

GitHub

Εγχειρίδιο χρήσης GitHub
μέσω Linux Terminals
ROS-autom 2017[®]

GITHUB

ΠΡΟΕΤΟΙΜΑΣΙΑ ΣΥΣΤΗΜΑΤΟΣ

Επαληθεύουμε ότι έχουμε τα απαραίτητα πακέτα

```
sudo apt-get install git  
sudo apt-get install repo
```

Το πιο πιθανό είναι να μην υπάρχει μόνο το repo και να απαιτεί εγκατάσταση. Στη συνέχεια πρέπει να συνδεθεί το github client του υπολογιστή μας, με το github στη σελίδα, ώστε να μπορώ στη συνέχεια να κάνω commits, pulls, pushes κλπ.

Τρέχουμε

```
git config --global user.email  
git config --global user.name mygithubname
```

(διπλές παύλες)

Το mail και ο κωδικός πρέπει να είναι σε πλήρη αντιστοιχία με αυτά που βλέπουμε στη σελίδα. Τα δικά μου είναι dkatkaridis@gmail.com με username dkat

Στη συνέχεια πρέπει να πιστοποιήσουμε στο github, τον υπολογιστή που θα κάνει push από το λογαριασμό μας.

```
ssh-keygen -t rsa -b 4096 -C
```

Και θα παρουμε ως επιστροφή

```
Generating public/private rsa key pair.  
Enter file in which to save the key  
πατάμε enter  
Enter passphrase  
πάλι enter  
Enter passphrase again  
Πάλι enter  
Τελικά θα μας δώσει το RSA 4096
```

```
The key's randomart image is:  
+---[RSA 4096]---+  
|  +oo+o  .=0+++ |  
|  ...oo.=.++.  |  
|  oo+*o. o  .  |  
|  o.+=.  o    |  
|  .+. oS.     |  
|  ....E+o.    |  
|  o  ...      |  
|  .  .  .     |  
|  ....o..     |  
+---[SHA256]---+
```

Στη συνέχεια πρέπει αυτο το sshkeyνα γραφτεί στον ssh-agent.

```
eval "$(ssh-agent -s)"
```

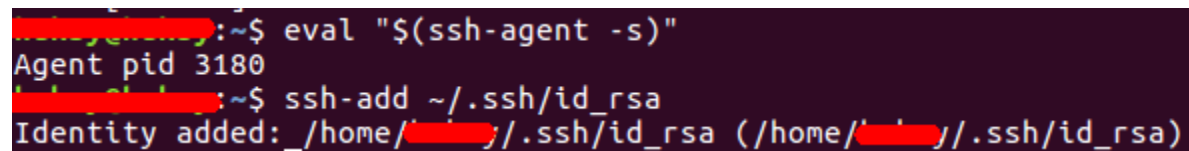
Θα παρουμε μια εξοδο σαν αυτη : **AgentpidXXXXX**

Αποθηκευουμε το sshσε ενα τοπικο αρχειο

```
ssh-add ~/.ssh/id_rsa
```

Με εξοδο

Identity added : /home/username/.ssh/id_rsa (home/username/.ssh/id_rsa)



```
username:~$ eval "$(ssh-agent -s)"
Agent pid 3180
username:~$ ssh-add ~/.ssh/id_rsa
Identity added: /home/username/.ssh/id_rsa (/home/username/.ssh/id_rsa)
```

Στη συνέχεια πρέπει να αντιγραφουμε το SSHστο clipboard.Ετσι κατεβαζουμε το εργαλειο xclip και αντιγραφουμε το SSHμεσω αυτου,στο clipboard

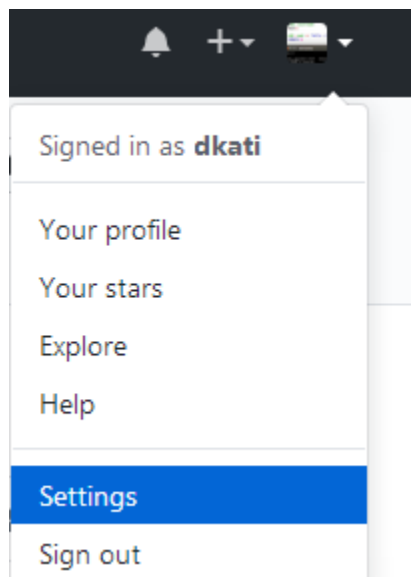
```
sudo apt-get install xclip && xclip -sel clip < ~/.ssh/id_rsa.pub
```

```

dkati@ubuntu:~$ sudo apt-get install xclip
[sudo] password for dkati:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  xclip
0 upgraded, 1 newly installed, 0 to remove and 3 not upgraded.
Need to get 17,0 kB of archives.
After this operation, 72,7 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu xenial/universe amd64 xclip amd64 0.12+svn84-4 [17,0 kB]
Fetched 17,0 kB in 0s (54,0 kB/s)
Selecting previously unselected package xclip.
(Reading database ... 346421 files and directories currently installed.)
Preparing to unpack .../xclip_0.12+svn84-4_amd64.deb ...
Unpacking xclip (0.12+svn84-4) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up xclip (0.12+svn84-4) ...
dkati@ubuntu:~$ xclip -sel clip < ~/.ssh/id_rsa.pub
dkati@ubuntu:~$

```

Στη συνέχεια αυτο το sshπρέπει να προσθεθει στο λογαριασμο μας στο github.Ανοιγουμε τις ρυθμισεις του λογαριασμου μας



Επιλεγουμε απο τα μενου , SSHandGPGkeysκαι παταμε στο δεξια κουμπι «NewSSHkey». Βαζουμε στο Titleεαν τιτλοκαι στο πλαίσιο keyαπλα παταμε δεξι κλικ/επικολληση

Personal settings

Profile

Account

Emails

Notifications

Billing

SSH and GPG keys

Security

Blocked users

Repositories

Organizations

Saved replies

Authorized OAuth Apps

SSH keys

2

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

SSH

Fingerprint:

Added on May 10, 2017 by

Last used within the last 2 weeks

Delete

SSH

Fingerprint:

Added on Jun 6, 2017

Last used within the last 3 days

Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

GPG keys

New GPG key

There are no GPG keys with access to your account.

Το τελικό αποτέλεσμα θα πρέπει να είναι έτσι

Title

Key

```
/PHFDF63YNq1ENBQNwezM5B2xMBPZnZueqI1GMFUkmDcqyRjWADCI0sOJ  
/uQd4UMsLZkeO3gSYpxQxRmEMtvJVy45tZnot+EXqB2  
/j6WnD3Mv3OyWoK53bj8hIH5A0Ny34qCffHEYsE65cDdaNP9nR6pxeAeKG2sH8QUul+E/efyOYT  
/rQEB7zludNyPvCXsogdMg7on7ARjTTMnd9vkXCQZ2QQRaEjYr0LtrZrf3AA6eJD+0fM4nj4j8KNehWziSA  
MKgJQV2fk437fGv9qK6+J6r1VyrhKHRmnDbNJHkgybE8chj3OCswGM6fQN3jeMYtrmKEGI3gDk2S4ztOW  
jp6Onn2kMeOqurt+y1PIVdCYefqw6lFRoVIEjmbP4AOhK  
/JRgmF0pxnQWUFsfzJWjKlegQWHDH0OFC8+s4UGgHiLVNsKIF2JstOtVvty7UQ==  
dkatkaridis@gmail.com  
|
```

Add SSH key

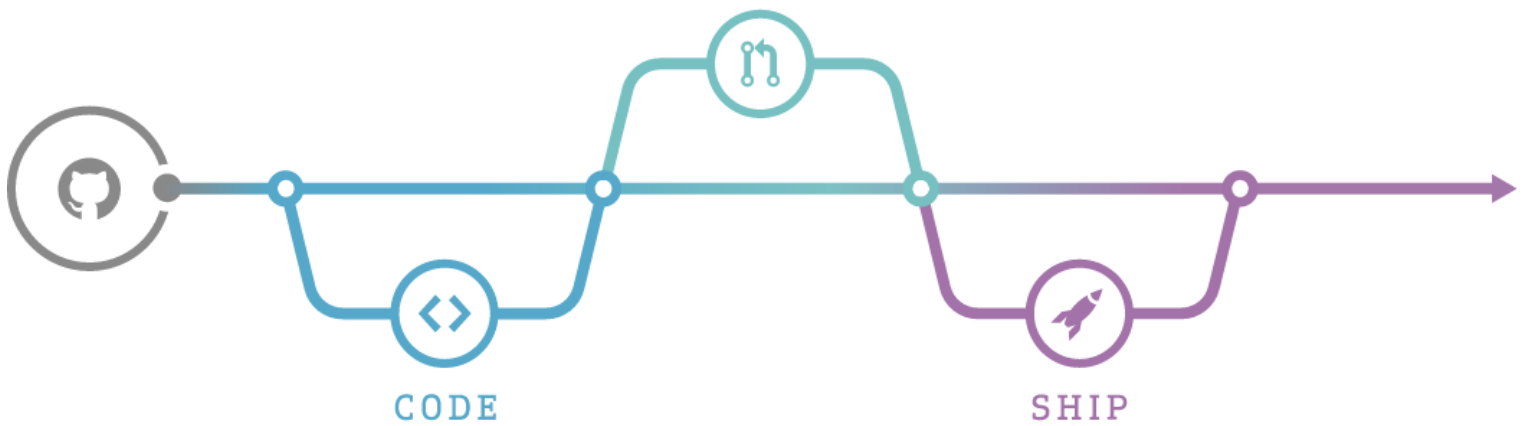
Και πατάμε AddSSHkey.

