ΟΔΗΓΟΣ ΧΡΗΣΗΣ GIT

Περιεχόμενα

1. Προτού Ξεκινήσουμε

-Περιγραφή

Το Git είναι το πιο δημοφιλές Version Control System (VCS, σύστημα ελέγχου εκδόσεων), ένα εργαλείο δηλαδή που αναλαμβάνει την καταγραφή αλλαγών σε αρχεία οποιουδήποτε είδους, ώστε να μπορεί να γίνει εύκολος διαμοιρασμός των αλλαγών και σύγκριση των version των αρχείων. Δίνει τη δυνατότητα σε κάθε χρήστη να έχει τοπικά ένα αντίγραφο της τελευταίας έκδοσης των αρχείων καθώς και όλο το ιστορικό των αλλαγών, παρέχοντας πληροφορίες για κάθε αλλαγή, όπως η ημερομηνία/ώρα που έγινε και τον χρήστη που την έκανε. Πιο συγκεκριμένα, το Git είναι μια συλλογή από εργαλεία (command line tools) που μας επιτρέπουν να ανταλλάσσουμε τις δικές μας αλλαγές, με τις αλλαγές των συνεργατών μας.

-Γιατί Git;

1)Ένα VCS όπως το Git, είναι με διαφορά ο καλύτερος τρόπος να συντονίζει μια ομάδα την δουλειά της. Ξεχάστε την ανταλλαγή αρχείων μέσω Drive ή ακόμα χειρότερα μέσω Facebook! Με μερικά απλά βήματα έχουμε άμεσα τις αλλαγές των συνεργατών μας στον υπολογιστή μας αυτόματα, χωρίς να πρέπει να αντιγράψουμε κώδικα και να συγκρίνουμε αρχεία.

2)Είναι ένας καλός τρόπος να κρατάμε backup της δουλειάς μας. Ακόμα και αν για κάποιο λόγο χάσουμε τα αρχεία μας από την τοπικό υπολογιστή, έχουμε ένα αξιόπιστο backup στον server που τα φιλοξενεί.

3)Είναι πολύ δύσκολο να χαλάσουμε κάτι. Έχοντας το πλήρες ιστορικό των αλλαγών του project διαθέσιμο, μπορούμε οποιαδήποτε στιγμή να ανατρέξουμε σε μία προηγούμενη έκδοση και να δούμε ξεκάθαρα τις προβληματικές αλλαγές, ή κάποιες γραμμές που μπορεί να διαγράψαμε κατά λάθος.

4)Δίνει ξεκάθαρη εικόνα για την πρόοδο του project. Μέσω των περιγραφικών μηνυμάτων που υποβάλλονται με κάθε αλλαγή, είναι εύκολο να δούμε και να αξιολογήσουμε την πρόοδο του project.

5)Χρησιμοποιείται σχεδόν παντού. Δεν υπάρχει περίπτωση να δουλέψουμε σε εταιρία που δε χρησιμοποιεί κάποιο είδος VCS, οπότε η γνώση αυτού του εργαλείου εκτιμάται ιδιαίτερα.

Να σημειωθεί ότι ένα VCS είναι ιδιαίτερα χρήσιμο ακόμα και αν δουλεύουμε μόνοι μας σε ένα project, για τους περισσότερους λόγους που αναφέρθηκαν, και είναι χρήσιμο εργαλείο όχι μόνο για προγραμματισμό, αλλά για οποιοδήποτε project απαιτεί συνεργασία σε αρχεία.

-Προαπαιτούμενα

Αρχικά πρέπει να κατεβάσουμε τα εργαλεία του Git (*https://git-scm.com/downloads*). Για την εγκατάσταση, αρκεί να αφήσουμε όλες τις επιλογές στο default.

Στη συνέχεια πρέπει να επιλέξουμε ένα hosting website για το project μας, όπως τα GitHub, Bitbucket, GitLab κλπ και να φτιάξουμε έναν λογαριασμό. Εκεί ουσιαστικά θα είναι ο server που θα αποθηκεύσουμε το repository μας (ο κεντρικός φάκελος του project μαζί με το ιστορικό αλλαγών). Ας επιλέξουμε το GitHub.

Ουσιαστικά αυτά είναι τα μόνα τελείως απαραίτητα βήματα για να μπορέσετε να χρησιμοποιήσετε τα εργαλεία του Git, αλλά επειδή η χρήση του command line για τη διαχείριση είναι ίσως άβολη για τους περισσότερους, θα εξετάσουμε τη χρήση κάποιου third-party GUI διαχειριστικού προγράμματος (στην προκειμένη περίπτωση το GitHub Desktop *https://desktop.github.com* , για διευκόλυνση, μιας και είναι πολύ απλό) και τη χρήση του Git απευθείας μέσω του IntelliJ IDEA, για να επιλέξετε έναν από τους δύο τρόπους.

Για να αποσαφηνίσουμε ξανά τις έννοιες: το Git είναι τα εργαλεία που μας δίνουν όλες τις δυνατότητες που αναφέραμε μέσω εντολών τερματικού, το GitHub είναι ένας hosting provider για το project/ένα repository management

system που κρατάει τα αρχεία μας και το GitHub Desktop είναι μια GUI διεπαφή με τα εργαλεία του Git (αν και το Git έρχεται από μόνο του με ένα GUI πρόγραμμα *"Git GUI"*, το οποίο όμως δεν είναι ιδιαίτερα βολικό).

-GitHub: repositories, licenses, .gitignore, collaborators

Αφού έχουμε φτιάξει το account μας στο GitHub λοιπόν και έχουμε συνδεθεί, ας δούμε κάποιες βασικές έννοιες που ίσως φανούν χρήσιμες.

Αρχικά πάμε να φτιάξουμε ένα repository (repo), την αποθήκη αρχείων δηλαδή του project μας. Από την αρχική σελίδα, πατάμε πάνω αριστερά στο πράσινο "New" ή πάνω δεξιά στο + και μετά "New repository". Μας ζητείται ένα όνομα για το repo και ένα προαιρετικό description για αυτό. Στη συνέχεια καλούμαστε να επιλέξουμε αν θα είναι public, δηλαδή αν θα μπορεί να το δει ο καθένας που μπαίνει στο προφίλ μας ή αν θα είναι private και θα μπορούμε να το δούμε μόνο εμείς και όποια άτομα θέσουμε ως collaborators. Ύστερα υπάρχει ένα dropdown που σε καλεί να επιλέξεις αρχείο .gitignore. Ουσιαστικά αυτό είναι ένα ειδικό αρχείο που λέει στο Git τι αρχεία και φακέλους να αγνοεί όταν ανεβάζουμε τα αρχεία μας από τον local φάκελο του υπολογιστή μας στον server. Πχ θα μας ενδιέφερε να του πούμε να αγνοήσει ένα αρχείο με κωδικούς ή έναν φάκελο που περιέχει πολλά βίντεο λόγω μεγέθους. Ακόμα θα μας ενδιέφερε αν χρησιμοποιήσουμε Java, να του πούμε να αγνοήσει όλα τα .class αρχεία που παράγονται στο compilation γιατί είναι αχρείαστα. Το GitHub μας παρέχει πολλά templates που μπορεί να μας ενδιαφέρουν, πχ άμα κάνω search για Java, θα φτιάξει μόνο του ένα .gitignore αρχείο για να αγνοεί ότι δε χρειάζεται να ανεβαίνει στον server. Επίσης μπορούμε να απευθυνθούμε σε αυτή τη σελίδα *https://www.gitignore.io* που μπορεί να δημιουργήσει .gitignore για πολλές γλώσσες/IDE/frameworks μαζί. Στη συνέχεια υπάρχει ένα αντίστοιχο dropdown που μας καλεί να επιλέξουμε license. Αυτό είναι μια περιγραφή των δικαιωμάτων που έχουν άλλοι χρήστες πάνω στο project μας. Το να μην γίνει επιλογή license υπονοεί ότι το project ΔΕΝ είναι δωρεάν, ούτε μπορεί να χρησιμοποιηθεί από τρίτους χωρίς την άμεση άδεια του συγγραφέα. Ίσως το πιο απλό και ελεύθερο license που μπορούμε να χρησιμοποιήσουμε να είναι το MIT license.

