινεται μπροστα στο repository.**Αυτο ισχυει μονο για τα αρχεια readme** .Δεν σημαινει πως οποιαδηποτε αλλαγη σε αρχειο θα εμφανιζεται μπροστα.



# αρχικοποιηση του manifest και των source codes

Απο τη στιγμη που ετοιμασαμε τα repositories και το manifest ,μπορουμε πλεον να ξεκινησουμε τη διαδικασια της αποθηκευσης των repositories αυτων,στον υπολογιστη μας.Εκτελουμε   
**cd  
mkdir myrossources && cd myrossources**

Aν μας βγαλει μηνυμα σχετικα με τα χρωματα του λογαριασμου παταμε ‘Y’ και προχωραμε.Με μια πρωτη ματια μεσα στο φακελο δεν φαινεται να υπαρχουν αρχεια.Παρολαυτα με την εντολη   
ls -a παρατηρουμε οτι υπαρχει ενα νεος φακελος .repo

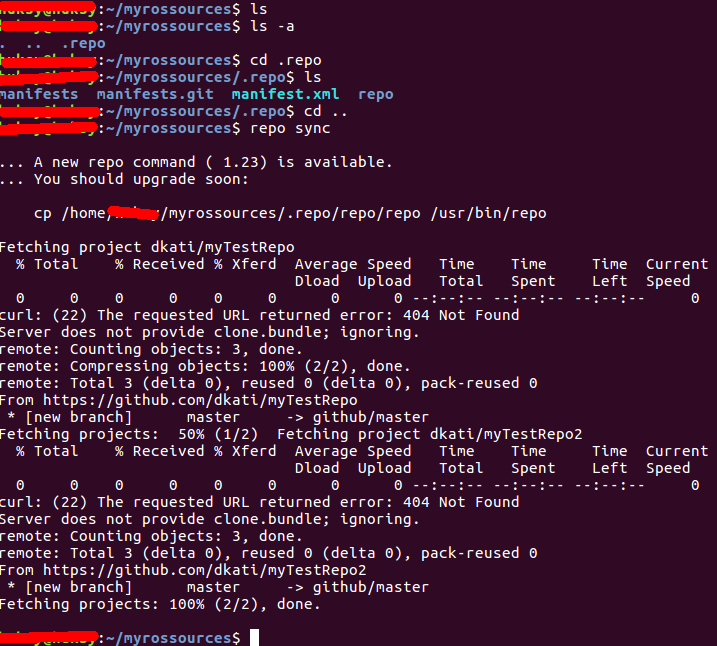


Στο φακελο αυτο υπαρχει το symlink manifest.xml το οποιο δειχνει στο **.repo/manifests/default.xml**  
το οποιο ειναι το manifest που γραψαμε πριν.

Ο φακελος repo εχει μεσα τα απαραιτητα αρχεια για να δουλεψει το git/repo service και δεν θα ασχοληθουμε με αυτα.  
  
Πηγαινουμε λοιπον ενα directory πισω και εκτελουμε

**repo sync**

**ΠΡΟΣΟΧΗ.  
Το repo sync εκτελειται παντοντε στο ιδιο directory με τον φακελο .repo  
Αν καταλαθως εκτελεστει η εντολη μεσα σε καποιον φακελο ,θα υπαρχει προβλημα σε ολο το source**



Μετα απο τη τελευταια μας εντολη , εχουμε κατεβασει ολο το source code στον υπολογιστη μας.

Μπορουμε επισης να κατεβασουμε μονο το 1 repository εαν θελουμε κανοντας

**repo sync myTestRepoLocalDir**

Αυτο θα κανει μια ανανεωση το τοπικο source code με αυτο του github.Αυτο βοηθαει οταν θελουμε να αναιρεσουμε τις αλλαγες μας.  
Παντοτε η λογικη ειναι να εχουμε updated τα source στο github ,ωστε να μπορουμε να κανουμε αλλαγες ,τοπικα και με ασφαλεια.Εαν λοιπον κατι παει στραβα,απλα σβηνω τον φακελο και εκτελω repo sync για το repository που θελω  
  
Το repo sync μπορει να παρει ως ορισμα τα threads με τα οποια θα κανει sync.

**repo sync -j4**

Γιατι ομως να μας ενδιαφερει το ποσα threads τρεχουν κατα το repo sync απο τη στιγμη που γινονται ολα μεσω ιντερνετ ?

Γιατι το repo ,κατεβαζει το source code σε συμπιεσμενη μορφη,αποσυμπιεζει τα αρχεια και μεσω μιας διαδικασιας diffcheck ελεγχει ποια αρχεια απο το source μας εχουν αλλαξει ωστε να τα αντικαταστησει.  
Αν εχουμε προβλημα με το μεγαλυτερο -j απλα το μειωνουμε.  
  
Σημειωση , το -j επιρρεαζει και τις εργασιες που κανει το δικτυο κατα το κατεβασμα.Πρακτικα ,μεγαλυτερο j σημαινει πιο γρηγορο κατεβασμα με μεγαλυτερη πιθανοτητα σφαλματος.Αυτο διαφοροποιειται απο συνδεση σε συνδεση διαδικτυου

**Μεχρι το σημειο αυτο,ολα πρεπει να γινουν μια και μονο φορα.Οτι ακολουθει ειναι αυτα που πρεπει να γνωριζουμε ωστε να εξοικιωθουμε με το github και τον τροπο λειτουργιας του.**

# Το πρωτο commit και το πρωτο push

Το πιο σημαντικο σημειο ειναι να καταλαβουμε τι θελουμε να κανουμε push και ποτε θελουμε να το κανουμε push.Οπως ειπαμε και πριν ενα commit πρεπει να ειναι καθαρο.Αυτο σημαινει οτι το καθε commit πρεπει

* Να εχει ξεκαθαρο τιτλο που θα περιγραφει τι ακριβως κανει
* Αν ειναι απαραιτητο να εχει μια περιγραφη με λεπτομεριες
* Να μην περιεχει αλλαγες σε αρχεια που δεν αφορουν την κυρια αλλαγη και σκοπο του commit.
* Να μην ειναι αντιγραφη απο αλλο commit.Αν θελω το commit καποιου τριτου θα το κανω με τον νομιμο τροπο που θα δουμε παρακατω

Ας εξηγησουμε την 3η περιπτωση.

Κατα την εξοικιωση μας με το github ,κανουμε push πραγματα τα οποια δεν πρεπει να γινουν push.

Εστω οτι θελουμε να κανουμε push μια αλλαγη σε ενα φακελο που περιεχει 3 αρχεια με σκοπο να γινει κατι συγκεκριμενο.Καλο θα ειναι λοιπον να μην αλλαξουμε κανενα αλλο αρχειο που δεν αφορα την κυρια επεξεργασια.