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

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

GITHUB

COLLABORATE



Τη λειτουργία του github μπορούμε να τη φανταστούμε ως ένα δέντρο (tree) με κλαδιά (branches) και κυριο κορμό(masterbranch), πάνω σε ένα χρονοδιαγράμμα. 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 όνομα master
- 1 πλαίσιο με όνομα hotfix
- 1 πλαίσιο release
- 1 πλαίσιο develop
- 2 πλαίσια feature
- Κάποια κυκλακία
- 6 διακεκομμένες γραμμές
- Βελακία ροής
- 3 κουτάκια που αναφέρουν την έκδοση της εφαρμογής κάθε φορά

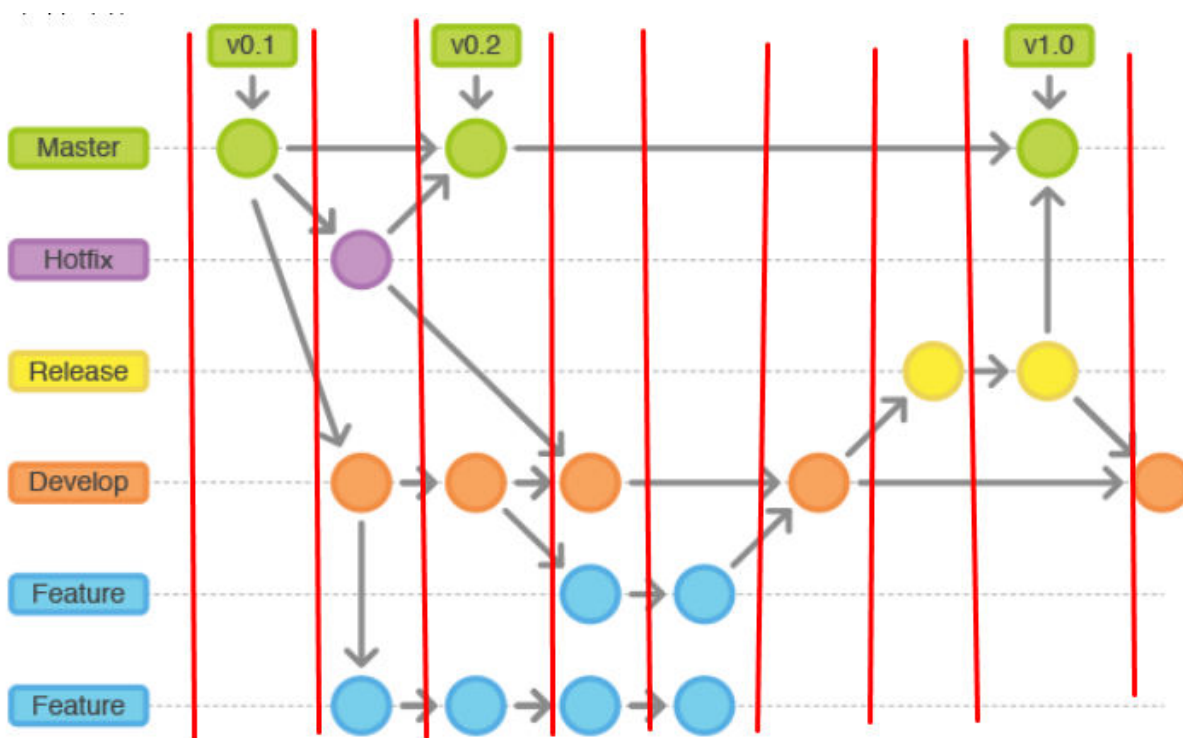
Το κύριο πλαίσιο είναι το master. Είναι το πρώτο branch που δημιουργείται και η κύρια ροή του προγράμματος. Τα υπόλοιπα πλαίσια είναι τα δευτερεύοντα branches.

Οι διακεκομμένες γραμμές είναι οι οριζόντιες γραμμές που μας δείχνουν την χρονική εξέλιξη των πραγμάτων από αριστερά προς τα δεξιά.

Τα βελακία ροής δείχνουν τα βήματα που κάνει η ροή του github το οποίο θα αναλύσουμε παρακάτω.

Τα κυκλακία αντιπροσωπεύουν μια αλλαγή (ένα commit) στον κώδικα.

Πολύ σημαντικό είναι να παρατηρήσουμε πως τα κυκλακία βρίσκονται σε κάθετη αντιστοιχία.



Ας το αναλύσουμε αυτο.Στο πρωτο καθετο πλαισιο υπαρχει ΜΟΝΟ το κυκλακι του masterbranch. Αυτο σημαινει οτι ειναι το ΠΡΩΤΟ-initialreleasetου sourcecodeμας στο github.Μπορουμε να φανταστουμε το masterbranchως το «επισημο» sourcecodeπου θα θελαμε καποιος να δει..Συνηθως ειναι η stableεκδοση 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 masterφτιαξαμε 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 branchγια να μπορει να προσθεσει μια αλλαγη.

-Και γιατι δεν βαζει την αλλαγη κατευθειαν στο masterbranch?

-Για να μπορει να την ελεγξει.Αν δουλευει σωστα τοτε την προσθετει στο masterbranchπου θα δουμε παρακατω

Στη τριτη καθετη γραμμη βλεπουμε το hotfixνα μπαινει στο master.Αυτο σημαινει πως το hotfixηταν πιθανον ενα bugfix ,οποτε ο προγραμματιστης το προσαρμοσε στο κυριο branch.Επισης ο προγραμματιστης ανεβαζει την εκδοση του προγραμματος σε νο.2

Στην ιδια τριτη καθετη γραμμη παρατηρω και τα αλλα κυκλακια απο developκαι 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 githubκραται ιστορικο των commits .Αν κατι γραφτει στο ιστορικο ,ΔΕΝ ΔΙΑΓΡΑΦΕΤΑΙ.
- Το ιστορικο του githubμπορει να αναιρεθει (Να γινει reset) ;ή να γινει Revert
 - Revertσημαινει να γυρισω τον κωδικα πως ηταν πριν.
Αν πχ,εσβησα ενα declarationμιας μεταβλητης,τοτε με το revertτη ξαναδηλωνει
- Ολοκληρο το sourcecodeμε τα ολα τα branchesονομαζεται repository (repo)
- Το οτι κανω pushμια αλλαγη/ενα commitδεν σημαινει πως αλλαζω branch.Ο λογος που αλλαζω branchειναι για να μπορω να κανω δοκιμες στον κωδικα χωρις να επιρρεαζω τον βασικο κωδικα που θεωρω stable
- Μεσα σε ενα repositoryμπορουμε να προσθεσουμε contributorsκαι 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 αρχικά δημιουργήθηκε με μοναδικό τρόπο χρήσης, τα linux terminals. Αργότερα δημιουργήθηκε και η desktop εφαρμογή για Windows. Εδώ θα αναλυθεί η χρήση μέσω linux terminal. Η εξοικείωση με το github μέσω terminal θα βοηθήσει και αυτόν που θα θέλει να χρησιμοποιήσει την windows desktop εφαρμογή. **Οχι όμως το αντιστρόφω** (Η εφαρμογή για windows περιέχει επίσης terminal, για advanced χρήστες)

Οι τρόποι που μπορώ να ανεβάσω/κατεβάσω τον κώδικά μου στο/από το github είναι δύο.

- Ανεβάσμα/Κατεβάσμα τον κώδικα σε συγκεκριμένο branch, στο github μου, και επεξεργασία. Το μειονέκτημα εδώ είναι πως ορισμένες φορές πρέπει να κατεβαζουμε ΟΛΟΚΛΗΡΟ τον κώδικα από το github και όχι μόνο τα τελευταία commits που κάποιος άλλος contributor έκανε στο project
- Δημιουργία manifest που έχει κατεβασμένα όλα τα branch σε .tar φακέλο και είναι ΠΑΝΤΑ προσβάσιμα χωρίς πιθανότητα διαγραφής. Το μειονέκτημα εδώ είναι πως ο τρόπος αυτός είναι λίγο πιο περιπλοκός



Παρακάτω θα αναλυθεί ο **δευτερος** τρόπος τον οποίο θεωρούμε πιο ασφαλές. Παρόλα αυτά θα αναφερθούμε και στον πρώτο τρόπο ο οποίος μας βοηθάει στο να παρούμε ολοκληρωτά repository από άλλους

ΞΕΚΙΝΩΝΤΑΣ ΝΕΟ REPOSITORY ΓΙΑ ΝΑ ΑΝΕΒΑΣΟΥΜΕ ΤΟΝ ΚΩΔΙΚΑ ΕΝΟΣ PROJECT ΠΟΥ ΕΧΟΥΜΕ ΓΡΑΨΕΙ

Μεταφερόμαστε στο προφίλ μας στο github.com. Αφού έχουμε ήδη κάνει login, επιλέγουμε το **+** από πάνω δεξιά και επιλέγουμε New repository. Εκεί μπορούμε να συμπληρώσουμε τα στοιχεία του αρχικού repository. Αρχικά θα είναι αδειο και μετά θα κάνουμε προσθήκη του κώδικα μας ως initial release. Γράφουμε το όνομα του repository, μια περιγραφή εάν θέλουμε, Δημοσίο repository και επιλέγουμε και το Initialize this repository with a README για να δημιουργήσει το πρώτο μας αρχείο. Αργότερα θα προσθέσουμε και το license.



Owner

Repository name

 **dkati** ▾ / 

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

Description (optional)

-
- ☒  **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.

| 

Create repository

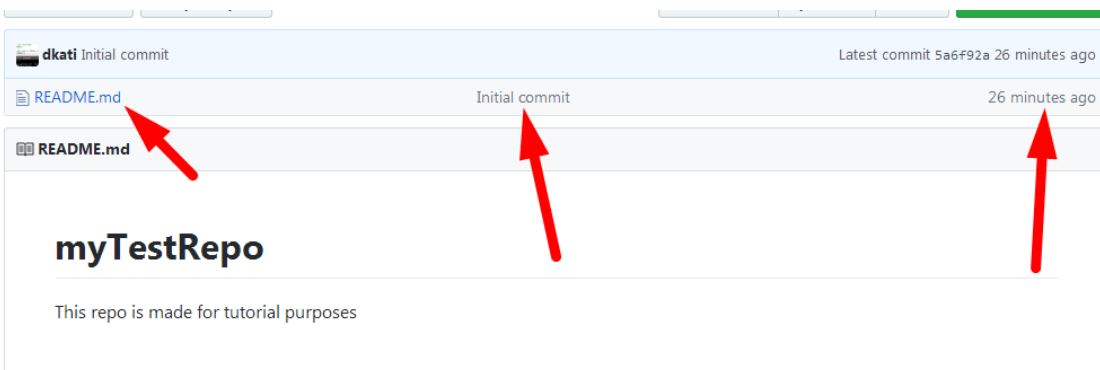
Αφου πατησουμε Createrepositoryθα δουμε την εικονα αυτη

The screenshot shows the GitHub interface for a repository named 'myTestRepo' by user 'dkati'. At the top, there are buttons for 'Unwatch' (1), 'Star' (0), and 'Fork' (0). Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Settings', and 'Insights'. A message states 'This repo is made for tutorial purposes' with an 'Edit' button. Below this, statistics show '1 commit', '1 branch', '0 releases', and '1 contributor'. Action buttons include 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The file list shows 'README.md' as the 'Initial commit' made 'just now'. The 'README.md' content area displays the repository name 'myTestRepo' and the same tutorial purpose message.