Τέλος, δίνεται η επιλογή να δημιουργήσουμε ένα README file κατά τη δημιουργία του project, ένα αρχείο δηλαδή που θα περιέχει οδηγίες/επεξηγήσεις/σχόλια σχετικά με το project, και τα περιεχόμενά του θα εμφανίζονται στην κεντρική σελίδα του repository στο GitHub. Βεβαιωθείτε ότι είναι τσεκαρισμένο το κουτί. Αφού κάνουμε create, το επόμενο βήμα είναι να προσθέσουμε συνεργάτες (collaborators) στο project. Εφόσον είμαστε μέσα στο repository που δημιουργήσαμε, βρίσκουμε την επιλογή Settings ως tab πάνω, και από το μενού αριστερά επιλέγουμε Manage access και μετά το πράσινο Invite a collaborator. Κάνουμε search το username ή το email και στέλνουμε αίτημα. Το άλλο άτομο θα πρέπει να τσεκάρει το email του για να αποδεχτεί το αίτημα (προσοχή, αν εμφανίζεται error 404 στο redirect του invitation από το email, είναι επειδή δεν έχει συνδεθεί στο account ο browser). Αφού αποδεχτούν τα άτομα της ομάδας το αίτημα, είμαστε έτοιμοι να ξεκινήσουμε.

2. Δουλεύοντας με το GitHub Desktop

-Αρχικό σετάρισμα

Από την αρχική οθόνη του προγράμματος θα επιλέξουμε το “Clone a repository from the internet” αν έχουμε ήδη δημιουργήσει repository στο GitHub (αλλιώς μπορούμε να δημιουργήσουμε επιτόπου καινούριο repo με σχεδόν την ίδια διαδικασία που αναφέρθηκε εδώ, και ως Local path να χρησιμοποιήσουμε τη διαδικασία που θα περιγραφεί σε λίγο). Στη συνέχεια είτε επιλέγουμε σύνδεση με το GitHub account για να εμφανιστεί η λίστα με τα διαθέσιμα repo του account ώστε να επιλέξουμε αυτό που μας ενδιαφέρει, είτε χρησιμοποιήσουμε το URL του repo (στο GitHub, άμα μπούμε μέσα στο repo, και πατήσουμε το πράσινο “clone or download” μας εμφανίζει το https URL). Σχετικά με το Local path, άμα θέλουμε να έχουμε ένα IntelliJ project μέσα στο repository μας:

1)δημιουργούμε ένα project από το IntelliJ πρώτα (άμα δεν έχουμε ήδη)
2)βρίσκουμε τον φάκελο που είναι αποθηκευμένο (μέσα από το IntelliJ, δεξί κλικ σε ένα αρχείο και show in explorer)
3)μεταφέρουμε προσωρινά όλα τα αρχεία του project σε κάποιον άλλο φάκελο (ώστε να μπορέσουμε να κάνουμε clone στον φάκελο του project, γιατί το clone δε δουλεύει αν το target directory δεν είναι άδειο).
4)επιλέγουμε ως Local path τον φάκελο του project,
έστω “project_folder_name” (αυτόν που του αφαιρέσαμε τα αρχεία).

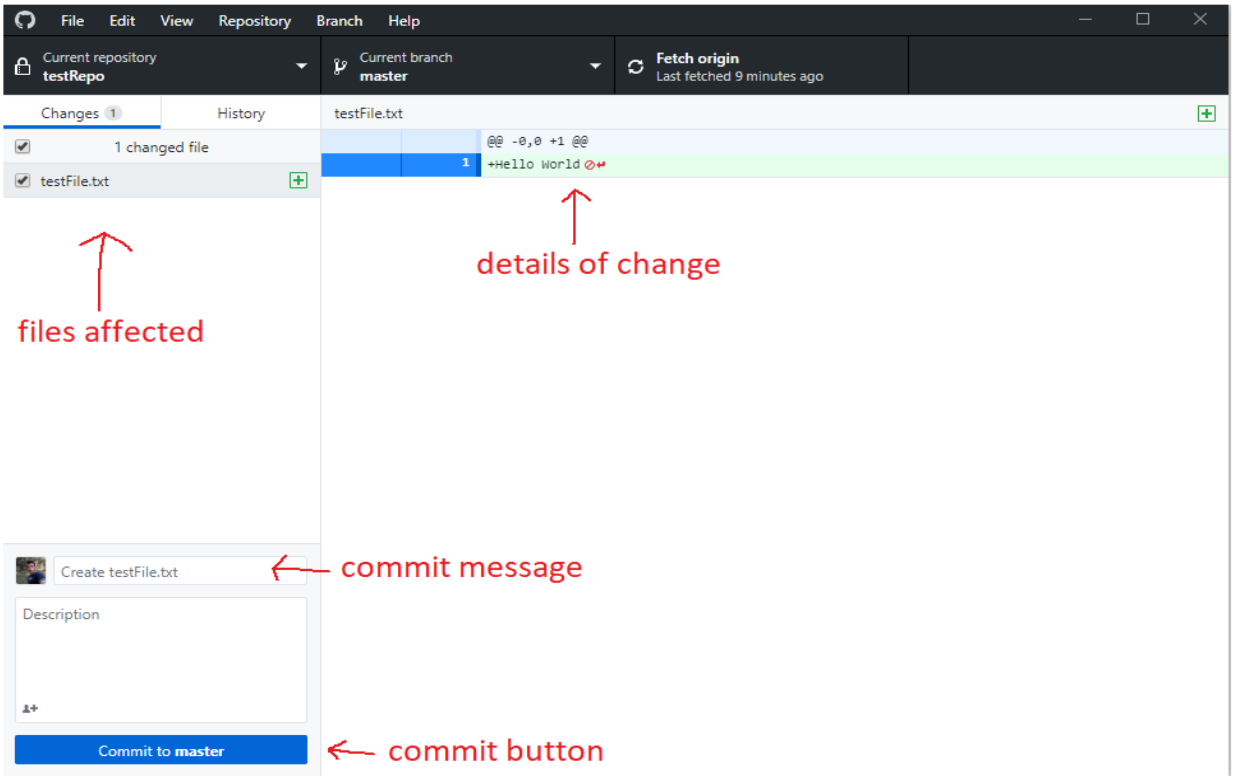
Προσοχή, όταν επιλέγουμε φάκελο για clone, δημιουργείται ένα νέος φάκελος μέσα σε αυτόν, και έχει το όνομα του repo, οπότε το Local path τώρα θα έχει την μορφή:
“C:\Users*user_name*\IdeaProjects*project_folder_name**repo_name*” αλλά στην προκειμένη περίπτωση δε θέλουμε να δημιουργηθεί φάκελος μέσα στον φάκελο που θα πρέπει να είναι τα αρχεία του IntelliJ. Οπότε:

5)διαγράφουμε το “*repo_name*”.
6) Κάνουμε clone και μετά μεταφέρουμε ξανά μέσα στον φάκελο τα αρχεία του IntelliJ

Τώρα μπορούμε να δημιουργήσουμε ένα συνδυαστικό .gitignore για Java και IntelliJ όπως αναφέρθηκε εδώ.