**Παραδειγμα**  
 *Σε ενα αρχειο αλλαζω 2 μεταβλητες απο int σε double  
και σε ενα αλλο αρχειο αλλαζω το ονομα μιας class.*

*Στο commit title βαζω “src:mylib Switch 2 vars from int to double”.Παρολαυτα υπαρχει και η αλλαγη του ονοματος της class και θα γινει και αυτο push.Κατι το οποιο δεν θα θελαμε.*

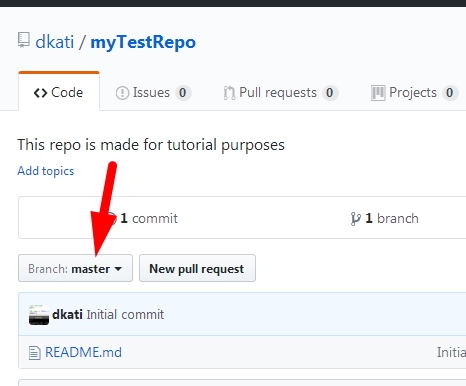
Πηγαινουμε λοιπον να κανουμε την πρωτη μας αλλαγη ,εστω στο myTestRepo  
Ο τοπικος φακελος του repository αυτου,ειναι το myTestRepoLocalDir.Οποτε

**cd myTestRepoLocalDir**

Αφου μπηκαμε στο directory πρεπει να δηλωσουμε το branch στο οποιο ειμαστε.Αρχικα να δουμε αν κατα τυχη ειμαστε ηδη σε branch.Αυτο μπορει να προκληθει απο καποια παλιοτερη μας διαδικασια μεσα στο φακελο.Εκτελουμε   
  
**git branch**

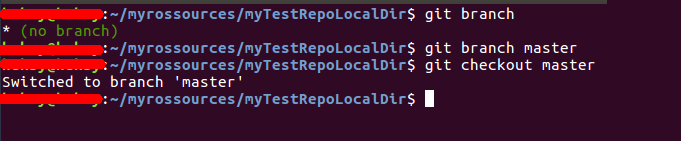
Και λογικα μας εμφανιζει (no branch)

Αυτο σημαινει οτι δεν ειμαστε σε κανενα branch.Βλεπουμε απο τη σελιδα του repo μας,οτι το branch μας λεγεται master.



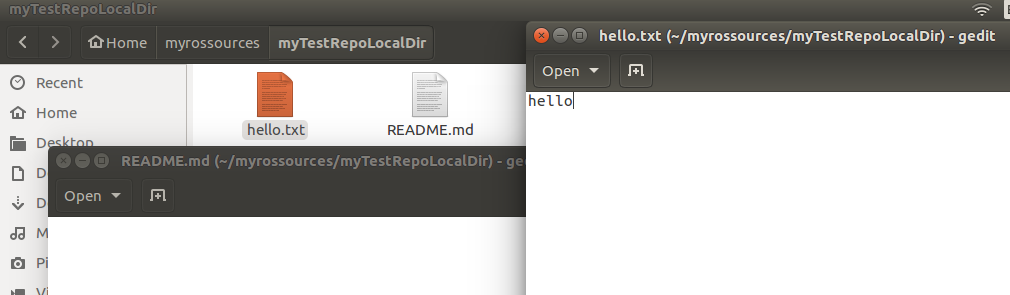
Aν θελω να στειλω τις μελλοντικες μου αλλαγες στο branch αυτο τοτε εκτελω

**git branch master  
git checkout master**

‘Η πιο απλα **git checkout -b master**

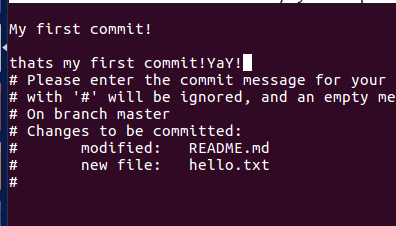
Απο τη στιγμη που αλλαξα το branch μου,μπορω ειτε να κρατησω το τερματικο ανοιχτο ειτε να το κλεισω.Δεν υπαρχει κατι που τρεχει στο background.

Κανω λοιπον τις αλλαγες μου τοπικα.  
Θα προσθεσουμε ενα αρχειο με ονομα hello.txt ,μεσα θα βαλουμε τη λεξη “Hello”  
και θα σβησουμε απο το readme οτι περιεχει



Εστω οτι αυτη ειναι η αλλαγη που θελαμε να κανουμε.Τωρα μενει να το προσθεσουμε στο commit.  
  
**git add -A  
git commit**

Αυτοματως θα μας εμφανισει ενα παραθυρο του nano οπου κανουμε επεξεργασια το commit μας.  
Η πρωτη σειρα ειναι παντα το commit title.Αφηνοντας μια σειρα και προσθετοντας μια τριτη,γραφουμε την περιγραφη.

  
Παρατηρουμε τις δυο σειρες  
**modified: README.md  
new file: hello.txt**

Αυτο μας επαληθευει οτι δεν «τραβηξαμε» στο commit καποιο αρχειο που δεν θα επρεπε να ειναι μεσα.  
Αν ειναι ολα σωστα,τοτε συνεχιζουμε.

*Διαφορετικα παταμε ctrl+x και μετα Y χωρις να εχουμε συμπληρωσει commit title και description για να βγουμε.Πηγαινουμε στο φακελο που ειναι το .repo (δηλαδη στο root του myrossources) και ειτε διαγραφουμε το φακελο και κανουμε παλι repo sync ειτε κανουμε κανονικα το push και μετα κανουμε νεο commit για το fix(που δεν συστηνεται).Μια αλλη τεχνικη ειναι να κρατησουμε τα αρχεια που καναμε επεξεργασια ,σε εναν αλλο φακελο,να κανουμε το repo sync και μετα paste τα αρχεια που ειναι επεξεργασμενα*

Παταμε ctrl+X και Y για να κλεισει το commit.Βλεπουμε στο τερματικο οτι δειχνει το τιτλο του commit και αναφερει οτι 2 αρχεια αλλαξαν,1 προσθηκη και 2 διαγραφες .Οι προσθηκες και οι διαγραφες αφορουν ΣΕΙΡΕΣ κωδικα.Επισης μας δειχνει οτι δημιουργηθηκε το hello.txt με mod 0644

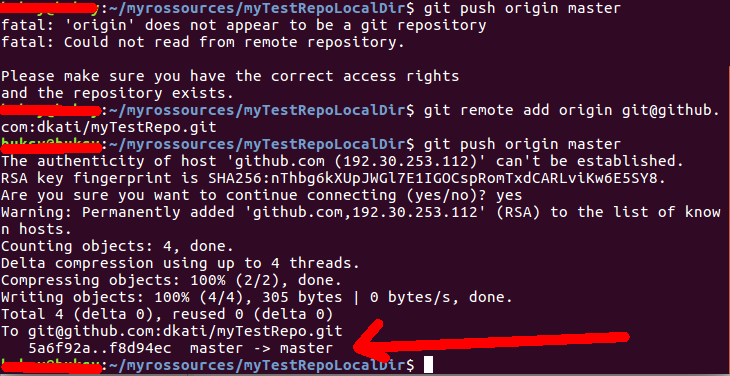
Απο τη στιγμη που εγινε το commit,πρεπει να το κανουμε push.