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 contributorsνα τα δουν και να απαντησουν/λυσουν τα ζητηματα
 - Pullrequests : Στη καρτελα αυτη ειναι μαζεμενες καποιες προτασεις που καποιος τριτος χρηστης κανει,σχετικα με τον κωδικα.Οι contributorsβλεπουν τις αλλαγες και αν θελουν με το πατημα ενος κουμπιου επιτρεπουν το τριτο χρηστη να προσθεσει τα committου στο repositoryμας,χωρις να ειναι μελος αυτου
 - Wiki : Το παραδοσιακο wikiπου οι contributorsδινουν καποιες οδηγιες σχετικα με το sourcecode
 - Settings : Ρυθμισειςτουrepository(Οχι τουsource 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 releases 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 αρχιου **(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 διαγραφει 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
<githubusername><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 μεσαio 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.

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

Δεδομένου λοιπόν ότι καταλαβαίμε πως είναι η δομή του repository παμε να φτιάξουμε **ΑΛΛΟ ΕΝΑ repository με όνομα myTestRepo2** το οποίο θα είναι το 2^ο εργαλείο μας και στη συνέχεια θα εξηγήσουμε και θα φτιάξουμε το manifest που τα συνδέει όλα αυτά μαζί.

Manifest είναι ένα ειδικό repository που περιέχει ένα ή και παραπάνω αρχεία xml τα οποία περιέχουν τα github links από τα projects που χρειαζόμαστε για να στησουμε ένα ολοκληρωμένο source code. Έχουμε παρατηρήσει ότι πολλά source code από επαγγελματικά προγράμματα είναι χωρισμένα σε διάφορα μέρη, που κάθε μέρος είναι ένα εργαλείο. Λόγου χάριν, στο ROS έχουμε διάφορα εργαλεία (Gazebo, ROScore, RVIZ) τα οποία θέλουμε να δουλέουν όλα μαζί.

Ετσι λοιπόν εμείς θα φτιάξουμε εστω 2 repositories για κάθε ένα εργαλείο που θέλουμε να κάνουμε τις δικές μας/custom αλλαγές.

Η δουλειά του manifest είναι να ορίζει στο git service, ποια source code πρέπει να κατεβούν.

Αντί λοιπόν να κατεβαίνουμε ένα-ένα τα project μας, τα τοποθετούμε σε ένα directory και το χωρίζουμε σε κομμάτια μέσω του manifest για καλύτερη οργάνωση.

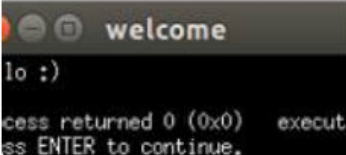
ΔΗΜΙΟΥΡΓΙΑ ΕΝΟΣ MANIFEST

Δημιουργούμε ένα νέο δημοσίευμα repository με όνομα myproject-manifest, με readme.

Πριν προχωρήσουμε ας δούμε λίγο την αρχική οθόνη του προφίλ μας.

```
include <iostream>
main()

std::cout << "Hello :)
return 0;
```



Dimitris
Katkaridis

dkati

[Add a bio](#)

✉ dkatkaridis@gmail.com

Overview Repositories **41** Stars **0** Followers **15** Following **6**

Search repositories...

Type: **All** ▾

Language: **All** ▾

 **New**

myproject-manifest

Updated 13 seconds ago

myTestRepo2

Updated 6 minutes ago

myTestRepo

This repo is made for tutorial purposes

Updated an hour ago

Παρατηρούμε ότι πλέον έχουμε 3 repositories. Το ένα είναι το myTestRepo που ενδεχομένως να είναι το ένα από τα εργαλεία που sourcecode μου, το δεύτερο είναι το myTestRepo2 το οποίο είναι κάποιο 2 εργαλείο μου και το άλλο είναι το manifest

Πατάμε πάνω στο myproject-manifest και επιλέγουμε από δεξιά «Create new file»
Στην επιλογή ονομάτος πληκτρολογούμε «default.xml» και ως περιεχόμενο 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 εξηγήσουμε τον κώδικα

Η πρώτη σειρά αφορά την μορφή του XML.

remote είναι το μέλος του xml που ορίζει από που θα τραβάει το git service, όλα τα sources.

το μέλος path του project, είναι το τοπικό directory που θέλουμε να πάει το κατεβασμένο source code

Το name είναι το github name του repository

Revision είναι το branch το οποίο θα εμφανίζεται στον τοπικό φάκελό μας και το branch το οποίο θα κατεβαζουμε/κάνουμε push.

Κάτω από την επικεφαλίδα Commit new file ορίζουμε το Commit title.

Εδώ ορίζουμε τον τίτλο του commit που θα φαίνεται στα commits. Γενικότερα πρέπει να προσεχούμε τα commit title που κάνουμε καθώς είναι αυτά που κάποιος τρίτος βλέπει. Οπότε η εικόνα που βγαίνει προς τα έξω πρέπει να είναι προσεγμένη. Ένα τυπικό πρότυπο commit title είναι το εξής:

<dir></subdir> : <Τι έχω αλλάξει>

Πχ αν έκανα αλλαγές μέσα σε μια class σε ένα αρχείο στο directory mysource/src/libs/mylib.cpp

Τότε στο commit title μπορώ να γράψω -> src/lib: Donot expose _var from mylib

ή κάτι παρόμοιο. Γενικότερα προσπαθούμε να κάνουμε ευστοχά commit titles χωρίς μεγάλη έκταση

Στο description βαζουμε την περιγραφή του commit, αν θέλουμε να εξηγήσουμε κάτι, και πατάμε commit new file

Μετά από τη δημιουργία του αρχείου μπαίνουμε στο αρχείο readme.md και επιλέγουμε το δεξιά μολύβι ώστε να το επεξεργαστούμε. Μέσα στο αρχείο θα γράψουμε την κύρια εντολή που μας κατεβαίνει το manifest στον υπολογιστή και το ρυθμίζει. Η εντολή είναι η


```
repoint -u git://github.com/dkati/myproject-manifest.git -b master
```



Στη συνέχεια πρέπει να συνταξουμε ένα commit title.


Ενας αποδεκτος τιτλος θα μπορουσε να ειναι ->**readme:Add our repo init command**

Αφου κανουμε το commit επιστρεφουμε στο myproject-manifest

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

 **dkati** committed on **GitHub** readme:Add our repo init command Latest commit 7eb7f47 an hour ago


 README.md	readme:Add our repo init command	an hour ago
 default.xml	Create default.xml	an hour ago

 **README.md**

myproject-manifest

Get our manifest

```
repo init -u git://github.com/dkati/myproject-manifest.git -b master
```



ΑΡΧΙΚΟΠΟΙΗΣΗ ΤΟΥ MANIFEST ΚΑΙ ΤΩΝ SOURCECODES

Απο τη στιγμή που ετοιμάσαμε τα repositories και το manifest, μπορούμε πλέον να ξεκινήσουμε τη διαδικασία της αποθήκευσης των repositories αυτών, στον υπολογιστή μας. Εκτελούμε

```
cd  
mkdir myrossources && cd myrossources
```

Αν μας βγάλει μήνυμα σχετικά με τα χρώματα του λογαριασμού πατάμε 'Υ' και προχωράμε. Με μια πρώτη ματιά μέσα στο φάκελο δεν φαίνεται να υπάρχουν αρχεία. Παρόλα αυτά με την εντολή ls -a παρατηρούμε ότι υπάρχει ένα νέος φάκελος .repo

```
~/myrossources$ ls  
~/myrossources$ ls -a  
..  .repo  
~/myrossources$
```

Στο φάκελο αυτό υπάρχει το symlink manifest.xml το οποίο δείχνει στο .repo/manifests/default.xml το οποίο είναι το manifest που γράψαμε πριν.

Ο φάκελος .repo έχει μέσα τα απαραίτητα αρχεία για να δουλέψει το git/reposervice και δεν θα ασχοληθούμε με αυτά.

Πηγαίνουμε λοιπόν ένα directory πίσω και εκτελούμε

```
reposync
```

ΠΡΟΣΟΧΗ.

Το reposync εκτελείται παντοτε στο ίδιο directory με τον φάκελο .repo

Αν καταλάθως εκτελεστεί η εντολή μέσα σε κάποιον φάκελο, θα υπάρχει πρόβλημα σε όλο το source

```

[redacted]~/myrossources$ ls
[redacted]~/myrossources$ ls -a
.  ..  .repo
[redacted]~/myrossources$ cd .repo
[redacted]~/myrossources/.repo$ ls
manifests manifests.git manifest.xml repo
[redacted]~/myrossources/.repo$ cd ..
[redacted]~/myrossources$ repo sync

... A new repo command ( 1.23) is available.
... You should upgrade soon:

cp /home/[redacted]/myrossources/.repo/repo/repo /usr/bin/repo

Fetching project dkati/myTestRepo
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %         %         Dload  Upload   Total   Spent    Left   Speed
0         0     0    0      0     0      0      0  --:--:--  --:--:--  --:--:--    0
curl: (22) The requested URL returned error: 404 Not Found
Server does not provide clone.bundle; ignoring.
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
From https://github.com/dkati/myTestRepo
* [new branch]      master    -> github/master
Fetching projects: 50% (1/2) Fetching project dkati/myTestRepo2
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %         %         Dload  Upload   Total   Spent    Left   Speed
0         0     0    0      0     0      0      0  --:--:--  --:--:--  --:--:--    0
curl: (22) The requested URL returned error: 404 Not Found
Server does not provide clone.bundle; ignoring.
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
From https://github.com/dkati/myTestRepo2
* [new branch]      master    -> github/master
Fetching projects: 100% (2/2), done.
[redacted]~/myrossources$

```

Μετα απο τη τελευταια μας εντολη , εχουμε κατεβασει ολο το sourcecodeστον υπολογιστη μας.

Μπορουμε επισης να κατεβασουμε μονο το 1 repositoryεαν θελουμε κανοντας

reposyncmyTestRepoLocalDir

Αυτοθακανε μια ανανεωση το τοπικο sourcecodeμε αυτο του github.Αυτο βοηθαι 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 reposync για το repository που θελω

Το reposyncμπορει να παρει ως ορισμα τα threadsμε τα οποια θα κανει sync.

reposync -j4

Γιατι ομως να μας ενδιαφερει το ποσα threadστρεχουν κατα το reposyncαπο τη στιγμη που γινονται ολα μεσω ιντερνετ ?