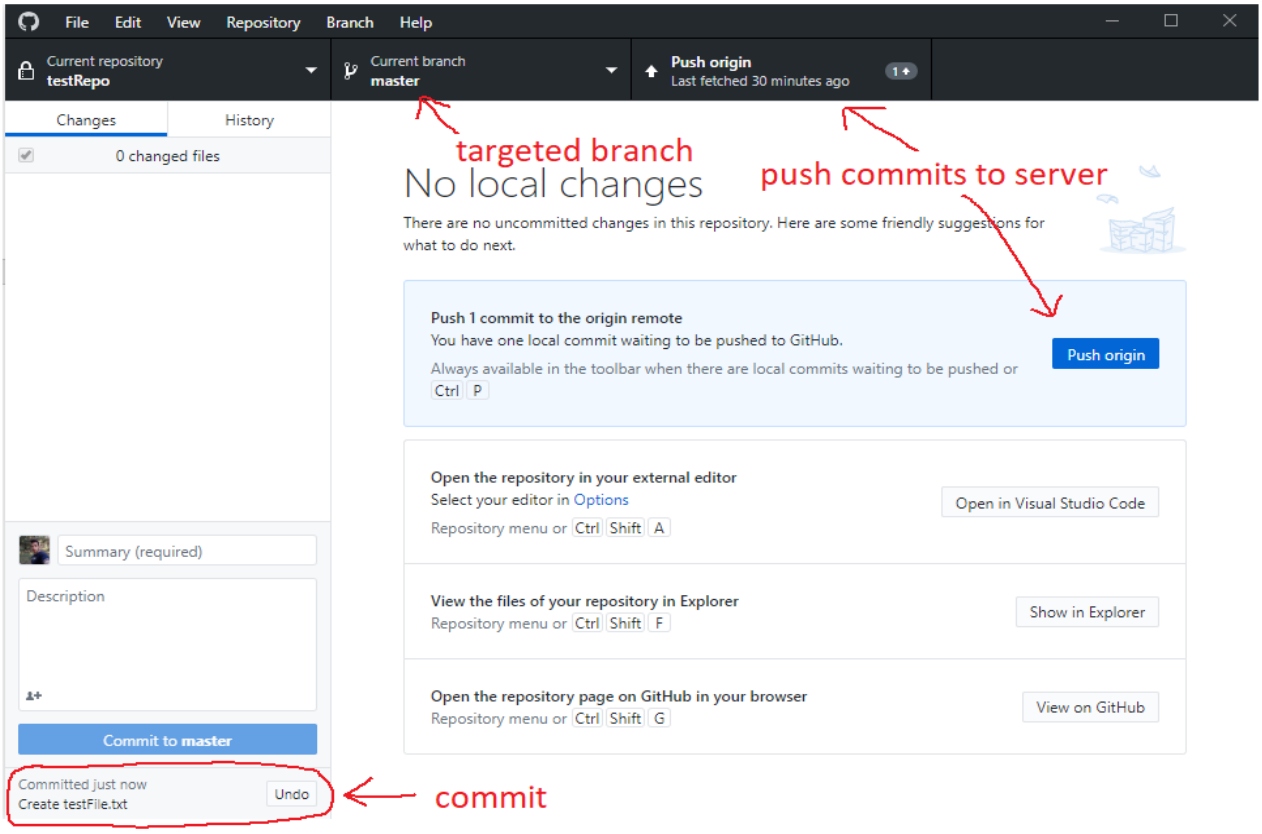
-Οδηγίες χρήσης

Έχοντας κάνει clone το remote repository σε κάποιο φάκελο του pc μας, από εδώ και πέρα ότι προσθήκες/διαγραφές/τροποποιήσεις γίνουν μέσα στον root folder του repository μας θα φαίνονται στο GitHub Desktop. Έστω λοιπόν ότι έχουμε τον συνεργάτη Α και τον Β που έχουν κάνει και οι 2 clone ένα απλό repo που περιέχει ένα README και ένα .gitignore αρχείο μόνο. Ο συνεργάτης Α δημιουργεί το αρχείο testFile.txt με την φράση Hello World.



Αριστερά βλέπουμε κάθε στιγμή όλα τα αρχεία που έχουν υποστεί αλλαγές από την τελευταία φορά που έγινε commit. Δίπλα από το όνομα του αρχείου, υπάρχει ένα αυτόματα επιλεγμένο checkbox που δηλώνει ότι το αρχείο είναι staged/added, δηλαδή θα συμπεριληφθεί στο commit (σαν να έχει εκτελεστεί η εντολή “git add testFile.txt” στο cmd). Μπορούμε να αποεπιλέξουμε αρχεία για να μην εφαρμοστούν οι αλλαγές τους. Κάνοντας δεξί κλικ, μπορούμε να κάνουμε “discard” τις αλλαγές του αρχείου, δηλαδή να το επαναφέρουμε στην μορφή που ήταν στο τελευταίο commit (σε αυτή την περίπτωση θα διαγραφεί γιατί δεν υπήρχε), ή μπορούμε να επιλέξουμε να προστεθεί στο .gitignore για να αγνοούνται πάντα οι αλλαγές του. Δεξιά είναι οι περιοχή που για το επιλεγμένο αρχείο, σου λέει συγκεκριμένα τι αλλαγές έγιναν από την τελευταία φορά. Πιο κάτω καλούμαστε να βάλουμε ένα commit message,

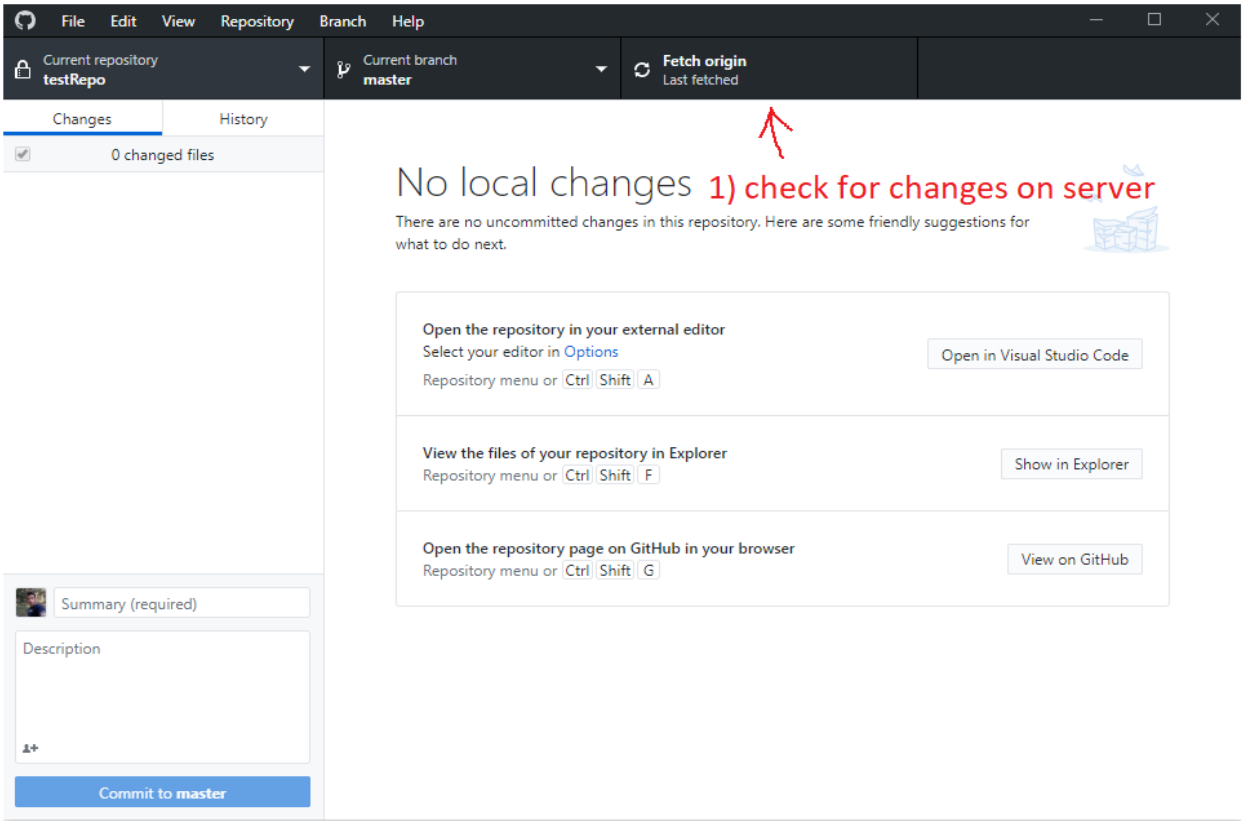
έναν περιγραφικό τίτλο δηλαδή για τις αλλαγές που κάναμε (είναι υποχρεωτικό το commit message, αλλά το GitHub Desktop βάζει ένα δικό του default όπως αυτό που έχει τώρα αν δεν βάλεις εσύ). Επίσης, στο πεδίο description μπορούμε να βάλουμε μια προαιρετική αναλυτική περιγραφή του τι κάναμε σε αυτό το commit, ώστε να διευκολυνθούν οι συνεργάτες μας. Τέλος, μπορούμε να κάνουμε το αρχείο commit, να πούμε δηλαδή ότι οι αλλαγές αυτές θέλουμε να εφαρμοστούν (στο local repository όμως, ο συνεργάτης Β δε μπορεί να τις δει ακόμα). Ουσιαστικά ένα “commit” είναι ένα σύνολο αλλαγών και όταν “κάνω commit” δηλώνω ότι οι αλλαγές που έχω κάνει στα αρχεία μου θέλω να εφαρμοστούν locally στο ιστορικό των αλλαγών.