**git push origin master**

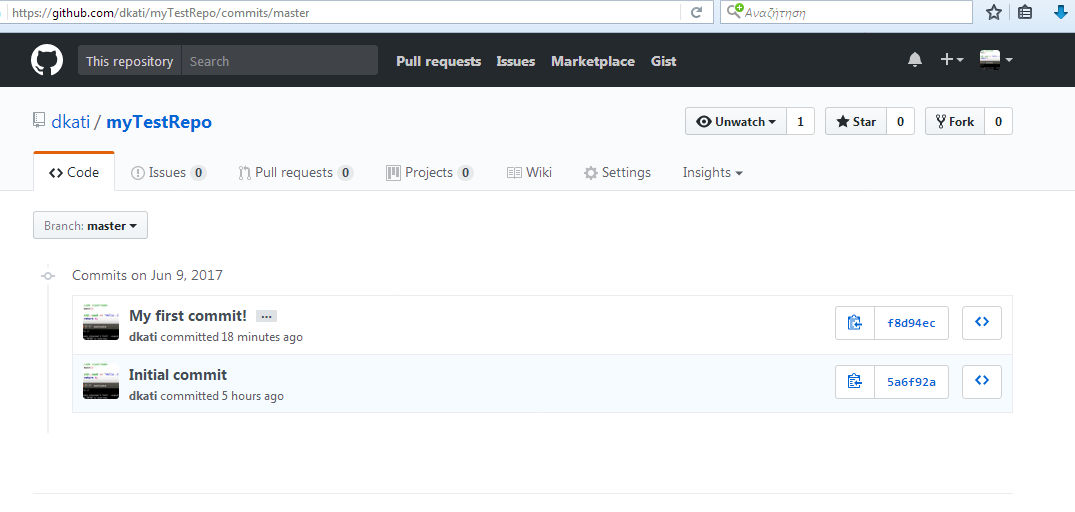
Η εντολη αυτη θα προσπαθησει να στειλει το commit στο branch master.  
Παρολαυτα μας βγαζει καποια fatal errors.Ο λογος ειναι οτι χρειαζεται να φτιαξουμε (απαιτειται μονο μια φορα) το remote.To remote ειναι υπευθυνο για την αποστολη του τοπικου commit ,στο προφιλ μας.Συνεπως εκτελουμε (μια φορα και μονο φορα)

**git remote add origin** [**git@github.com:dkati/myTestRepo.git**](mailto:git@github.com:dkati/myTestRepo.git)

Και ξανακανουμε **git push origin master**

Θα μας ρωτησει αν θελουμε πραγματι να γινει το push (Θα ρωτησει μονο για μια φορα και ποτε ξανα).παταμε yes και το στελνει.Αν βλεπουμε την παρακατω οθονη,τοτε ολα εγιναν σωστα

Στα commits λοιπον του project ,πλεον βλεπω την αλλαγη μου



Κανοντας κλικ πανω στο commit βλεπω τις λεπτομεριες.



Παρατηρουμε ολα τα στοιχεια του commit.Το τιτλο,τις αλλαγες μας και την περιγραφη.

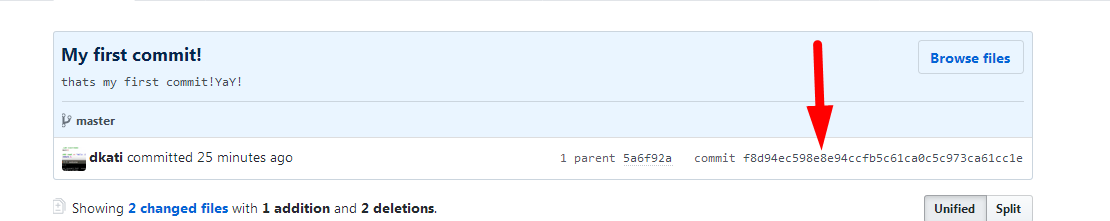
Οι κοκκινες γραμμες ειναι οτι σβηστηκε απο το αρχειο ενω οι πρασινες ειναι αυτες που εχουν προσθεθει.Αν ολες οι γραμμες απο ενα αρχειο ειναι πρασινες ,σημαινει πως το αρχειο εχει δημιουργηθει κατα το commit.

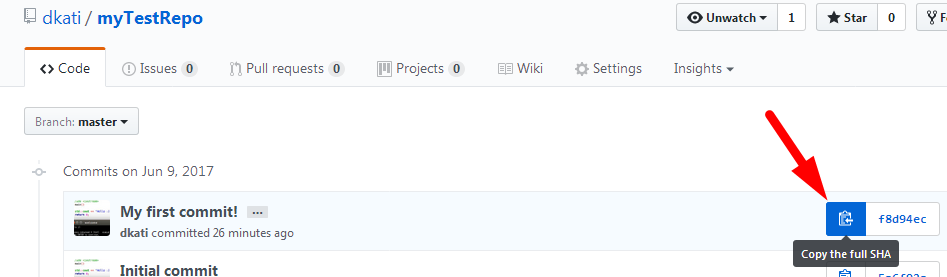
# αντιστροφη ενοσ commit

Εχουμε ηδη αναφερει πως στο github δεν μπορουμε να σβησουμε commits.Οτι κανουμε push παραμενει στο ιστορικο.Παρολαυτα μπορουμε να αντιστρεψουμε ενα commit και αυτο γινεται με μια μονο εντολη.

**git revert SHA\_CODE**

Το SHA ειναι το μοναδικο κλειδι που εχει καθε commit και το βρισκουμε μεσα απο το commit ή απο το ιστορικο.

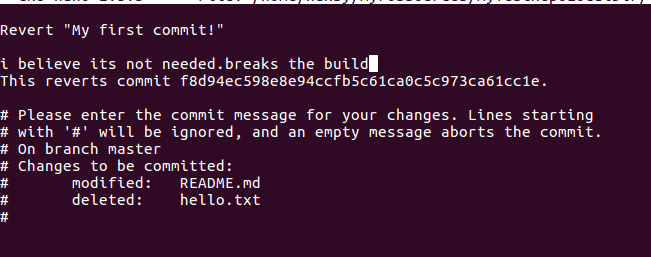


Οποτε ειτε το παιρνουμε με copy paste απο εκει ειτε με ενα απλο κλικ ,οπως φαινεται στην κατω φωτογραφια  