Γιατι το repo ,κατεβαζει το sourcecodeσε συμπιεσμενη μορφη,αποσυμπιεζει τα αρχεια και μεσω μιας διαδικασιαςdiffcheckελεγχει ποια αρχεια απο το sourceμας εχουν αλλαξει ωστε να τα αντικαταστησει. Αν εχουμε προβλημα με το μεγαλυτερο -jaπλα το μειωνουμε.

Σημειωση , το -jεπιρρεαζει και τις εργασιες που κανει το δικτυο κατα το κατεβασμα.Πρακτικα ,μεγαλυτερο jσημαινει 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 COMMITΚΑΙ ΤΟ ΠΡΩΤΟ PUSH

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

- 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 3^η περιπτωση.

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

Εστω οτι θελουμε να κανουμε pushμια 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 ένα αρχείο αλλάζω 2 μεταβλητες απο int σε double και σε ένα άλλο αρχείο αλλάζω το ονομα μιας class.

Στο commit title βάζω "src:mylibSwitch 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

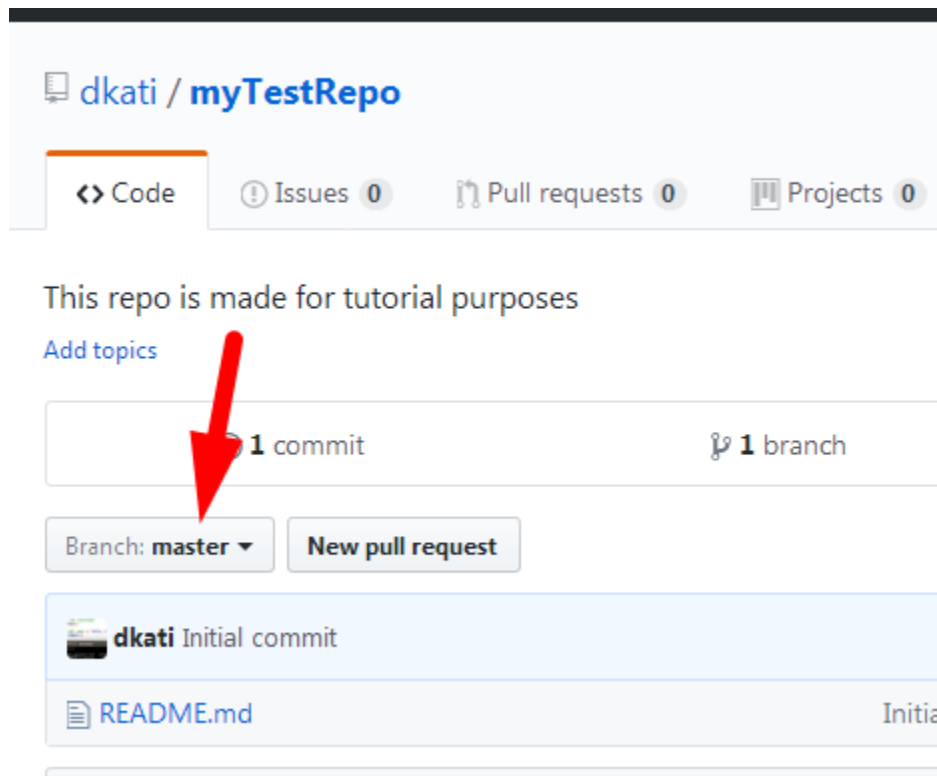
```
cd myTestRepoLocalDir
```

Αφου μπηκαμε στο directory πρεπει να δηλωσουμε το branch στο οποιο ειμαστε. Αρχικα να δουμε αν κατα τυχη ειμαστε ηδη σε branch. Αυτο μπορεί να προκληθει απο καποια παλιότερη 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.



Αν θελω να στείλω τις μελλοντικές μου αλλαγές στο branch αυτό τότε εκτελώ

```
git branch master  
git checkout master
```

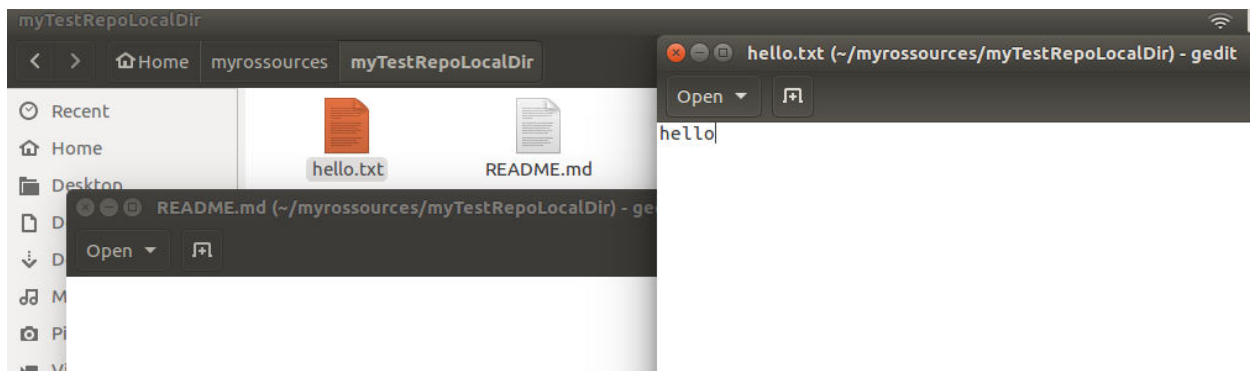
Ή πιο απλά `git checkout -b master`

```
user@myrossources/myTestRepoLocalDir$ git branch  
* (no branch)  
user@myrossources/myTestRepoLocalDir$ git branch master  
user@myrossources/myTestRepoLocalDir$ git checkout master  
Switched to branch 'master'  
user@myrossources/myTestRepoLocalDir$
```

Απο τη στιγμή που άλλαξα το branch μου, μπορώ είτε να κρατήσω το τερματικό ανοιχτό είτε να το κλείσω. Δεν υπάρχει κάτι που τρέχει στο background.

Κάνω λοιπόν τις αλλαγές μου τοπικά.

Θα προσθέσουμε ένα αρχείο με όνομα `hello.txt`, μέσα θα βάλουμε τη λέξη "Hello" και θα σβήσουμε από το `readme` ότι περιέχει



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 που κανουμε επεξεργασια το commit μας. Η πρωτη σειρα ειναι παντα το commit title. Αφηνοντας μια σειρα και προσθετοντας μια τριτη, γραφουμε την περιγραφη.

```
My first commit!
thats my first commit!YaY!
# Please enter the commit message for your
# with '#' will be ignored, and an empty me
# On branch master
# Changes to be committed:
#   modified:   README.md
#   new file:   hello.txt
#
```

Παρατηρουμε τις δυο σειρες

modified: README.md

newfile: 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 reposync 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 reposync και μετα paste τα αρχεια που ειναι επεξεργασμενα

Πατάμε ctrl+X και γύγια να κλείσει το commit. Βλέπουμε στο τερματικό ότι δείχνει το τίτλο του commit και αναφέρει ότι 2 αρχεία άλλαξαν, 1 προσθήκη και 2 διαγραφές. Οι προσθήκες και οι διαγραφές αφορούν ΣΕΙΡΕΣ κωδικά. Επίσης μας δείχνει ότι δημιουργήθηκε το hello.txt με mod 0644

Από τη στιγμή που έγινε το commit, πρέπει να το κάνουμε push.

`git push origin master`

Η εντολή αυτή θα προσπαθήσει να στείλει το commit στο branch master.

Παρολαυτά μας βγάζει κάποια fatal errors. Ο λόγος είναι ότι χρειάζεται να φτιάξουμε (απαιτείται μόνο μια φορά) το remote. Το remote είναι υπεύθυνο για την αποστολή του τοπικού commit, στο προφίλ μας. Συνεπώς εκτελούμε (μια φορά και μόνο φορά)

`git remote add origin`

Και ξανακάνουμε `git push origin master`

Θα μας ρωτήσει αν θέλουμε πραγματικά να γίνει το push (Θα ρωτήσει μόνο για μια φορά και ποτέ ξανά). Πατάμε yes και το στέλνει. Αν βλέπουμε την παρακάτω οθόνη, τότε όλα έγιναν σωστά

```
~/myrossources/myTestRepoLocalDir$ git push origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
~/myrossources/myTestRepoLocalDir$ git remote add origin git@github.com:dkati/myTestRepo.git
~/myrossources/myTestRepoLocalDir$ git push origin master
The authenticity of host 'github.com (192.30.253.112)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUPJWGL7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.253.112' (RSA) to the list of known hosts.
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 305 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github.com:dkati/myTestRepo.git
  5a6f92a..f8d94ec master -> master
```

Στα commits λοιπόν του project, πλέον βλέπω την αλλαγή μου

https://github.com/dkati/myTestRepo/commits/master

Αναζήτηση

This repository Search Pull requests Issues Marketplace Gist

dkati / myTestRepo

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master

Commits on Jun 9, 2017

My first commit! f8d94ec

dkati committed 18 minutes ago

Initial commit 5a6f92a

dkati committed 5 hours ago

Κανοντας κλικ πανω στο commit βλεπω τις λεπτομεριες.

https://github.com/dkati/myTestRepo/commit/f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e

Αναζήτηση

This repository Search Pull requests Issues Marketplace Gist

dkati / myTestRepo

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

My first commit! Browse files

thats my first commit!YaY!

master

dkati committed 19 minutes ago 1 parent 5a6f92a commit f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e

Showing 2 changed files with 1 addition and 2 deletions. Unified Split

2 README.md

@@ -1,2 +0,0 @@

1 -# myTestRepo

2 -This repo is made for tutorial purposes

1 hello.txt

@@ -0,0 +1 @@

1 +hello

0 comments on commit f8d94ec

Lock conversation

Παρατηρούμε όλα τα στοιχεία του commit. Το τίτλο, τις αλλαγές μας και την περιγραφή.

Οι κοκκινές γραμμές είναι ότι σβήστηκε από το αρχείο ενώ οι πράσινες είναι αυτές που έχουν προστεθεί. Αν όλες οι γραμμές από ένα αρχείο είναι πράσινες, σημαίνει πως το αρχείο έχει δημιουργηθεί κατά το commit.