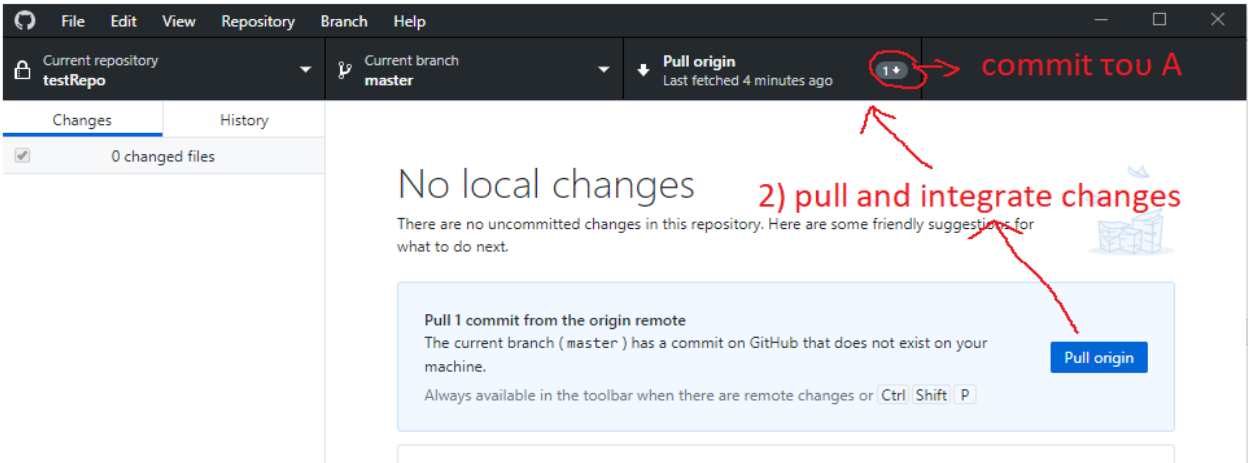


Αφού κάνουμε commit, έχουμε τη δυνατότητα να κάνουμε push, να στείλουμε δηλαδή όλες τις αλλαγές μας (όλα τα commits μας) στον server, ώστε να έχουν πρόσβαση όλα τα μέλη της ομάδας. Να σημειωθεί ότι δεν είναι απαραίτητο να κάνουμε push κάθε φορά που κάνουμε commit, θα μπορούσαμε να συνεχίσουμε να κάνουμε αλλαγές και commits locally και να τα ανεβάσουμε όλα μαζί με ένα push, αν δουλεύουμε πάνω σε features τα οποία δε χρειάζεται να αξιοποιήσει κανένας άλλος άμεσα.

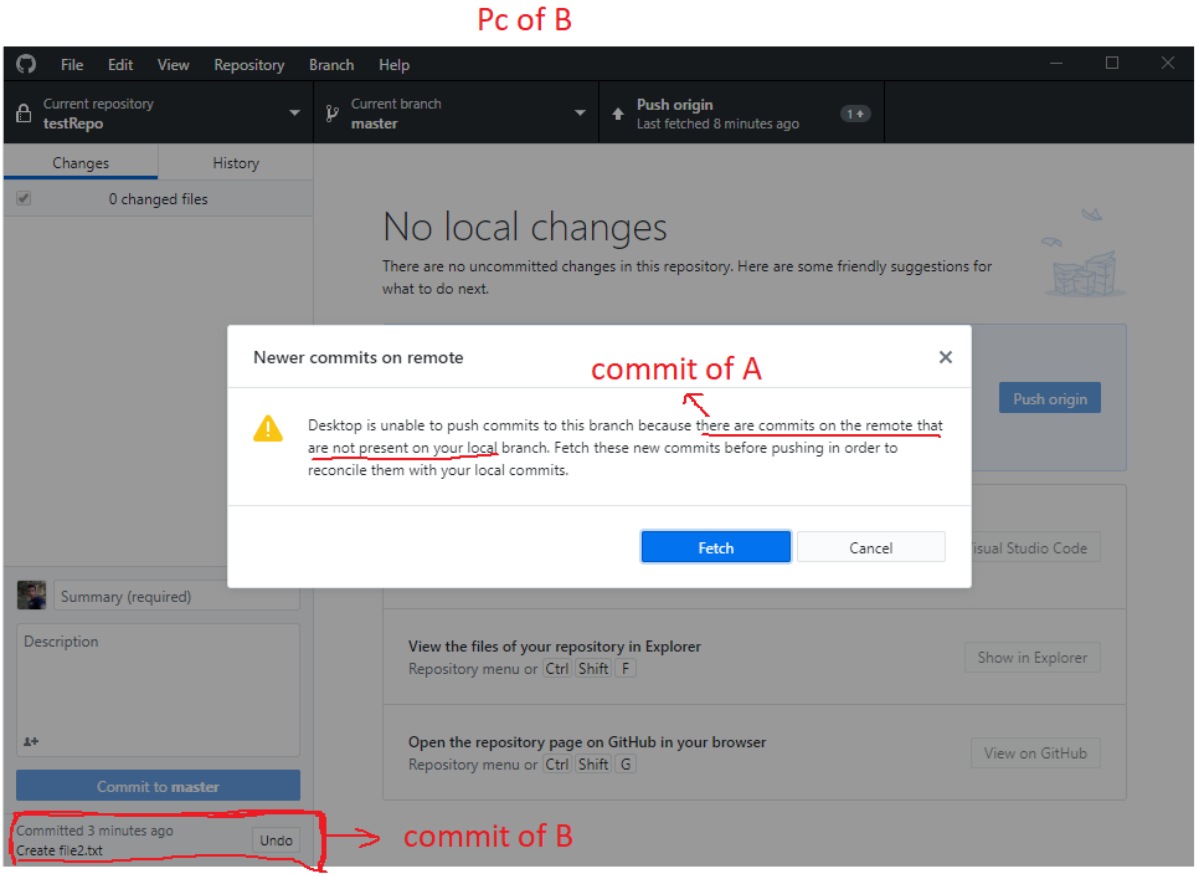
Έστω λοιπόν ότι ο συνεργάτης Α έχει δημιουργήσει το αρχείο, έχει γράψει κάτι σε αυτό και το έχει κάνει push στο GitHub. Ο συνεργάτης Β για να πάρει τις αλλαγές που έχουν σταλθεί στον server, πρέπει να πατήσει το κουμπί “fetch origin” ώστε να ελέγξει για καινούριες αλλαγές και αν υπάρχουν να τις τραβήξει. Ύστερα, μπορεί να κάνει pull αν θέλει να τις ενσωματώσει στα δικά του local αρχεία.