Αυτο θα μας αποθηκευση στο clipboard,το commit SHA

Αρα τρεχουμε **git revert 7fb76ff5c019adf04bf2369583fbcee346a1c4dd**

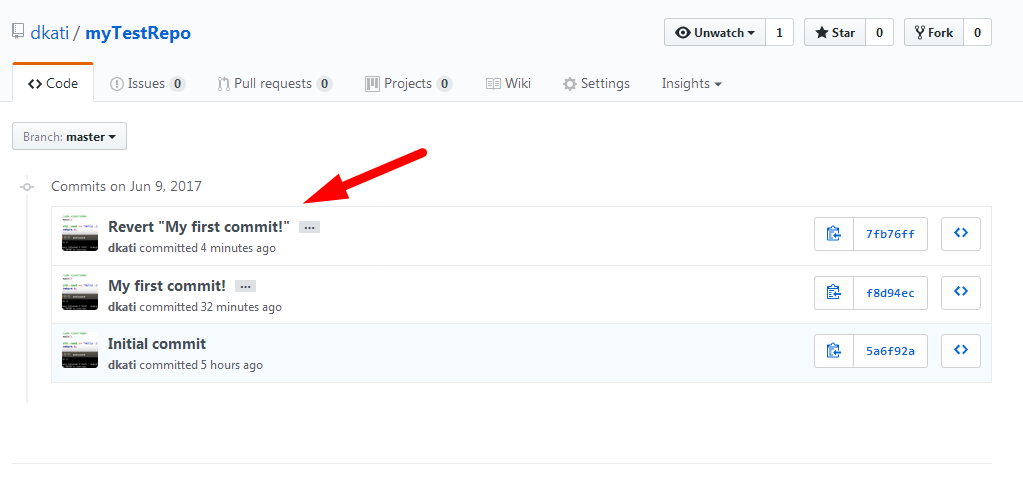
Οταν κανουμε το revert μας ανοιγει και παλι το commit message.  
Συνηθιζεται να αφηνουμε το commit title οπως ειναι,και να αλλαζουμε το description εξηγωντας γιατι το καναμε αντιστροφη.  
αφου κανουμε τις απαραιτητες αλλαγες βγαινουμε μαζι με ctrl+x και y



Και κανουμε το push

Git push origin master

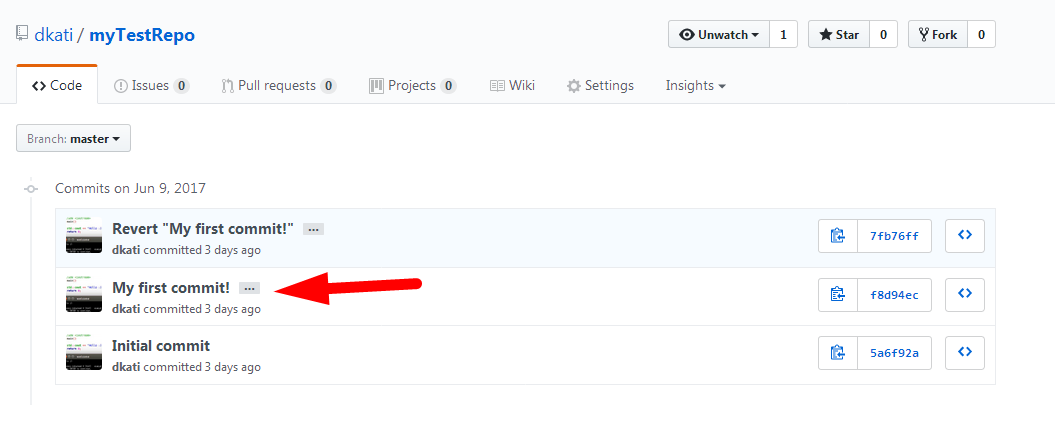
Αφου ολοκληρωθει το push ,μπορω να δω το commit στο commit history



# cherry-picking.Η «νομιμη» αντιγραφη κωδικα

Οποιοσδηποτε κωδικας στο github μπορει να ασφαλιστει με καποιο διεθνες license(apache,MIT κτλπ).  
Τα license προστατευουν τον συντακτη απο τυχον «κλοπες» του κωδικα.Το github εχει μεριμνησει για την εξασφαλιση της νομιμοτητας μεσω των λειτουργιων του,και δινει τη δυνατοτητα στο χρηστη να αντιγραψει ενα commit καποιου τριτου προγραμματιστη ,δινοντας τα απαραιτητα credits.  
Το cherry-pick μας βοηθαει στο να ανανεωνουμε κομματια κωδικα απο κωδικες αλλων προγραμματιστων,δινοντας παντα αυτοματως τα απαραιτητα credits.

Για να μπορεσουμε να κανουμε cherry-pick ,πρεπει πρωτα να κατεβασουμε ΟΛΟΚΛΗΡΟ του source code,μαζί με τα commits, απο το οποιο θελουμε καποια συγκεκριμενα commits.  
Εστω οτι θελουμε να κανουμε cherry-pick το commit που καναμε πιο πριν “my first commit” απο το   
<https://github.com/dkati/myTestRepo>

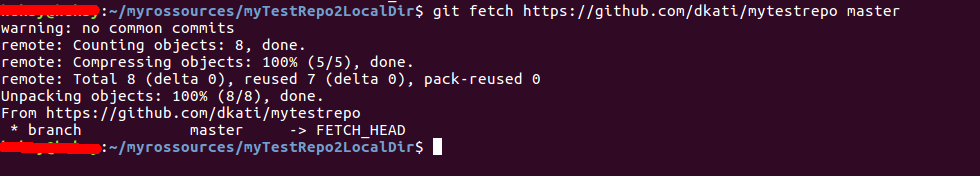


Αφου επιλεξαμε το commit που θελουμε πρεπει να παμε μεσω του τερματικου στο directory του project που θελω να βαλω το commit.Εστω οτι θελουμε να μπει στο myTestRepo2LocalDir.Eκτελουμε

**git fetch https://github.com/dkati/myTestRepo master**

Η συνταξη της εντολης ειναι   
git fetch <link του προτζεκτ που θελουμε> <ονομα branch>

Με την παραπανω εντολη κρατησαμε το ιστορικο των commits του συγκεκριμενου repository.



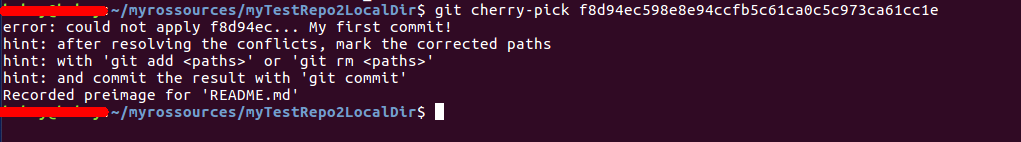
Πλεον ειμαστε ετοιμοι να «τραβηξουμε» το commit

**git cherry-pick f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e**

Η συνταξη της εντολης ειναι

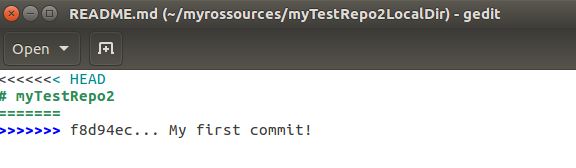
Git cherry-pick <SHA code>

Τρεχοντας την εντολη μας βγαζει error που αναφερει οτι δεν μπορει να προσαρμοσει το commit μεσα στον κωδικα μας.  
Οταν κανουμε cherry-pick ,κατι τετοιο ειναι πολυ συχνο και οφειλουμε να ειμαστε σε θεση να το διορθωσουμε.Το αποτελεσμα της εντολης ειναι το παρακατω



Κοιτωντας το commit παρατηρουμε οτι οι αλλαγες ειναι σε 2 αρχεια.