ΑΝΤΙΣΤΡΟΦΗ ΕΝΟΣ COMMIT

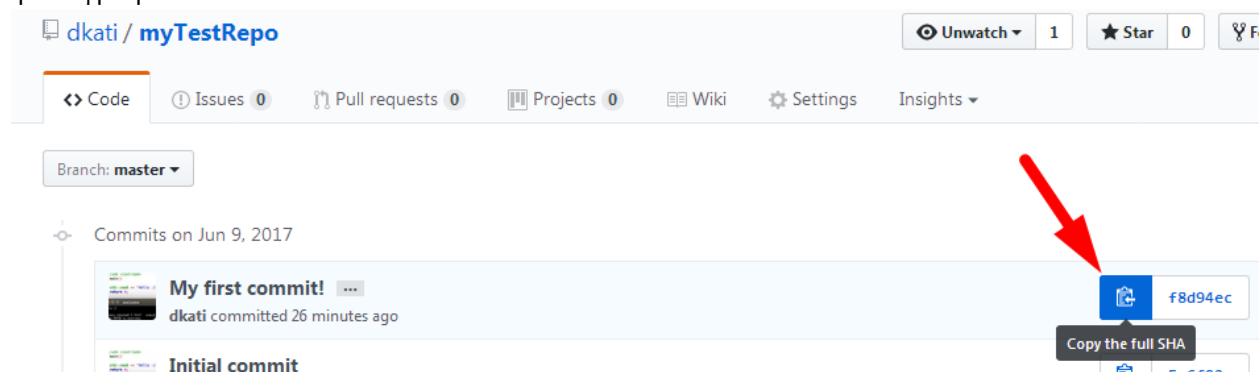
Εχουμε ήδη αναφέρει πως στο github δεν μπορούμε να σβήσουμε commits. Ότι κάνουμε push παραμένει στο ιστορικό. Παρόλα αυτά μπορούμε να αντιστρέψουμε ένα commit και αυτό γίνεται με μια μόνο εντολή.

`git revert SHA_CODE`

Το SHA είναι το μοναδικό κλειδί που έχει κάθε commit και το βρίσκουμε μέσα από το commit ή από το ιστορικό.



Οποτε είτε το παίρνουμε με copy paste από εκεί είτε με ένα απλό κλικ, όπως φαίνεται στην κάτω φωτογραφία



Αυτό θα μας αποθηκεύσει στο clipboard, το commit SHA

Αρα τρεχουμε `git revert`

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`

```
Revert "My first commit!"

i believe its not needed.breaks the build
This reverts commit f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   modified:   README.md
#   deleted:    hello.txt
#
```

Και κανουμε το `push`

`Git push origin master`

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

dkati / myTestRepo

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master

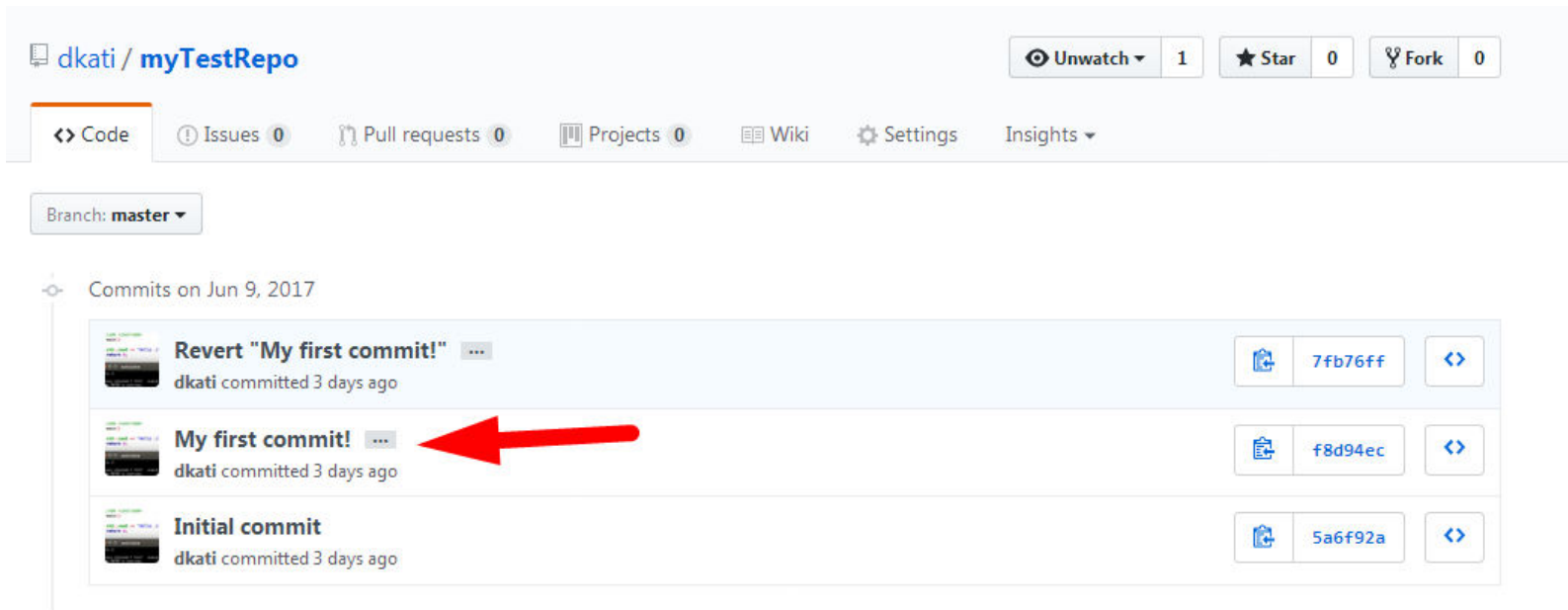
Commits on Jun 9, 2017

	Revert "My first commit!" ...		7fb76ff	
dkati committed 4 minutes ago				
	My first commit! ...		f8d94ec	
dkati committed 32 minutes ago				
	Initial commit		5a6f92a	
dkati committed 5 hours ago				

CHERRY-PICKING.Η «ΝΟΜΙΜΗ» ΑΝΤΙΓΡΑΦΗ ΚΩΔΙΚΑ

Οποιοσδήποτε κωδικας στο githubμπορει να ασφαλιστει με καποιο διεθνες license(apache,MITκτλπ). Τα licenseπροστατευουν τον συντακτη απο τυχον «κλοπες» του κωδικα.Το githubχει μεριμνησει για την εξασφαλιση της νομιμοτητας μεσω των λειτουργιων του,και δινει τη δυνατοτητα στο χρηστη να αντιγραψει ενα commitκαποιου τριτου προγραμματιστη ,δινοντας τα απαιριτητα credits. Το cherry-pickμας βοηθαει στο να ανανεωνουμε κομματα κωδικα απο κωδικες αλλων προγραμματιστων,δινοντας παντα αυτοματως τα απαιριτητα credits.

Για να μπορεσουμε να κανουμε cherry-pick,πρεπει πρωτα να κατεβασουμε ΟΛΟΚΛΗΡΟ του sourcecode,μαζί με τα commits,απο το οποιο θελουμε καποια συγκεκριμενα commits. Εστω οτι θελουμε να κανουμε cherry-pickτο commitπου καναμε πιο πριν "myfirstcommit" απο το <https://github.com/dkati/myTestRepo>



The screenshot shows the GitHub interface for the repository `dkati / myTestRepo`. The top navigation bar includes links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Settings, and Insights. The current branch is `master`. Below the navigation bar, the commit history for June 9, 2017, is displayed. The commits are listed in reverse chronological order:

- Revert "My first commit!"** (commit hash: 7fb76ff) - dkati committed 3 days ago
- My first commit!** (commit hash: f8d94ec) - dkati committed 3 days ago (highlighted with a red arrow)
- Initial commit** (commit hash: 5a6f92a) - dkati committed 3 days ago

Αφου επιλεξαμε το commitπου θελουμε πρεπει να παμε μεσω του τερματικου στο directoryτου projectπου θελω να βαλω το commit.Εστω οτι θελουμε να μπει στοmyTestRepo2LocalDir.Εκτελουμε

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

Η συνταξη της εντολης ειναι
`gitfetch<linkτου προτζεκτ που θελουμε><ονομα branch>`

Με την παραπανω εντολη κρατησαμε το ιστορικο των commitτου συγκεκριμενου repository.

```

~/myrossources/myTestRepo2LocalDir$ git fetch https://github.com/dkati/mytestrepo master
warning: no common commits
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 8 (delta 0), reused 7 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), done.
From https://github.com/dkati/mytestrepo
* branch      master      -> FETCH_HEAD
~/myrossources/myTestRepo2LocalDir$

```

Πλεον ειμαστε ετοιμοι να «τραβηξουμε» το commit

```
git cherry-pick f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e
```

Η συνταξη της εντολης είναι

Git cherry-pick <SHA code>

Τρεχοντας την εντολη μας βγαζει errorπου αναφερει οτι δεν μπορει να προσαρμοσει το commitμεσα στον κωδικα μας.

Οταν κανουμε cherry-pick,κατι τετοιο είναι πολυ συχνο και οφειλουμε να ειμαστε σε θεση να το διορθωσουμε.Το αποτελεσμα της εντολης είναι το παρακατω

```

~/myrossources/myTestRepo2LocalDir$ git cherry-pick f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e
error: could not apply f8d94ec... My first commit!
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
Recorded preimage for 'README.md'
~/myrossources/myTestRepo2LocalDir$

```

Κοιτωντας το commitπαρατηρουμε οτι οι αλλαγες είναι σε 2 αρχεια.

https://github.com/dkati/myTestRepo/commit/f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e

Αναζήτηση

This repository Search Pull requests Issues Marketplace Gist

dkati / myTestRepo Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

My first commit!

thats my first commit!YaY!

master

dkati committed 19 minutes ago 1 parent 5a6f92a commit f8d94ec598e8e94ccfb5c61ca0c5c973ca61cc1e

Showing 2 changed files with 1 addition and 2 deletions.

Unified Split

2 README.md

<> View

```
@@ -1,2 +0,0 @@
1 -# myTestRepo
2 -This repo is made for tutorial purposes
```

1 hello.txt

View

```
@@ -0,0 +1 @@
1 +hello
```

0 comments on commit f8d94ec

Lock conversation

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

```
README.md (~/.myrossources/myTestRepo2LocalDir) - gedit
Open
<<<<<< HEAD
# myTestRepo2
=====
>>>>>> f8d94ec... My first commit!
```

Ας αναλυσουμε τη συνταξη του αρχειου

<<<<<<<< HEAD

myTestRepo2

=====

Αυτο σημαίνει πως κανονικά το αρχείο μας περιέχει ότι βρίσκεται ανάμεσα στο

<<<<<<<< και το =====

Απο το ===== μέχρι το >>>> είναι αυτό που περιέχει η αλλαγή του commit.

Στη συγκεκριμένη περίπτωση το περιεχόμενο ανάμεσα στο ===== και στο >>>> είναι κενό.