PC συνεργάτη Β, καμία αλλαγή δε φαίνεται





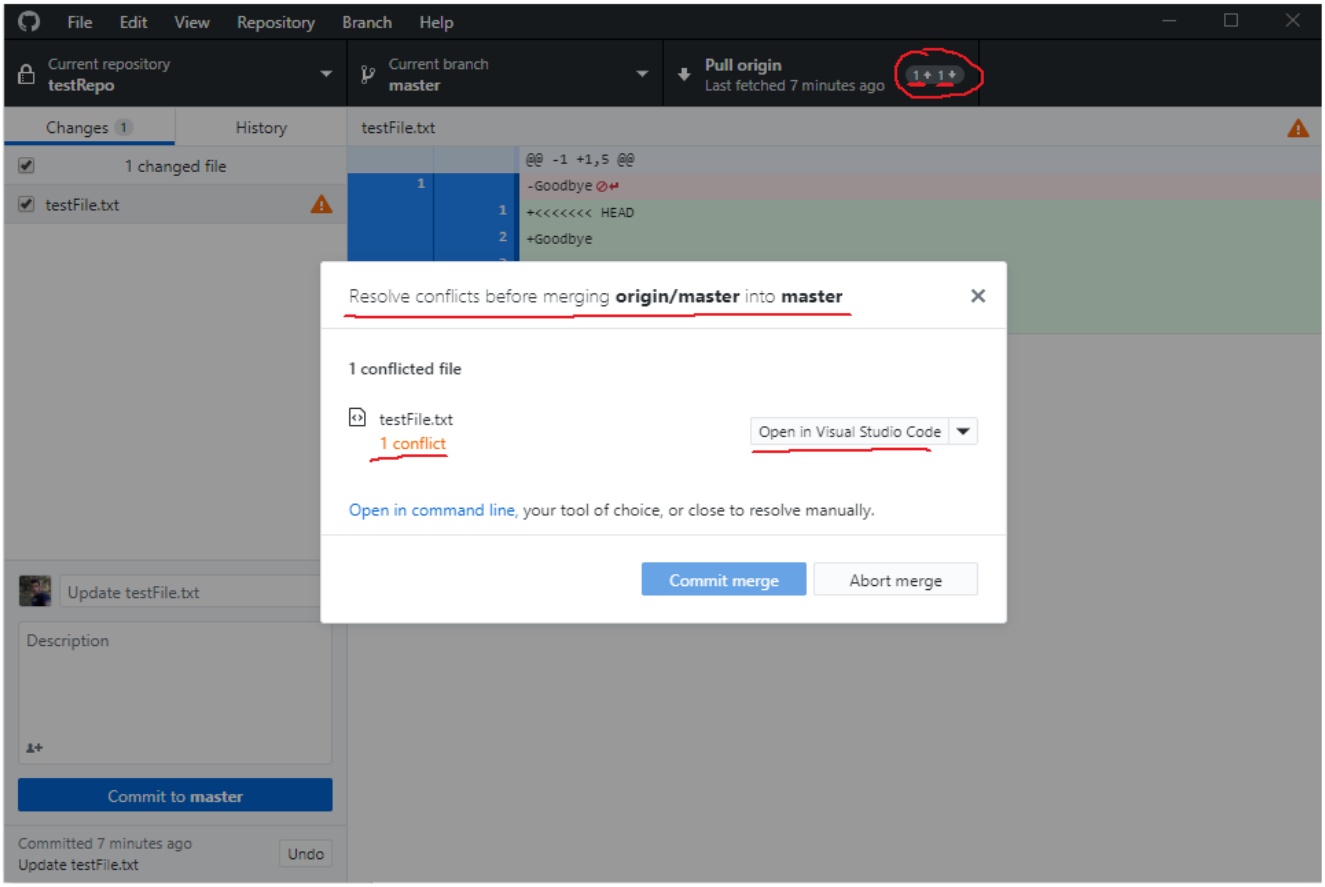
Αυτή ήταν η πιο απλή περίπτωση ως interaction 2 ατόμων. Τι θα συνέβαινε όμως αν ο Β είχε δημιουργήσει και αυτός αρχεία και είχε κάνει δικές του αλλαγές και προσπαθούσε να κάνει push πριν ελέγξει για αλλαγές στον server, ενώ ο Α είχε ήδη κάνει push τις δικές του; Έστω λοιπόν ότι ο Β φτιάχνει και αυτός ένα αρχείο "file2.txt" και γράφει κάτι μέσα. Κάνει commit το αρχείο, βάζοντας ένα μήνυμα, ακριβώς με την ίδια διαδικασία. Τώρα όμως όταν προσπαθεί να κάνει push το commit του:



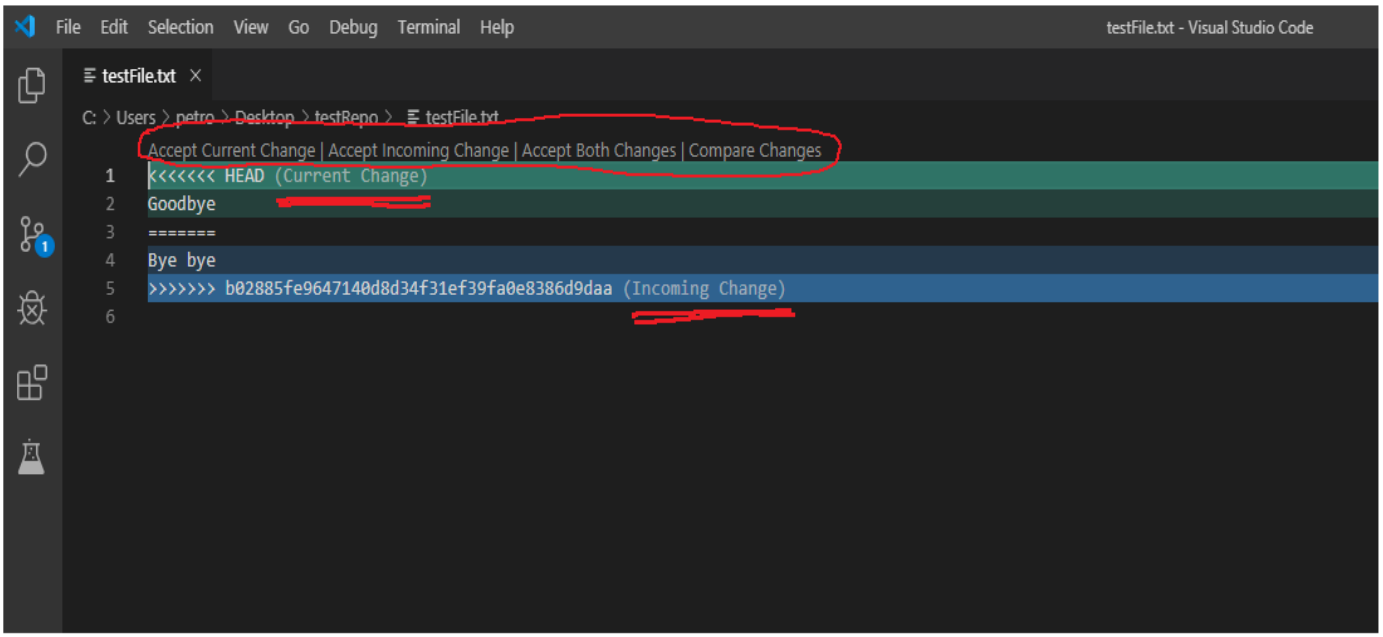
Δεν του επιτρέπεται να κάνει push τις local αλλαγές από την στιγμή που υπάρχουν αλλαγές στον server που δεν τις έχει. Τώρα θα πρέπει πρώτα να τραβήξει τις αλλαγές από τον server και να τις ενσωματώσει στο δικό του version και μετά να κάνει push το ολοκληρωμένο version στον server.
Άρα τώρα έχουμε την ακολουθία: αποτυχημένο push->fetch->pull->push. Και έτσι ολοκληρώνεται και αυτή η εκδοχή interaction. Να σημειωθεί ότι αν ο Β ήξερε ότι ο Α έχει ανεβάσει αλλαγή, θα μπορούσε να είχε κάνει fetch και pull από την αρχή, χωρίς να κάνει το αποτυχημένο push και να δει αυτό το warning.