Παρολαυτα στο τερματικο μας βγαλε   
**Recorded preimage for README.md(βλ.παραρτημα\*\*\*)**. Αυτο σημαινει οτι ΜΟΝΟ σε αυτο το αρχειο υπαρχει προβλημα.  
Ανοιγοντας το αρχειο βλεπουμε το εξης

Ας αναλυσουμε τη συνταξη του αρχειου

**<<<<<<<<< HEAD  
# myTestRepo2  
=======**  
Αυτο σημαινει πως κανονικα το αρχειο μας περιεχει οτι βρισκεται αναμεσα στο  
<<<<<<<< και το ======  
  
Απο το ====== μεχρι το >>>>> ειναι αυτο που περιεχει η αλλαγη του commit.  
  
Στη συγκεκριμενη περιπτωση το περιεχομενο αναμεσα στο ===== και στο >>>> ειναι κενο.  
αυτο σημαινει οτι το commit που κανουμε cherry-pick σβηνει οτι περιεχεται απο το <<<<   
ως το ==== (το HEAD δηλαδη).Συνεπως η σωστη λυση εδω ειναι να σβησω τα παντα.

Αποθηκευω το αρχειο και αφου πλεον εχουμε κρατησει τις αλλαγες μας ξεκιναμε το git commit μας

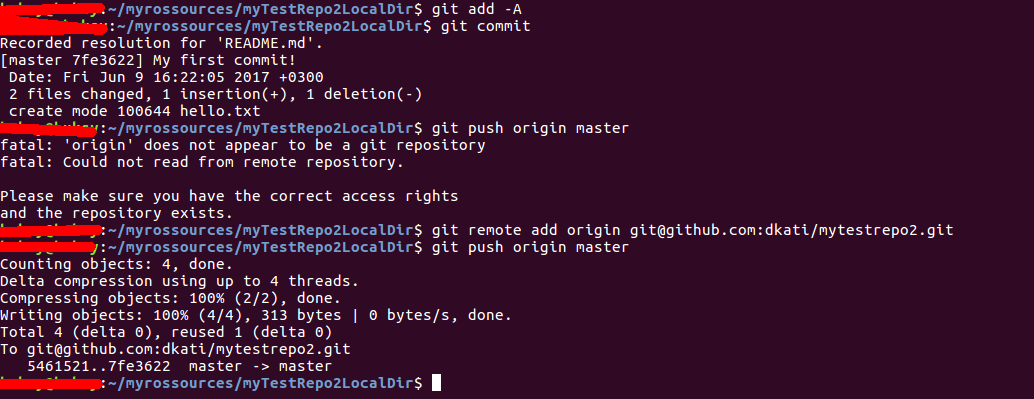
**git add -A   
git commit**  
  
και παρατηρουμε οτι εχει ηδη συμπληρωσει το commit title και το description.Παταμε ctrl+x για να βγουμε απο το nano.Πλεον μπορουμε να κανουμε push το commit.

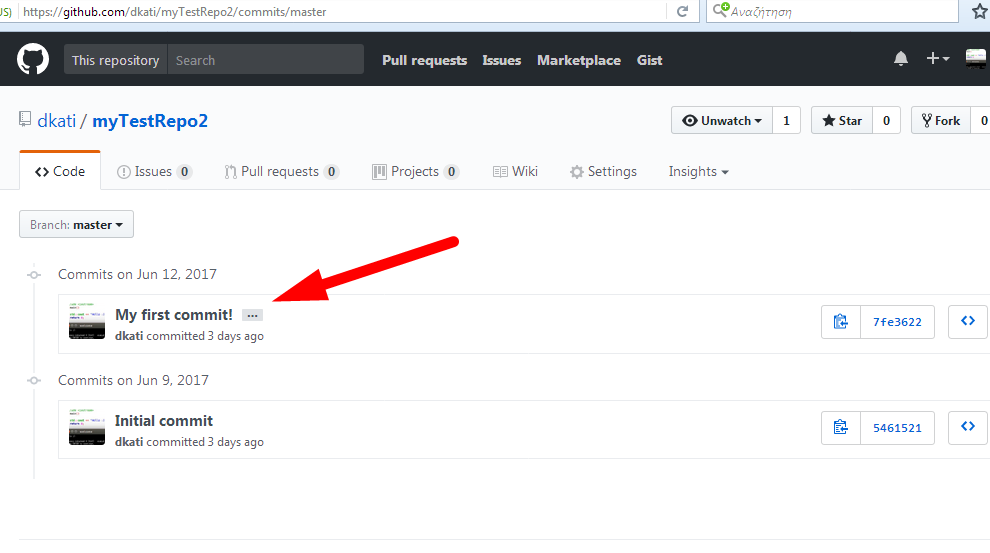
**git push origin master**

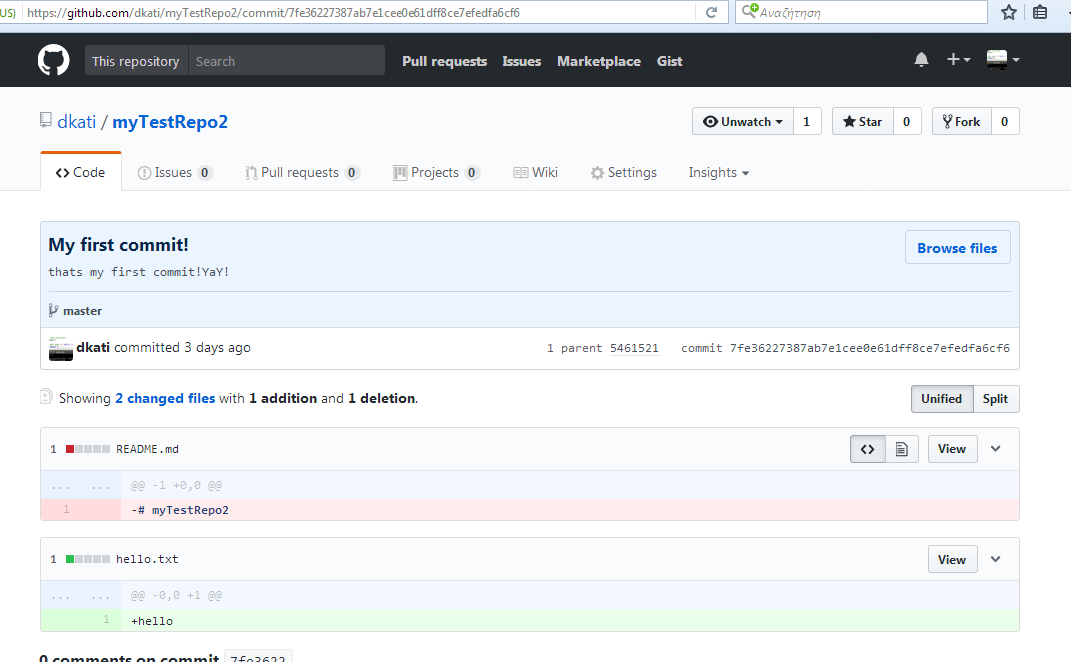
Στη περιπτωση μας θα μας βγαλει προβλημα στο remote.Οποτε οπως και πριν

**git remote add origin** [**git@github.com:dkati/myTestRepo2.git**](mailto:git@github.com:dkati/myTestRepo2.git)

Και ξανα git push origin master



Πραγματι παρατηρηρουμε οτι το commit εγινε επιτυχως   




Αυτο που παρατηρουμε στα 2 commit ειναι οτι αλλες αλλαγες εγιναν στο commit του myTestRepo και αλλες εγιναν στο commit του myTestRepo2.  
**Αυτος ειναι και ο λογος που ειχαμε conflict κατα το cherry-pick**.Ειναι σπανιες οι φορες που το cherrypick θα γινει χωρις conflict