αυτό σημαίνει ότι το commit που κάνουμε cherry-pick σβήνει ότι περιέχεται από το <<<<

ως το ===== (το HEAD δηλαδή). Συνεπώς η σωστή λύση εδώ είναι να σβήσω τα πάντα.

Αποθηκεύω το αρχείο και αφού πλέον έχουμε κρατήσει τις αλλαγές μας ξεκινάμε το git commit μας

```
git add -A
git commit
```

και παρατηρούμε ότι έχει ήδη συμπληρώσει το commit title και το description. Πατάμε ctrl+x για να βγούμε από το nano. Πλέον μπορούμε να κάνουμε push το commit.

```
git push origin master
```

Στη περίπτωση μας θα μας βγάλει πρόβλημα στο remote. Οπότε όπως και πριν

```
git remote add origin
```

Και ξανά git push origin master

```
~/myrossources/myTestRepo2LocalDir$ git add -A
~/myrossources/myTestRepo2LocalDir$ git commit
Recorded resolution for 'README.md'.
[master 7fe3622] My first commit!
Date: Fri Jun 9 16:22:05 2017 +0300
2 files changed, 1 insertion(+), 1 deletion(-)
create mode 100644 hello.txt
~/myrossources/myTestRepo2LocalDir$ git push origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
~/myrossources/myTestRepo2LocalDir$ git remote add origin git@github.com:dkati/mytestrepo2.git
~/myrossources/myTestRepo2LocalDir$ git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 313 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 1 (delta 0)
To git@github.com:dkati/mytestrepo2.git
5461521..7fe3622 master -> master
~/myrossources/myTestRepo2LocalDir$
```

Πραγματι παρατηρηουμε οτι το commit εγινε επιτυχως

USJ | <https://github.com/dkati/myTestRepo2/commits/master> Αναζήτηση

This repository Search Pull requests Issues Marketplace Gist

dkati / myTestRepo2 Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master

Commits on Jun 12, 2017

My first commit! 7fe3622

dkati committed 3 days ago

Commits on Jun 9, 2017

Initial commit 5461521

dkati committed 3 days ago

USJ | <https://github.com/dkati/myTestRepo2/commit/7fe36227387ab7e1cee0e61dff8ce7efedfa6cf6> Αναζήτηση

This repository Search Pull requests Issues Marketplace Gist

dkati / myTestRepo2 Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

My first commit! 7fe3622

thats my first commit!YaY!

master

dkati committed 3 days ago 1 parent 5461521 commit 7fe36227387ab7e1cee0e61dff8ce7efedfa6cf6

Showing 2 changed files with 1 addition and 1 deletion. Unified Split

1 README.md View

... @@ -1,0 +0,0 @@

1 -# myTestRepo2

1 hello.txt View

... @@ -0,0 +1 @@

1 +hello

0 comments on commit 7fe3622

Αυτο που παρατηρούμε στα 2 commit είναι ότι άλλες αλλαγές έγιναν στο commit του myTestRepo και άλλες έγιναν στο commit του myTestRepo2.

Αυτος είναι και ο λόγος που ειχαμε conflict κατά το cherry-pick. Είναι σπάνιες οι φορές που το cherry-pick θα γίνει χωρίς conflict

Όταν ένα commit γίνεται cherry-pick και ο συντακτής είναι διαφορετικό άτομο από αυτόν που έκανε το cherry-pick τότε το commit εμφανίζεται έτσι



sensitive_pn: Add Canadian sensitive numbers ...

Dmole committed with brinlyau 9 days ago

Ο πραγματικός συντακτής είναι ο **brinlyau** ενώ αυτός που έκανε το cherry-pick είναι ο **Dmole**

ΕΠΑΝΕΓΓΡΑΦΗ ΤΗΣ ΙΣΤΟΡΙΑΣ ΤΩΝ COMMIT (ADVANCED USERS)

Όπως έχουμε αναφέρει, δεν είναι εφικτή η **ΔΙΑΓΡΑΦΗ** των commits. Παρόλα αυτά υπάρχει μια εντολή που επαναφέρει **ΟΛΟΚΛΗΡΟ** το commit history σε μια παλαιότερη έκδοση σβηνοντας τα ενδιάμεσα commits. Η εντολή αυτή είναι λίγο επικίνδυνη καθώς διαγράφει ότι commit υπάρχει

Πρακτικά δεν τα διαγράφει την ύπαρξη τους απλά δεν φαίνονται στο commit history. Επίσης αναιρούνται από τον τοπικό source code. Η διαδικασία αναιρέσης της διαγραφής είναι δύσκολη και απαιτείται μεγάλη εξοικείωση με το github shell.

Η συνταξή της εντολής είναι

git reset --hard SHA_CODE (Διπλές παύλες)

Πχ Αν θέλω να «σβήσω»

το τελευταίο commit του myTestRepo2, τότε αρκεί να κάνω reset στο προτελευταίο.

git reset --hard 5461521f50d472f7950f330608438a70634b435e

και με το φειλωνατο κάνω push

git push -f origin master

Το -f σημαίνει

«force». Το github το κάνει αυτό για ασφάλεια. αν δοκιμάσουμε να κάνουμε git push origin master δεν θα μας αφήσει.


```
~/myrossources/myTestRepo2LocalDir$ git reset --hard 5461521f50d472f7950f330608438a70634b435e
HEAD is now at 5461521 Initial commit
~/myrossources/myTestRepo2LocalDir$ git push -f origin master
Warning: Permanently added the RSA host key for IP address '192.30.253.113' to the list of known hosts.
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:dkati/mytestrepo2.git
+ 7fe3622...5461521 master -> master (forced update)
~/myrossources/myTestRepo2LocalDir$
```

Αν το μετανιωσουμε μπορούμε να πάρουμε το σβησμένο commit (είναι το **7f3622**) κανοντας πάλι reset

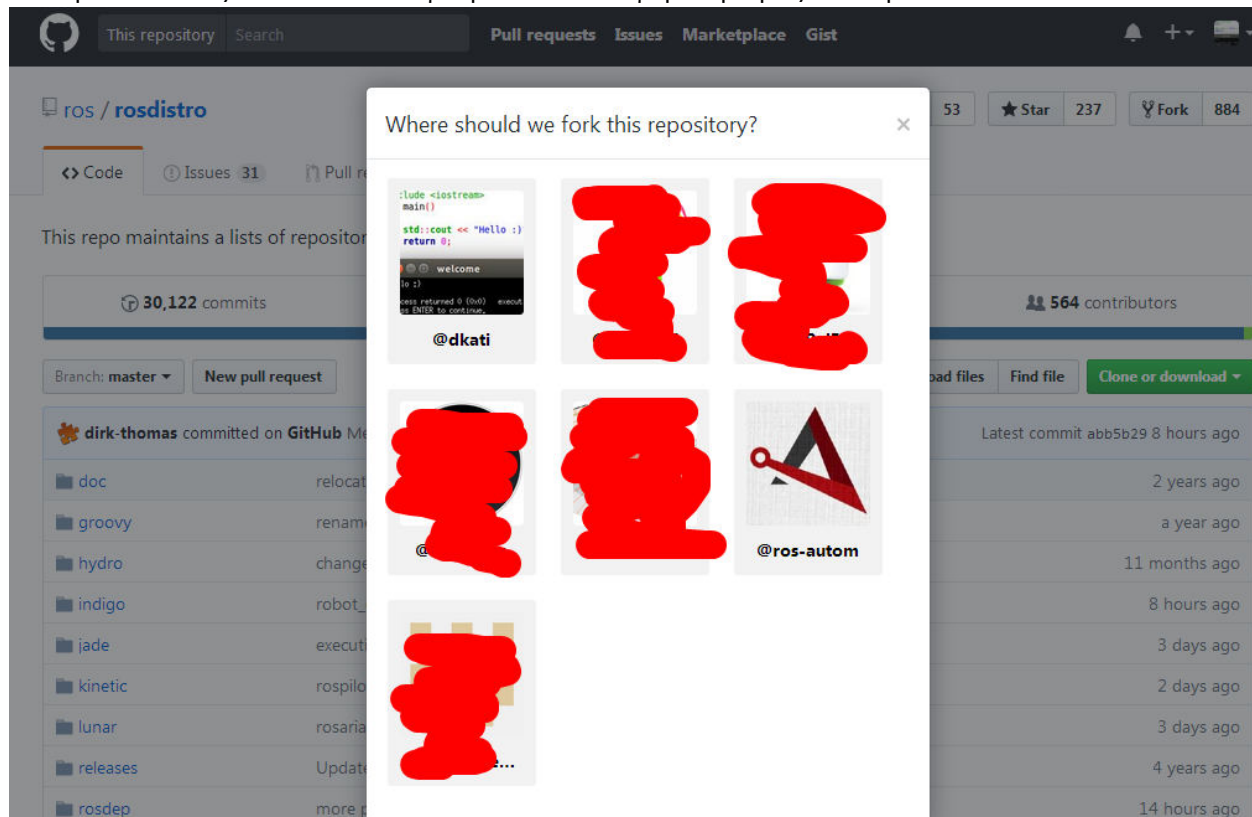
```
gitreset --hard 7f3622
```

Συνεπώς καταλαβαίνουμε ότι το resetγραφεί την ιστορία είτε προς τα πίσω είτε προς τα εμπρός.