-Conflict resolution

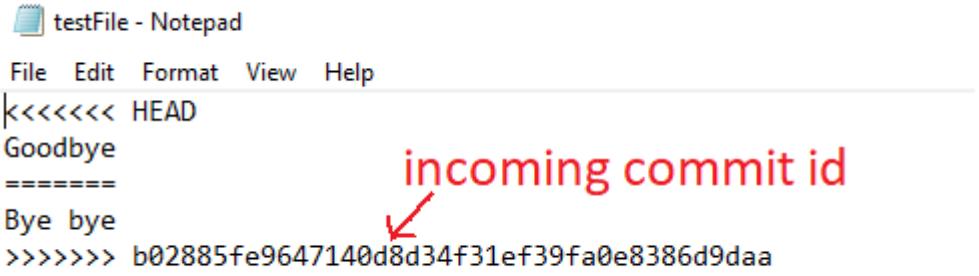
Τώρα θα εξετάσουμε την τελευταία και πιο δύσκολη περίπτωση interaction, την δημιουργία και επίλυση conflicts. Τα conflicts είναι συγκρούσεις που δημιουργούνται συνήθως όταν 2 συνεργάτες πειράζουν το ίδιο πράγμα, πχ την ίδια γραμμή κώδικα ή το όνομα του ίδιου αρχείου. Σε αυτή την περίπτωση το Git δεν ξέρει πως να ενσωματώσει τις αλλαγές και των 2 σε ένα κοινό version γιατί δεν παίρνει ποτέ αυθαίρετες πρωτοβουλίες για το ποιανού τις αλλαγές πρέπει να πετάξει. Δηλαδή έστω ότι στο αρχείο testFile.txt που υπήρχε η φράση “Hello World” γινόταν αλλαγή αυτής της γραμμής από τον A σε “Goodbye” και από τον B σε “Bye bye”, το Git δε μπορεί αυτόματα να αποφασίσει ποια αλλαγή θα θέλαμε να κρατήσει, οπότε χρειάζεται ανθρώπινη βοήθεια. Αυτή η διαδικασία της ένωσης αλλαγών με το χέρι ονομάζεται conflict resolution. Ας το δούμε στην πράξη:
Ο A ετοιμάζεται να κάνει push στον server την αλλαγή του “Hello World” σε “Goodbye” ενώ ο B έχει ήδη pushάρει την αλλαγή σε “Bye bye”, οπότε έχουμε: commit->αποτυχημένο push->fetch->conflict warning on pull



Εγώ έχοντας εγκατεστημένο το Visual Studio Code στο pc μου, έχω την επιλογή να ανοίξω εκεί το conflicted αρχείο και να προσπαθήσω να το διορθώσω. Θα μοιάζει κάπως έτσι

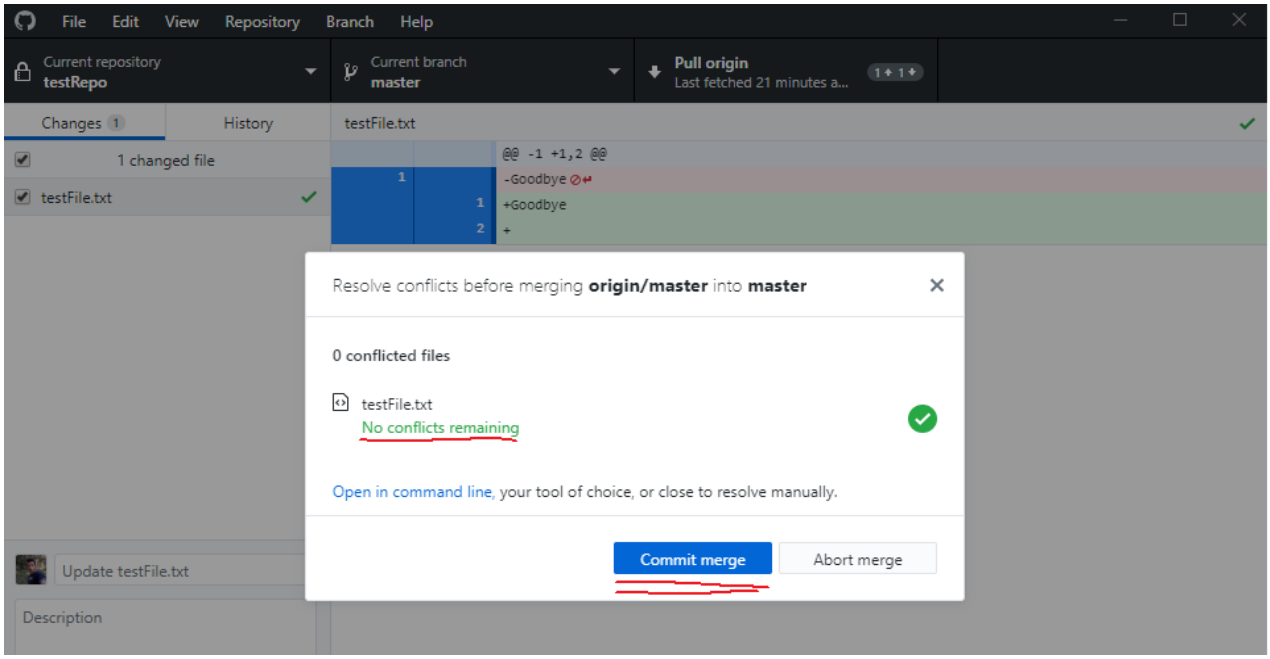


Με μπλε βλέπουμε το version που υπάρχει στον server (incoming change) και με πράσινο το τοπικό version (local change). Επίσης εμφανίζεται και ένα βοηθητικό μενού με 4 κουμπιά. Αν πατήσω το πρώτο, κρατάει το Goodbye, το δεύτερο το αντίθετο, το τρίτο κρατάει και τα 2 το ένα κάτω από το άλλο και το τέταρτο τα ανοίγει δίπλα δίπλα σαν ξεχωριστά αρχεία για πιο βολική προεπισκόπηση. Το θέμα είναι ότι το πως θα εμφανίζεται το conflict resolution εξαρτάται τελείως από το editing πρόγραμμα που θα χρησιμοποιηθεί, πχ στο παρακμιακό notepad των windows, το conflict εμφανίζεται κάπως έτσι



Και ο μόνος τρόπος για να το διορθώσω είναι να διαγράψω τα πάντα με το χέρι εκτός από το “Goodbye” ή το “Bye bye” και να κάνω save.

Όταν διορθωθεί το conflict λοιπόν και γυρίσουμε στο GitHub Desktop βλέπουμε αυτό



Πατώντας το commit merge, δημιουργείται αυτόματα ένα commit με τίτλο "**Merge branch 'master' of https://github.com/subamanis /testRepo**" (μπορούμε να το δούμε στο History tab δίπλα από το Changes, μαζί με όλο το ιστορικό των commits) και μετά μπορούμε να κάνουμε επιτυχώς push.

Ένα tip για να αποφεύγετε τα conflicts είναι να μην δουλεύετε ταυτόχρονα στο ίδιο αρχείο πάνω από ένα άτομο, θα γλυτώσετε πονοκέφαλο.

Συνοψίζοντας, υπάρχουν 3 βασικές κατηγορίες interaction χρησιμοποιώντας το Git

1)commit->push (αν δεν υπάρχουν αλλαγές στον server που δεν έχεις)

2)commit->αποτυχημένο push->fetch->pull->push (αν υπάρχουν αλλαγές)

3)commit->αποτυχημένο push->fetch->conflict warning on pull->resolve-> merge->push

3. Δουλεύοντας με το IntelliJ

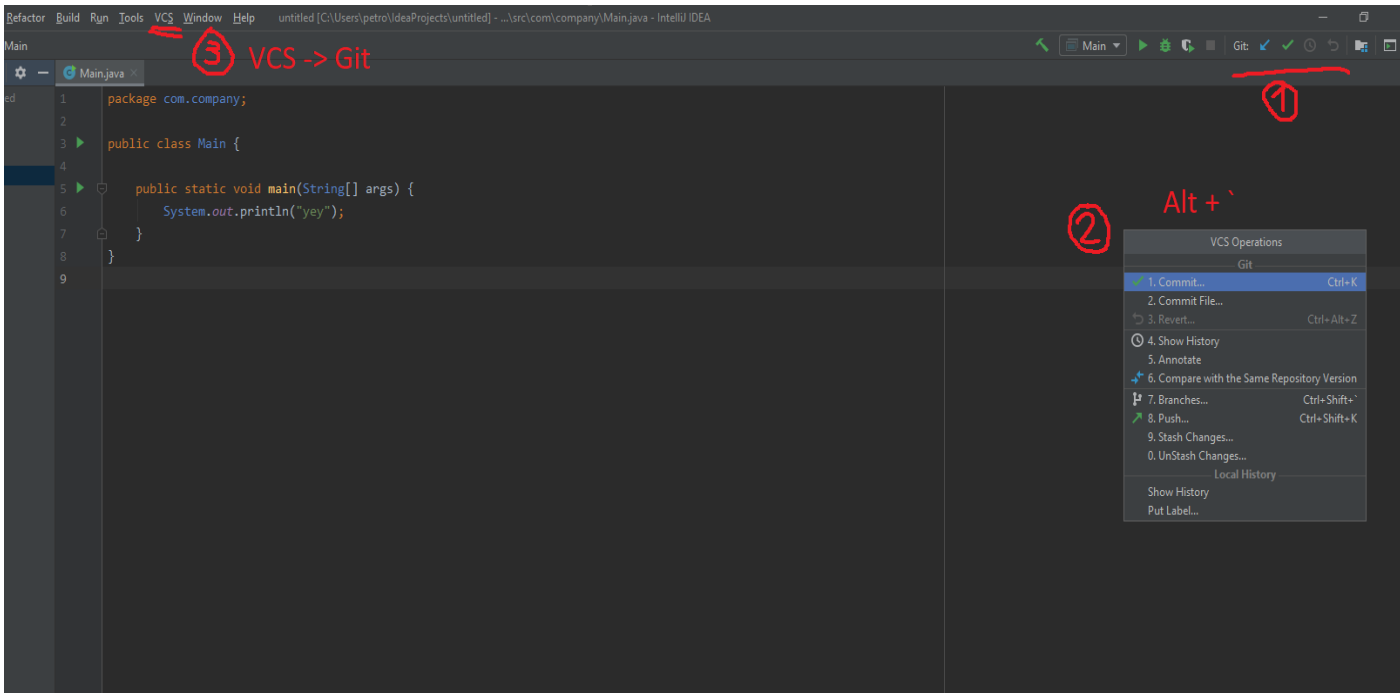
Το IntelliJ όπως και τα περισσότερα IDE έχουν built in την δυνατότητα χρήσης VCS με πάνω κάτω την ίδια διαδικασία και λειτουργίες με αυτές που περιγράψαμε λεπτομερώς πριν, οπότε καλό θα ήταν να δείτε και την προηγούμενη ενότητα.