Οταν ενα commit γινεται cherry-pick και ο συντακτης ειναι διαφορετικο ατομο απο αυτον που εκανε το cherry-pick τοτε το commit εμφανιζεται ετσι



Ο πραγματικος συντακτης ειναι ο **brinlyau** ενω αυτος που εκανε το cherry-pick ειναι ο **Dmole**

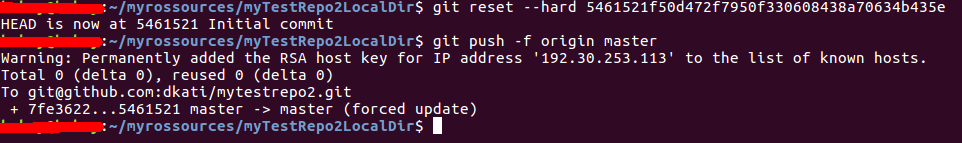
# Επανεγγραφη της ιστοριασ των commit (advanced users)

Οπως εχουμε αναφερει,δεν ειναι εφικτη η **ΔΙΑΓΡΑΦΗ** των commits.Παρολαυτα υπαρχει μια εντολη που επαναφερει **ΟΛΟΚΛΗΡΟ** το commit history σε μια παλαιοτερη εκδοση σβηνοντας τα ενδιαμεσα commits.Η εντολη αυτη ειναι λιγο επικινδυνη καθως διαγραφει οτι commit υπαρχει

*Πρακτικα δεν τα διαγραφει την υπαρξη τους απλα δεν φαινονται στο commit history.Επισης αναιρουνται απο τον τοπικο source code.Η διαδικασια αναιρεσης της διαγραφης ειναι δυσκολη και απαιτειται μεγαλη εξοικειωση με το github shell.*

Η συνταξη της εντολης ειναι

**git reset –hard SHA\_CODE** (Διπλές παύλες)  
Πχ Αν θελω να «σβησω» το τελευταιο commit του myTestRepo2 ,τοτε αρκει να κανω reset στο προτελευταιο.  
**git reset –hard 5461521f50d472f7950f330608438a70634b435e**  
και μετα οφειλω να το κανω push  
**git push -f origin master**  
Το -f σημαινει «force» .Το github το κανει αυτο για ασφαλεια.αν δοκιμασουμε να κανουμε git push origin master δεν θα μας αφησει.

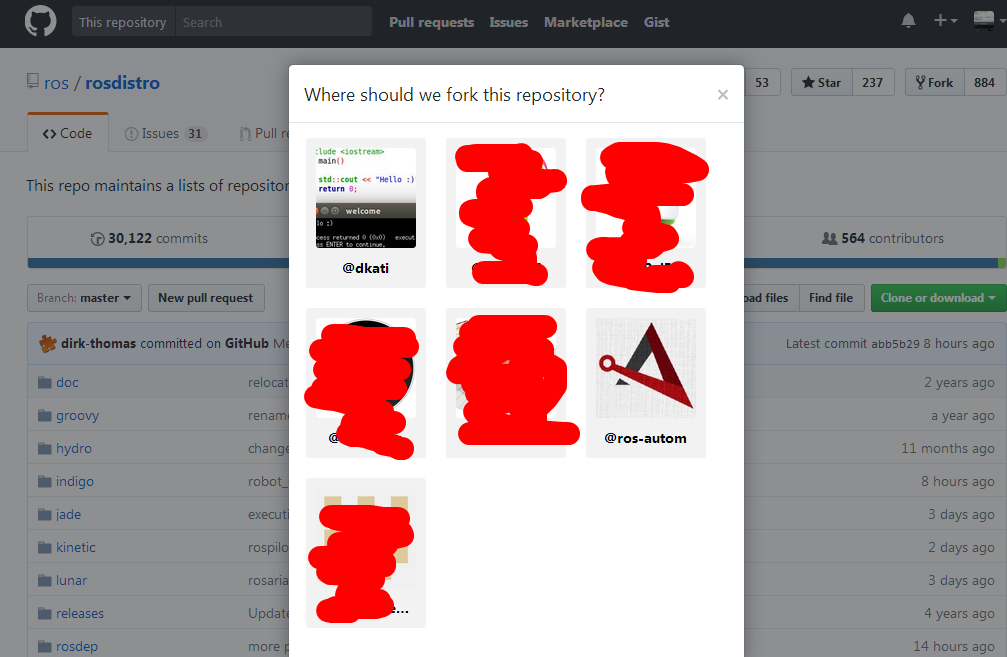


Αν το μετανιωσουμε μπορουμε να παρουμε το σβησμενο commit (ειναι το **7f3622**) κανοντας παλι reset  
  
**git reset –hard 7f3622**

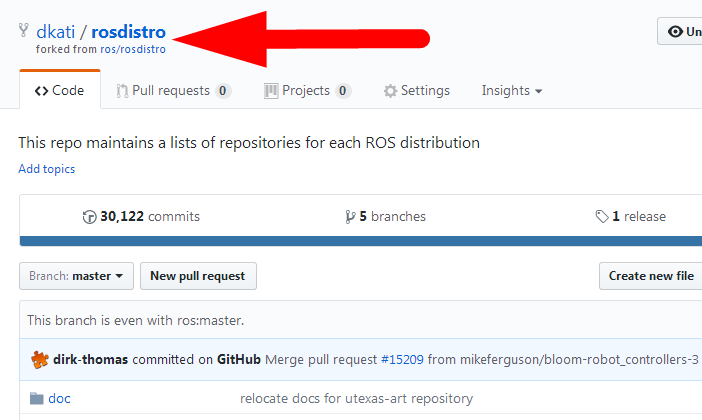
Συνεπως καταλαβαινουμε οτι το reset γραφει την ιστορια ειτε προς τα πισω ειτε προς τα εμπρος.

# Forking.Ο νομιμοσ τροπος να αντιγραψουμε ολοκληρα repositories

Forking ειναι η διαδικασια που αντιγραφουμε πληρως ολοκληρο το repository καποιου.Ειναι πολυ απλο και επιτυγχανεται με 2 απλα click.

Πηγαινουμε στο repository το οποιο θελουμε να κανουμε fork.Εστω το github.com/ros/rosdistro  
Παταμε πανω δεξια fork και επιλεγουμε σε ποιο λογαριασμο μας θελουμε να παει.  


Το repository εμφανιζεται στο προφιλ μας οπως παρακατω.  
φαινεται πολυ καθαρα οτι το repository ειναι εργο του ros/distro

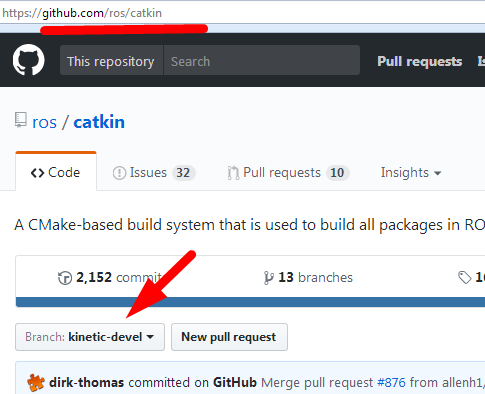


Απο εκει και περα χρησιμοποιω το project οπως ακριβως εργαστηκαμε με τα αλλα 2 repositories των παραδειγματων.