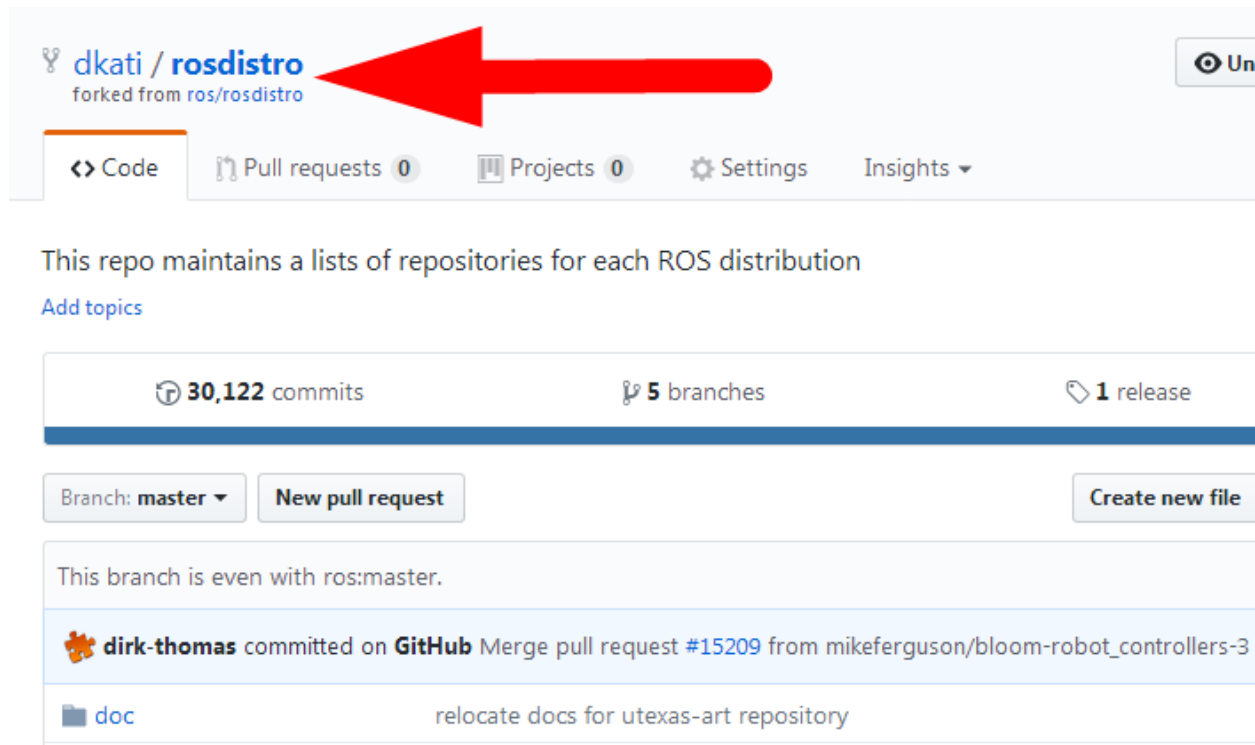
FORKING.Ο ΝΟΜΙΜΟΣ ΤΡΟΠΟΣ ΝΑ ΑΝΤΙΓΡΑΨΟΥΜΕ ΟΛΟΚΛΗΡΑ REPOSITORIES

Forkingείναι η διαδικασία που αντιγράφουμε πλήρως ολόκληρο το repositoryκαποιού.Είναι πολύ απλό και επιτυγχάνεται με 2 απλά click.

Πηγαίνουμε στο repository το οποίο θέλουμε να κάνουμε fork. Εστω το github.com/ros/rosdistro. Πατάμε πάνω δεξιά fork και επιλεγούμε σε ποιο λογαριασμό μας θέλουμε να παει.



Το repository εμφανίζεται στο προφίλ μας όπως παρακάτω. φαινεται πολυ καθαρα οτι το repository ειναι εργο του ros/distro



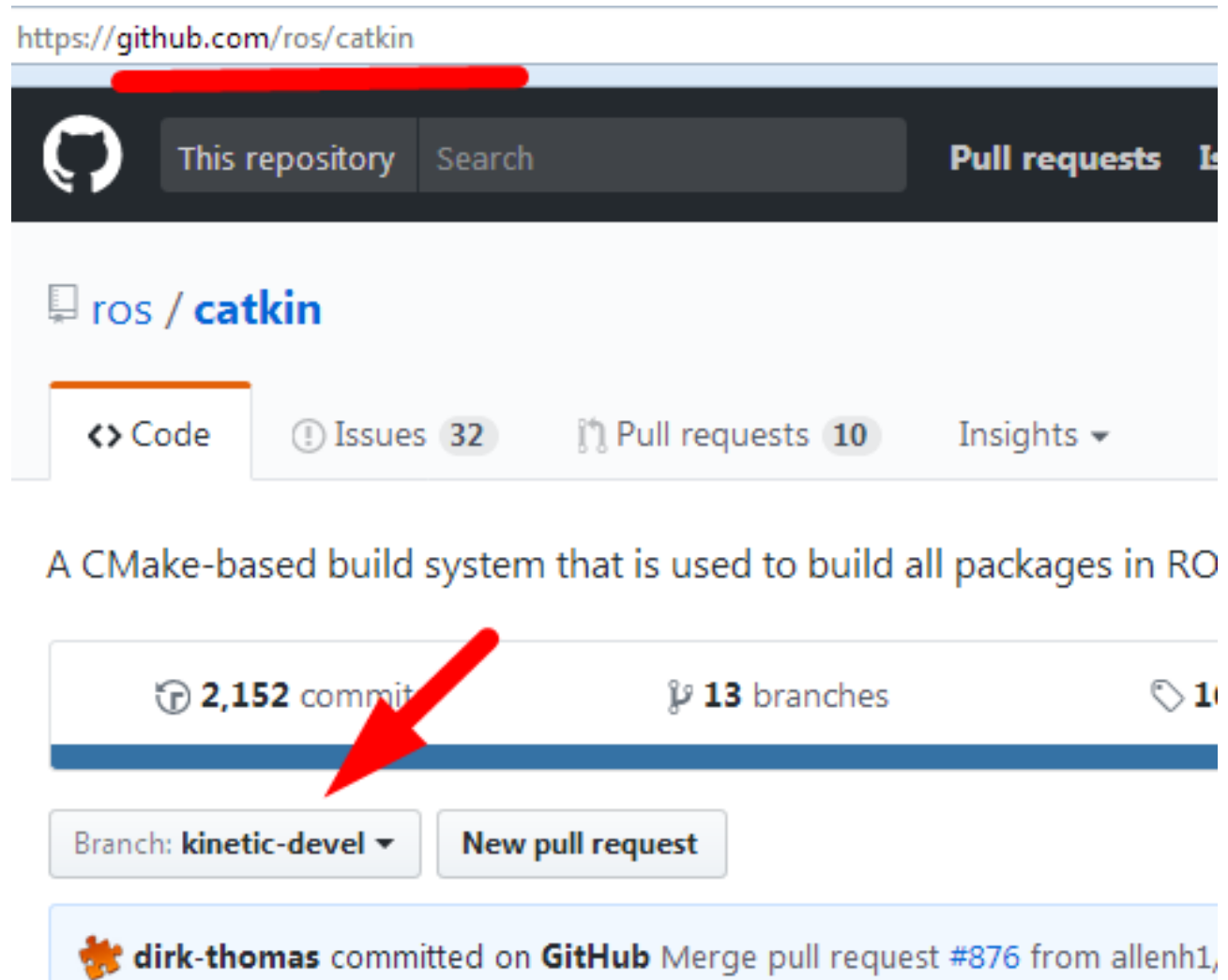
Απο εκεί και περα χρησιμοποιω το projectοπως ακριβως εργαστηκαμε με τα αλλα 2 repositoriesτων παραδειγματων.

Παντοτε πρεπει να θυμομαστε οτι κατα το πρωτο pushπου θελουμε να κανουμε , χρειαζομαστε gitremoteaddorigin

ΓΡΑΨΙΜΟ ΟΛΟΚΛΗΡΗΣ ΙΣΤΟΡΙΑΣ ΑΠΟ ΑΛΛΟ REPOSITORY

Εστω οτι έχω ένα repository με branch master και θελω να φτιαξω ένα ολοκληρο branch το οποιο θα είναι πρακτικά, το branch ενός τριτου repository.

Εστω λοιπον οτι θελω στο myTestRepo να φτιαξω ένα 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 ένα 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
```

```
kinetic-devel
```

```
cd kinetic-devel
```

```
git branch
```

Παρατηρούμε ότι είμαστε στο branch που έχουμε κατεβάσει. Από το σημείο αυτό φτιάχνουμε το δικό μας branch.

```
git branch mykinetic
```

```
git branch
```

Πλέον βλέπουμε ότι έχουμε 2 branches και ξέρω ότι το mykinetic έχει προερχθεί από το kinetic-devel. Με πράσινο χρώμα είναι το τοπικό branch στο οποίο βρισκόμαστε. Αλλάζουμε λοιπόν το branch

```
git checkout mykinetic
```

Αφού είμαι στο branch, κάνω push ότι υπάρχει στο branch (και source code και commit history). Αν γράψουμε `Git push origin mykinetic` θα μας πει ότι δεν έχουμε δικαιώματα να κάνουμε push το οποίο είναι προφανές καθώς δεν ανήκουμε στην ομάδα του ros. Αρα πρέπει να φτιάξουμε νέο origin

```
git remote add myorigin
```

```
git push -f myorigin mykinetic
```

Χρησιμοποιώ -f γιατί αναγκάζω το github να μου φτιάξει στη σελίδα νέο branch