-Αρχικό σετάρισμα

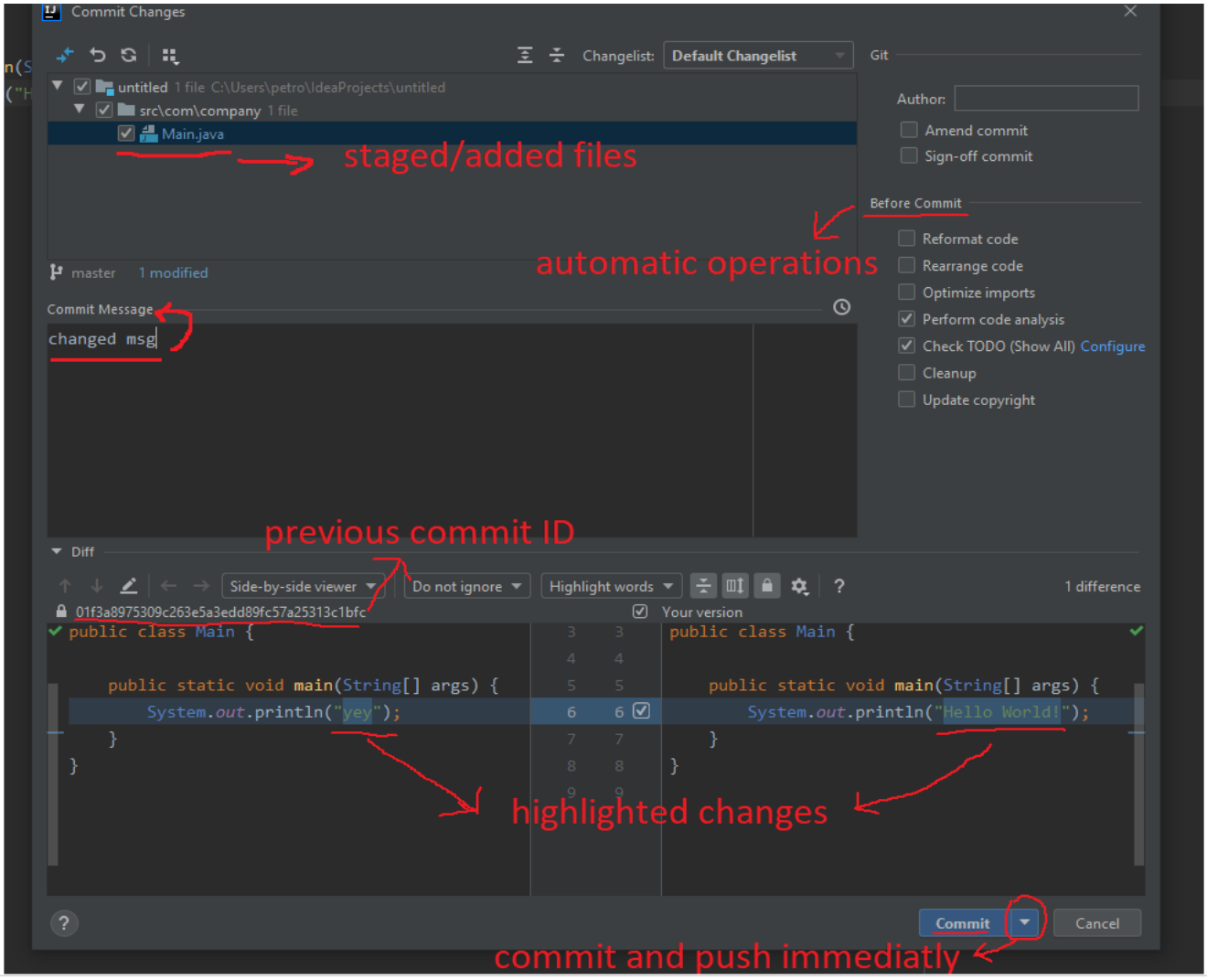
Δημιουργούμε ένα project(αν δεν έχουμε ήδη) και αφού το ανοίξουμε, από το μενού πάνω επιλέγουμε
VCS->Import into Version Control->Share project on GitHub. Μπορούμε να καθορίσουμε το access, το όνομα και το description του project αυτή τη στιγμή. Στην επόμενη οθόνη επιλέγουμε τα αρχεία που θέλουμε να ανεβάσουμε, καλύτερα να τα επιλέξουμε όλα για να έχουμε ένα κοινό configuration με τους συνεργάτες μας και ύστερα να βάλουμε το σχετικό .gitignore στο repo για να αγνοηθούν τα άχρηστα.
(δες εδώ: https://github.com/github/gitignore/blob/master/Global/JetBrains.gitignore ή εδώ)

-Οδηγίες χρήσης

Μπορούμε να χρησιμοποιήσουμε τα εργαλεία του Git με έναν από τους 3 τρόπους:

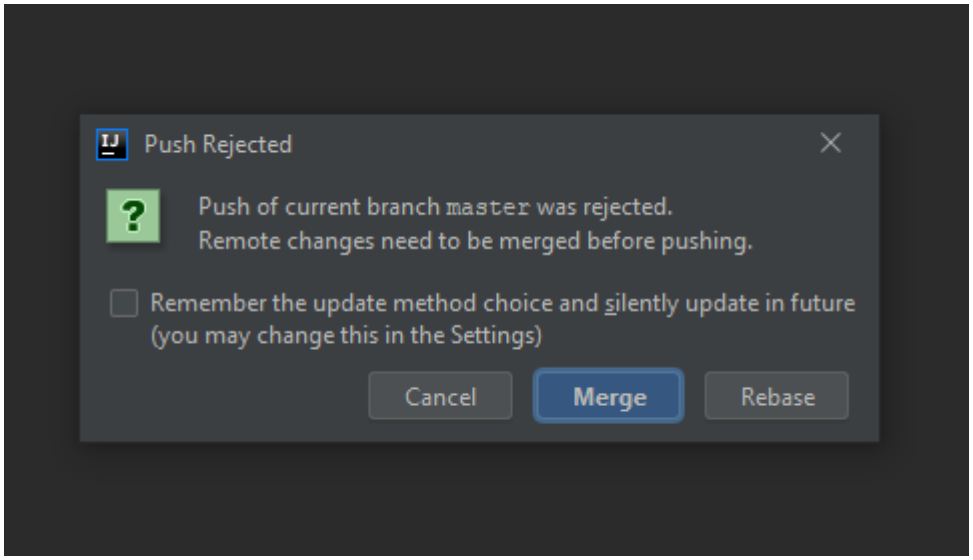


Το πράσινο check εικονίδιο συμβολίζει το commit



Άμα δε χρησιμοποιήσετε το commit and push operation από το βελάκι, θα βρείτε την επιλογή push στα μενού, ως το πράσινο βελάκι προς το πάνω. Το μπλε εικονίδιο με το βελάκι προς τα κάτω που αναφέρεται ως “update project”, είναι το pull operation.