Παντοτε πρεπει να θυμομαστε οτι κατα το πρωτο push που θελουμε να κανουμε , χρειαζομαστε git remote add origin

# Γραψιμο ολοκληρης ιστοριασ απο αλλο repository

Εστω οτι εχω ενα repository με branch master και θελω να φτιαξω ενα ολοκληρο branch το οποιο θα ειναι πρακτικα,το branch ενος τριτου repository.  
Εστω λοιπον οτι θελω στο mytestrepo να φτιαξω ενα branch που θα περιεχει το kinetic-devel branch απο το ros/catkin repository



Προφανως δεν γινεται να κανω fork διοτι δεν θελω ΟΛΑ τα branch.Συνεπως πρεπει να δουλεψουμε ως εξης

* Τραβαω ολοκληρο το repository.
* Μπαινω στο kinetic-devel branch
* Φτιαχνω τοπικα ενα branch το οποιο προερχεται απο το kinetic-devel (βλ.την εικονα με τα βελακια και τα κυκλακια)
* Το κανω push

Συνεπως μπαινουμε στο myTestRepoLocalDir

**git clone** [**https://github.com/ros/catkin**](https://github.com/ros/catkin) **kinetic-devel  
cd kinetic-devel  
git branch**

Παρατηρουμε οτι ειμαστε στο branch που εχουμε κατεβασει.Απο το σημειο αυτο φτιαχνουμε το δικο μας branch.

**git branch mykinetic  
git branch**

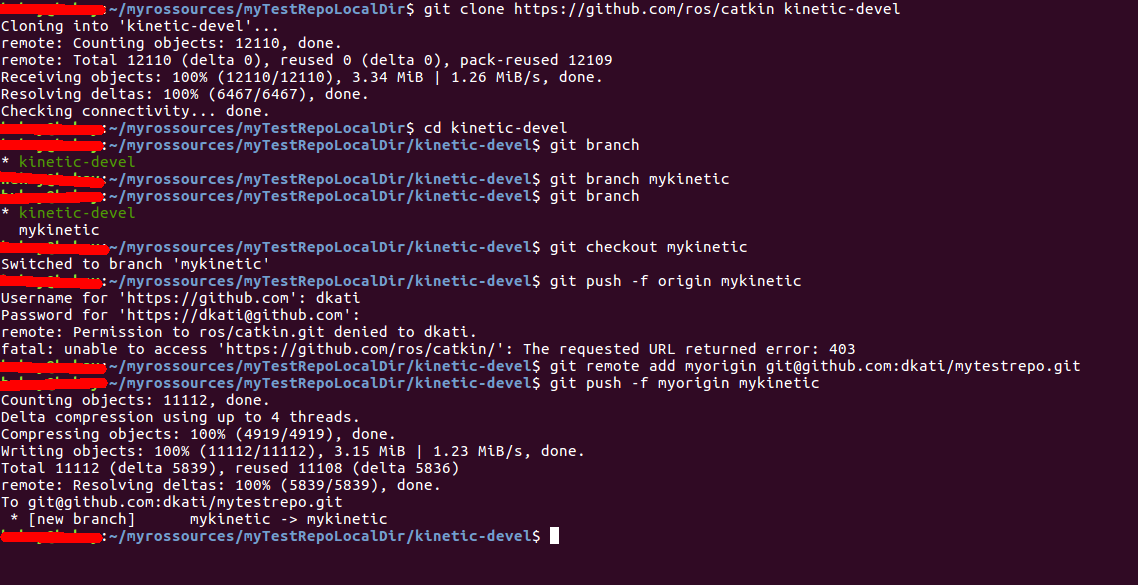
Πλεον βλεπω οτι εχω 2 branches και ξερω οτι το mykinetic εχει προερθει απο το kinetic-devel.  
Με πρασινο χρωμα ειναι το τοπικο branch στο οποιο βρισκομαστε.Αλλαζουμε λοιπον το branch

**git checkout mykinetic**

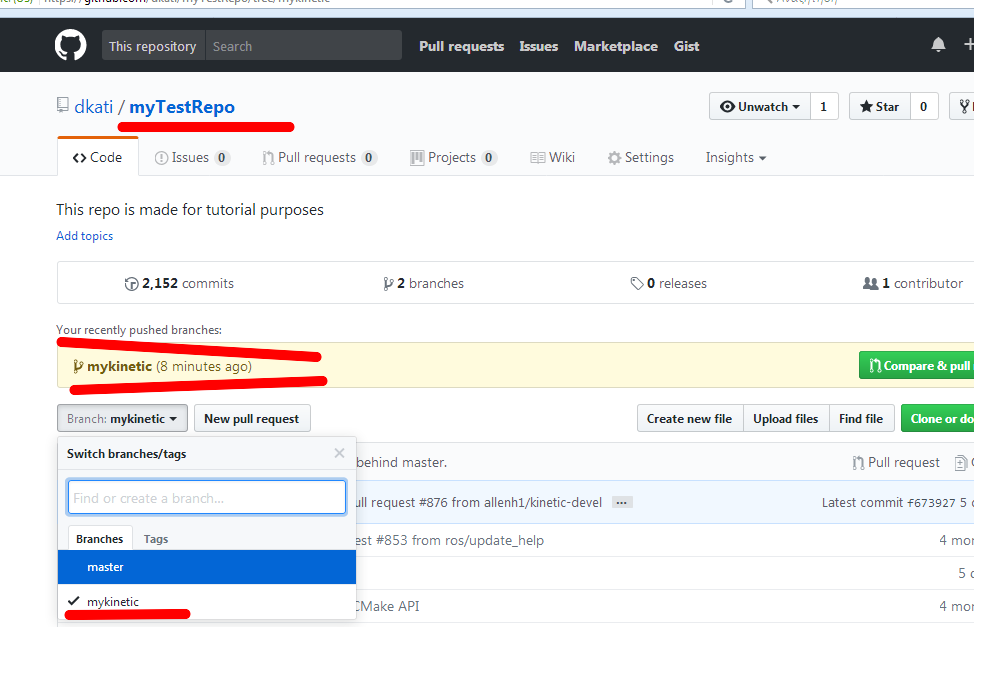
Αφου ειμαι στο branch ,κανω push οτι υπαρχει στο branch(και source code και commit history).Αν γραψουμε Git push origin mykinetic θα μας πει οτι δεν εχουμε δικαιωματα να κανουμε push το οποιο ειναι προφανες καθως δεν ανηκουμε στην ομαδα του ros.Αρα πρεπει να φτιαξουμε νεο origin

**git remote add myorigin** [**git@github.com:dkati/mytestrepo.git**](mailto:git@github.com:dkati/mytestrepo.git) **git push -f myorigin mykinetic**