```
~/myrossources/myTestRepoLocalDir$ git clone https://github.com/ros/catkin kinetic-devel
Cloning into 'kinetic-devel'...
remote: Counting objects: 12110, done.
remote: Total 12110 (delta 0), reused 0 (delta 0), pack-reused 12109
Receiving objects: 100% (12110/12110), 3.34 MiB | 1.26 MiB/s, done.
Resolving deltas: 100% (6467/6467), done.
Checking connectivity... done.
~/myrossources/myTestRepoLocalDir$ cd kinetic-devel
~/myrossources/myTestRepoLocalDir/kinetic-devel$ git branch
* kinetic-devel
~/myrossources/myTestRepoLocalDir/kinetic-devel$ git branch mykinetic
~/myrossources/myTestRepoLocalDir/kinetic-devel$ git branch
* kinetic-devel
  mykinetic
~/myrossources/myTestRepoLocalDir/kinetic-devel$ git checkout mykinetic
Switched to branch 'mykinetic'
~/myrossources/myTestRepoLocalDir/kinetic-devel$ git push -f origin mykinetic
Username for 'https://github.com': dkati
Password for 'https://dkati@github.com':
remote: Permission to ros/catkin.git denied to dkati.
fatal: unable to access 'https://github.com/ros/catkin/': The requested URL returned error: 403
~/myrossources/myTestRepoLocalDir/kinetic-devel$ git remote add myorigin git@github.com:dkati/mytestrepo.git
~/myrossources/myTestRepoLocalDir/kinetic-devel$ git push -f myorigin mykinetic
Counting objects: 11112, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4919/4919), done.
Writing objects: 100% (11112/11112), 3.15 MiB | 1.23 MiB/s, done.
Total 11112 (delta 5839), reused 11108 (delta 5836)
remote: Resolving deltas: 100% (5839/5839), done.
To git@github.com:dkati/mytestrepo.git
 * [new branch]    mykinetic -> mykinetic
~/myrossources/myTestRepoLocalDir/kinetic-devel$
```

Εν τέλει στο repository μας παρατηρούμε το νέο branch το οποίο είναι ΑΚΡΙΒΩΣ ίδιο με το πραγματικό branch του ros

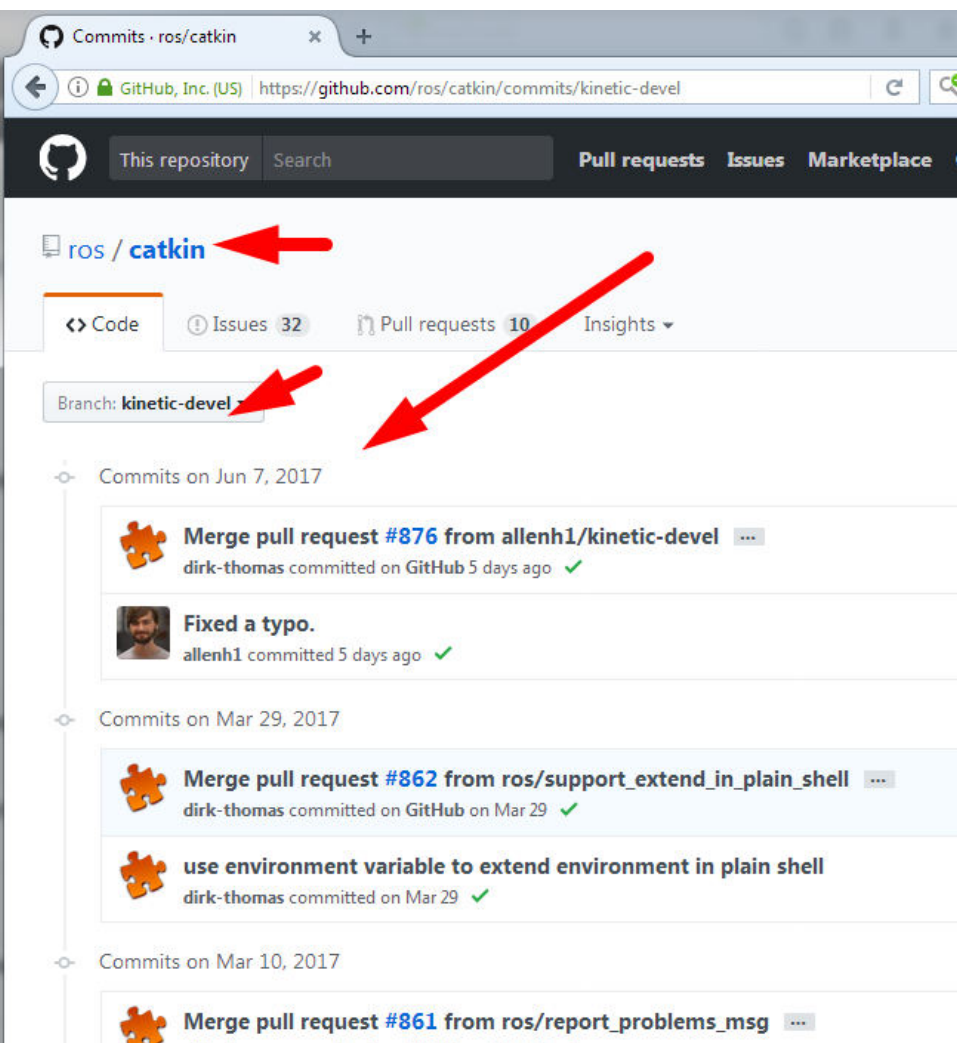
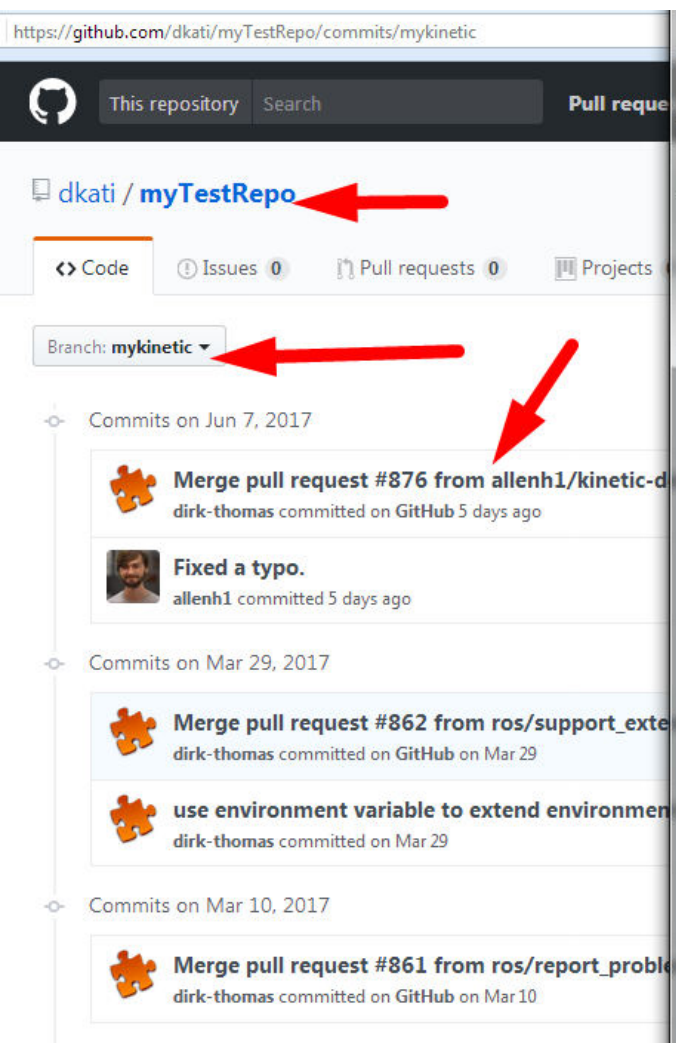
The screenshot displays the GitHub interface for the repository 'dkati / myTestRepo'. At the top, there are navigation tabs: 'This repository', 'Search', 'Pull requests', 'Issues', 'Marketplace', and 'Gist'. Below the repository name, there are buttons for 'Unwatch', 'Star', and a fork icon. A secondary navigation bar includes 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Settings', and 'Insights'. A message states 'This repo is made for tutorial purposes' with a link to 'Add topics'. Repository statistics are shown: 2,152 commits, 2 branches, 0 releases, and 1 contributor. Under 'Your recently pushed branches:', the 'mykinetic' branch is listed as pushed 8 minutes ago, with a 'Compare & pull' button. Below this, a 'Branch: mykinetic' dropdown and a 'New pull request' button are visible. A 'Switch branches/tags' modal is open, showing a search bar and a list of branches: 'master' and 'mykinetic' (which is checked and highlighted). In the background, a list of pull requests is partially visible.

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

```
cd ../../  
rm -rf myTestRepoLocalDir  
repo sync myTestRepoLocalDir --force-sync
```

--force-syncκαλο ειναι να χρησιμοποιουμε σε καθε reposyncγια να το αναγκασουμε να «τραβηξει» καθε νεα αλλαγη και να σβησει τυχον ξεχασμενα λαθη στο τοπικο source

Πολυ σημαντικό είναι να παρατηρήσουμε ότι ακόμη και το commit history είναι ΑΚΡΙΒΩΣ ίδιο με το αυθεντικό repository του ros



- `Git branch`
Εμφάνιση των διαθεσιμων τοπικων branches.Επισης εμφανιζει το branchστο οποιο ειμαστε τωρα(εμφανιζεται με πρασινα γραμματα).
- `Git branch<branchname>`
Δημιουργιαενοςνεουbranch ,αποτοbranchπου ειμαστε ηδη
- `Git checkout <branch name>`
Μεταβασησεαλλοbranch
- `Git checkout -b<branchname>`
Δημιουργιαενοςνεουbranchκαι μεταβαση σε αυτο
- `Gitbranch -D<branchname>`
Διαγραφηενοςbranch.ΔΕΝ ΜΠΟΡΟΥΜΕ ΝΑ ΔΙΑΓΡΑΨΟΥΜΕ ΤΟ BRANCHΣΤΟ ΟΠΟΙΟ ΕΙΜΑΣΤΕ
- `Git clone <github link απο repository> <directory>`
Κατεβασμα του repository (default branch) και αποθηκευση σε ενα νεο φακελο με ονομα το ονομα του directory.Ο φακελος βρισκεται στο ιδιο directory που ειμαστε με το terminal
- `git clone -b <BRANCH> <github link απο repository> <directory>`
Κατέβασμα του repository (Το branch που εχουμε δωσει) και αποθηκευση σε ενα νεο φακελο με ονομα το ονομα του directory.Ο φακελος βρισκεται στο ιδιο directory που ειμαστε με το terminal
- `Git fetch<github link απο repository><branchname>`
Κατεβασμα του commit history του repository.ΠΡΕΠΕΙ ΝΑ ΤΡΕΞΕΙ ΜΕΣΑ ΣΤΟ DIRECTORY ΣΤΟ ΟΠΟΙΟ ΘΑ ΚΑΝΩ ΤΟ cherry-pick
- `repoint -ugit://<link απο manifest>.git -bmaster`
Αρχικοποιηση του manifest
πχ `repoint -ugit://github.com/dkati/myproject-manifest.git -bmaster`
- `git cherry-pick<SHA>`
Νομιμηαντιγραφηενοςcommitμε συγκεκριμενο SHA
- `git reset --hard <SHA>`
Επαναφορατουcommit history στοσυγκεκριμενοcommit
- `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>`
Ανεβασματοςcommit ,μεσωτουremote,στοbranch name
- `git remote add <remote name>git@github.com:<repo name>.git`
Δημιουργιαremote
- `git remote remove <remote name>`
Σβησιμremote

- Repo sync ή repo sync –force-sync
Συγχρονισμός τοπικού source code με αυτόν στο github. –force-sync
αν θέλω να αναγκάσω το github να σβήσει τις τοπικές αλλαγές μου

Όλες οι εντολές πρέπει να εκτελούνται μέσα στο root directory του κάθε repository. Εκεί δηλαδή όπου υπάρχει φακέλος .git

***=

Όταν το github μας αναφέρει Recorded preimage σημαίνει ότι έχει καταγράψει το λάθος ΚΑΙ ΤΗΝ ΕΠΙΛΥΣΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ του χρήστη. Αυτό το κάνει για να λύσει αυτομάτως το ίδιο πρόβλημα σε κάποιο μελλοντικό cherry-pick. Πολλές φορές όμως δεν θέλουμε να θυμάται το πως το λύσαμε γιατί ενδεχομένως η λύση να μην είναι ίδια. Έτσι λοιπόν χρησιμοποιώ την εντολή

git reset --hard

Με την εντολή αυτή λοιπόν αναγκάζω το github να «ξεχάσει» οποιαδήποτε πιθανή λύση ενός conflict.