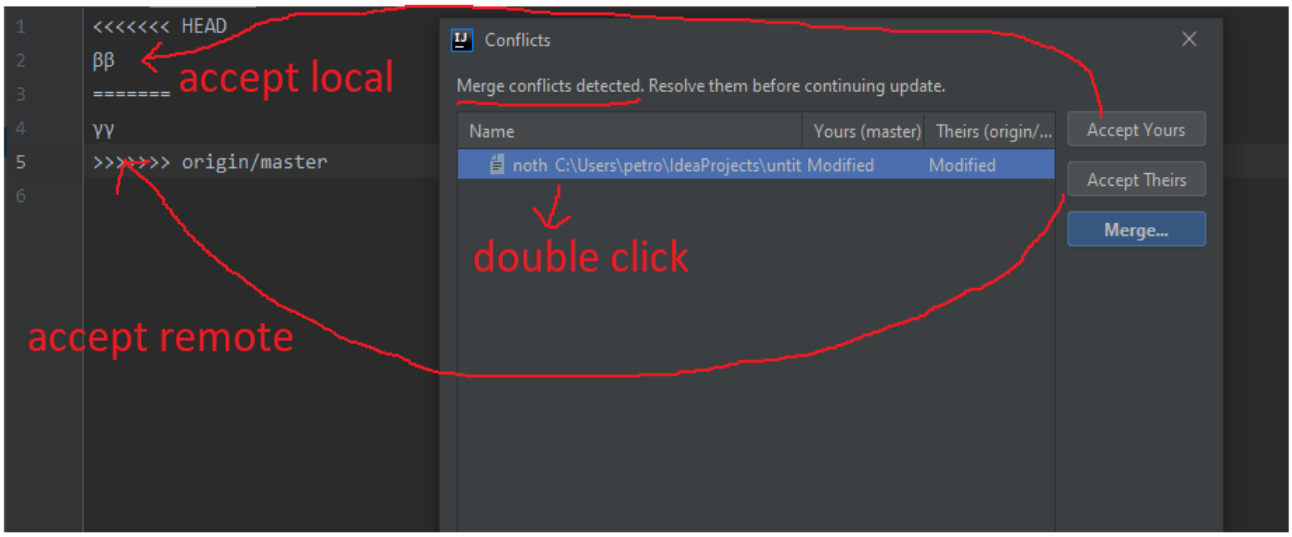
Οι περιπτώσεις των interaction μεταξύ των συνεργατών είναι ακριβώς ίδιες με πριν. Στην δεύτερη περίπτωση που πάμε να pushάρουμε κάτι στον server χωρίς να έχουμε την τελευταία έκδοση του project locally, έχουμε το ακόλουθο μήνυμα σφάλματος όπως είναι αναμενόμενο



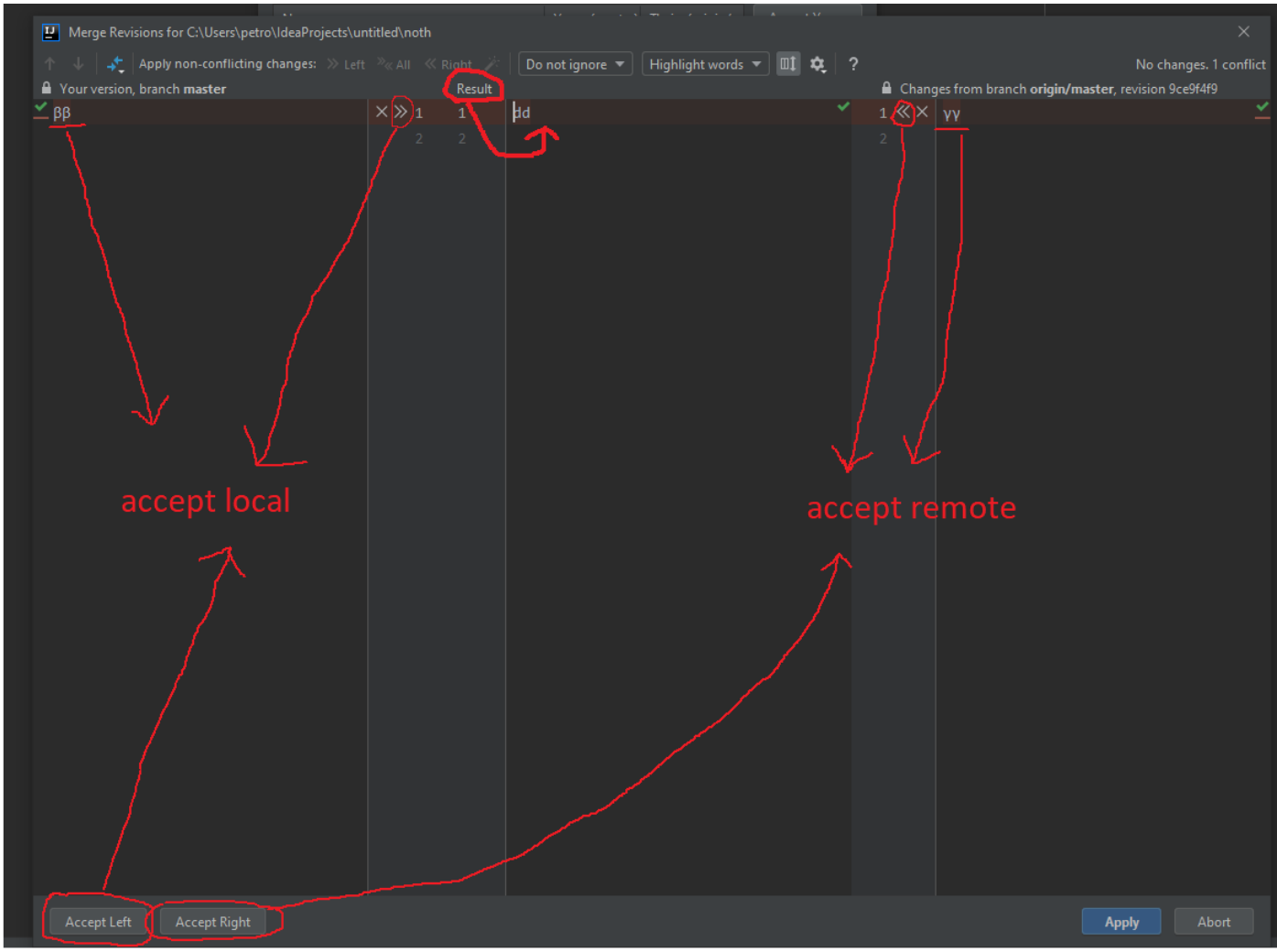
Πατώντας μόνο το merge, γίνεται αυτόματα το fetch->pull->push operation!

-Conflict Resolution

Ας εξετάσουμε και τον ελέφαντα στο δωμάτιο, εν ονόματι conflicts.
Έστω πάλι ότι 2 συνεργάτες πειράξανε την ίδια γραμμή σε ένα αρχείο με διαφορετικό τρόπο. Η διαδικασία είναι ίδια με την από πάνω μέχρι το σημείο του merge, το merge θα αποτύχει



Μπορώ στα γρήγορα από τα κουμπιά να αποδεχτώ την μία ή την άλλη αλλαγή, ή να συγκρίνω πιο διεξοδικά τα αρχεία, κάνοντας διπλό κλικ στο όνομα του αρχείου



Αφού τελειώσουμε με το resolution, αρκεί να ξανακάνουμε push.

Απ’ ότι φαίνεται, η χρήση του Git μέσα από το IntelliJ είναι αρκετά βολική και πιο άμεση. Επίσης, άμα πατήσουμε την επιλογή show history (γκρι ρολόι εικονίδιο) μπορούμε να έχουμε ένα visualization για τα commits του project και στο αριστερά tab από το μενού που μας άνοιξε, στην επιλογή “Log” έχουμε μια ακόμα πιο λεπτομερή περιγραφή, με τα αυτοματοποιημένα commits να συμπεριλαμβάνονται.