Χρησιμοποιω -f γιατι αναγκαζω το github να μου φτιαξει στη σελιδα νεο branch



Εν τελει στο repository μας παρατηρουμε το νεο branch το οποιο ειναι ΑΚΡΙΒΩΣ ιδιο με το πραγματικο branch του ros

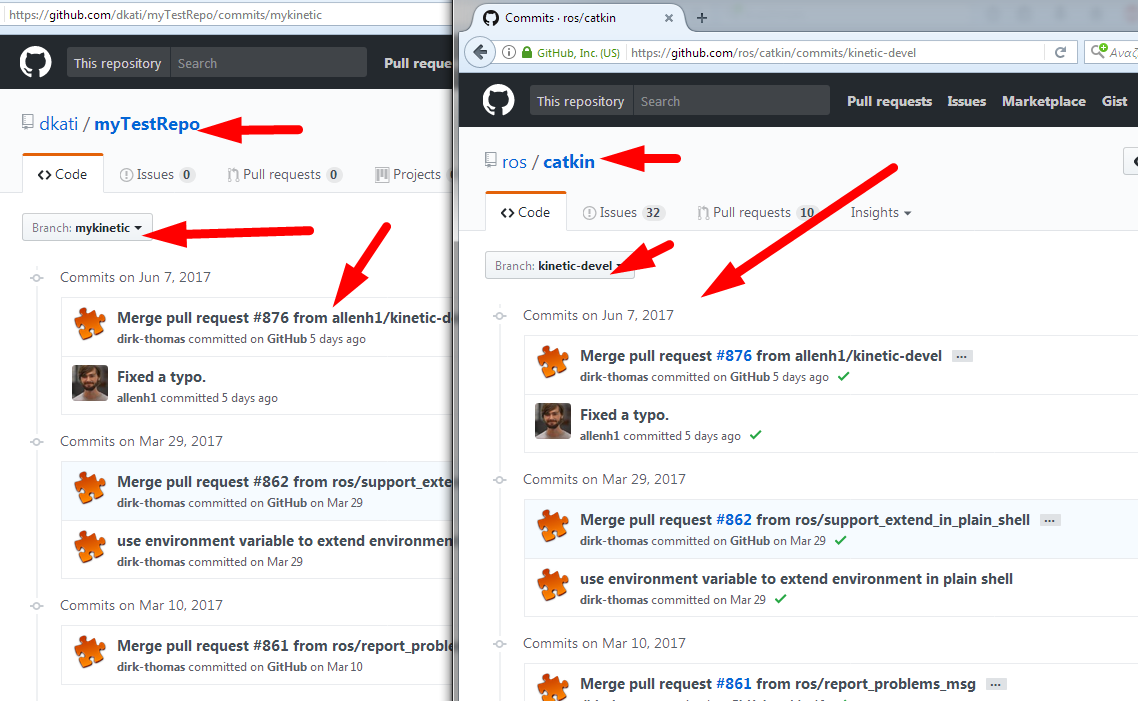


Αφου τελειωσαμε με το github,ας κανουμε ενα καθαρισμο τον κωδικα μας

**cd ../..  
rm -rf myTestRepoLocalDir  
repo sync myTestRepoLocalDir –force-sync**

--force-sync καλο ειναι να χρησιμοποιουμε σε καθε repo sync για να το αναγκασουμε να «τραβηξει» καθε νεα αλλαγη και να σβησει τυχον ξεχασμενα λαθη στο τοπικο source

Πολυ σημαντικο ειναι να παρατηρησουμε οτι ακομη και το commit history ειναι ΑΚΡΙΒΩΣ ιδιο με το αυθεντικο repository του ros



# Παραρτημα εντολων

* Git branch   
  Εμφανιση των διαθεσιμων τοπικων branches.Επισης εμφανιζει το branch στο οποιο ειμαστε τωρα(εμφανιζεται με πρασινα γραμματα).
* Git branch <branch name>  
  Δημιουργια ενος νεου branch ,απο το branch που ειμαστε ηδη
* Git checkout <branch name>  
  Μεταβαση σε αλλο branch
* Git checkout -b <branch name>  
  Δημιουργια ενος νεου branch και μεταβαση σε αυτο
* Git branch -D <branch name>  
  Διαγραφη ενος branch.ΔΕΝ ΜΠΟΡΟΥΜΕ ΝΑ ΔΙΑΓΡΑΨΟΥΜΕ ΤΟ BRANCH ΣΤΟ ΟΠΟΙΟ ΕΙΜΑΣΤΕ
* Git clone <github link απο repository> <branch name>  
  Κατεβασμα του branch απο το repository και αποθηκευση σε ενα νεο φακελο με ονομα το ονομα του branch.Ο φακελος βρισκεται στο ιδιο directory που ειμαστε με το terminal
* Git fetch <github link απο repository> <branch name>  
  Κατεβασμα του commit history του repository.ΠΡΕΠΕΙ ΝΑ ΤΡΕΞΕΙ ΜΕΣΑ ΣΤΟ DIRECTORY ΣΤΟ ΟΠΟΙΟ ΘΑ ΚΑΝΩ ΤΟ cherry-pick
* repo init -u git://<link απο manifest>.git -b master  
  Αρχικοποιηση του manifest   
  πχ repo init -u git://github.com/dkati/myproject-manifest.git -b master
* git cherry-pick <SHA>  
  Νομιμη αντιγραφη ενος commit με συγκεκριμενο SHA
* git reset –hard <SHA>  
  Επαναφορα του commit history στο συγκεκριμενο commit
* git revert <SHA>  
  Αναστροφη του συγκεκριμενου commit
* git commit   
  Δημιουργια commit
* git add -A  
  Προσθηκη των αλλαγων μου στο commit
* git push <remote name> <branch name>  
  Ανεβασμα του commit ,μεσω του remote,στο branch name
* git remote add <remote name> [git@github.com:<repo](mailto:git@github.com:%3crepo) name>.git  
  Δημιουργια remote
* git remote remove <remote name>  
  Σβησιμο remote
* Repo sync ή repo sync –force-sync  
  Συγχρονισμος τοπικου source code με αυτον στο github. –force-sync αν θελω να αναγκασω το github να σβησει τις τοπικες αλλαγες μου

**Ολες οι εντολες πρεπει να εκτελουνται μεσα στο root directory του καθε repository.Εκει δηλαδη οπου υπαρχει φακελος .git**

\*\*\*=  
Οταν το github μας αναφερει Recorded preimage σημαινει οτι εχει καταγραψει το λαθος ΚΑΙ ΤΗΝ ΕΠΙΛΥΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ του χρηστη .Αυτο το κανει για να λυσει αυτοματα το ιδιο προβλημα σε καποιο μελλοντικο cherry-pick .Πολλες φορες ομως δεν θελουμε να θυμαται το πως το λυσαμε γιατι ενδεχομενως η λυση να μην ειναι ιδια.Ετσι λοιπον χρησιμοποιω την εντολη

**git rerere forget \***

Με την εντολη αυτη λοιπον αναγκαζω το github να «ξεχασει» οποιαδηποτε πιθανη λυση ενος conflict.