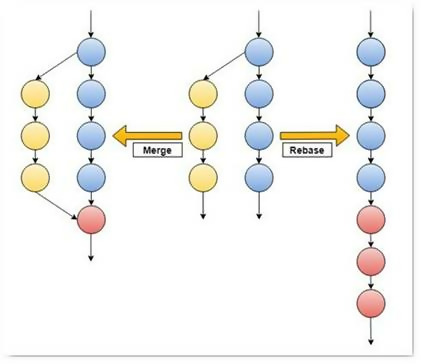
לאחר לא מעט מחשבה והתייעצויות החלטנו לסדר יותר את הצורה בה אנחנו עובדים בGit, אלו השינויים שחשבנו עליהם:

במקום להכניס את הקוד לbranch הראשי עם merge commit כמו שעשינו עכשיו אנחנו נכניס את התכולה עם rebase and fast-forward ו squash(אל דאגה זה הגדרות שאפשר לעשות בgit ככה שלא צריך להתעסק עם זה יותר מידי).

**למה זה יגרום?**

בגדול יהפוך את העץ של ההיסטוריה של הRepository  להרבה יותר פשוט ויפה ויגרום לזה שהתחקור של הcommitים אחורה יהיה הרבה יותר קל.

  אם נפרט יותר, הrebase  לעומת הmerge  נראה ככה: (תסתכלו ימינה  👉)



שימו לב את השינוי בעץ שנוצר, ב**rebase** הוא לוקח את הcommitים

ופשוט מעתיק אותם בעץ לאחרי הcommitים שהיו בbranch הראשי

לעומת זאת ב**merge**, רואים את היציאה של הברנאצ' והחזרה לבראנצ' הראשי, בעזרת יצירת commit נוסף: "Merge branch 'feat/some-feature' into 'dev'"

ומה הsquash עושה?

לוקח את כל הcommitים והופך אותם לקומיט אחד ויחיד

\*\* תמשיכו עדיין לעבוד עם קומיטים במהלך הפיתוח, זה נוח לחלק את העבודה,

וחשוב לגבות את העבודה בremote אחת לפרק זמן למקרה והמחשב יפסיק לעבוד

או שתרצו לעבוד מעמדה אחרת או עבודה משותפת עם מישהו וכדומה

***לאחר התהליך, לbranch הראשי יתווסף*** ***commit*** ***אחד שיכיל את הfeature שהכנסנו*** 😁

ואז ההיסטוריה של הbranch תהיה מאוד רזה ותכיל רק את התכולות שהכנסנו

בלי commitים מיותרים שלא נותנים שום מידע משמעותי לאחר שהתכולה נכנסה.

שינוי נוסף, החלטנו להפסיק להשתמש בברנאנצ'ים של dev וpreprod 😱. אני יודעת שזו הצהרה מפחידה, אז אני אסביר.

**מעכשיו נעבוד רק מול** **branch אחד, מול הmaster.**

אז איך זה יעבוד? איך נוריד (או נעלה, תלוי את מי שאולים 😅) גרסה לסביבות השונות?

וכאשר נרצה לעלות את השינויים לסביבה מסוימת, אנחנו ניצור **release branch** שאוטומטית יריץ בילד וקומפילציה של הקוד, וטסטים במידה וקיימים.

לאחר מכן, אנחנו נשנה למספר הגרסה התואם למספר release במקום הנכון (כל פרויקט ומה שמתאים לו), ניצור tag עם הגרסא והתאריך

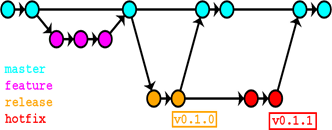
ולבסוף אנחנו פשוט נפעיל את הstage התואם לסביבה שאנחנו צריכים (סביבת בדיקות / סביבה מבצעית).

לאחר שהגרסה יורדת נכניס אותו חזרה למאסטר אבל בלי למחוק אותו.

עבודה עם release branch נותנת לנו לעקוב אחר הגרסאות שאנחנו מורידים לפרוד, ויתר מזה, מאפשר לנו להוריד דברים רק לחלק מהאתרים במקרה של בדיקת פיצ'ר מסוים אצל לקוח ספציפי.

סוג הברנאצ' הזה נשאר לאורך תקופה, ותמיד נוכל לעבור אליו במקרה הצורך.

ואם נרצה לראות נגיד איזה קוד נפרש לדוגמא בprod נראה מאיזה release נפרש לסביבה הספציפית פעם אחרונה, וכך נעשה אם נצטרך להכניס hotfix.

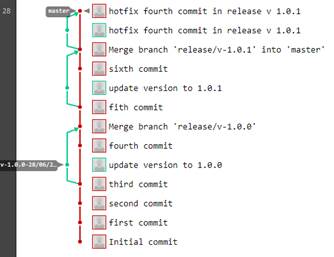
במקרה של hotfix, נעבור לברנצ של הrelease המתאים, נוציא ממנו בראנצ של hotfix ונמרג'ג' אותו חזרה לברנאצ הrelease,

נעלה את מספר הגרסא ב1 ופרוש את השינוי.

לבסוף, ננסה למרג'ג' את את הקוד המועדכן לmaster, אם לא ניתן (במקרה של קונפליקטים) נעשה לבראנצ' הזה פול מהמסטר,

נפתור קונפליקטים ונמרג'ג' שנית למאסטר, וכך השינוי יהיה אצל כולם **את כל השלבים יש לבצע ביחד עם רשצ או code owner**

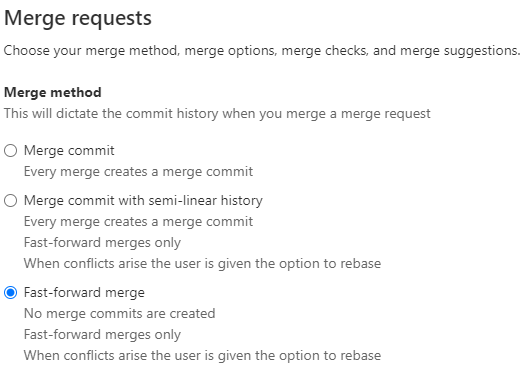
ככה זה בערך אמור להראות (היתה התעצלות מסוימת בקומיטים חחח)

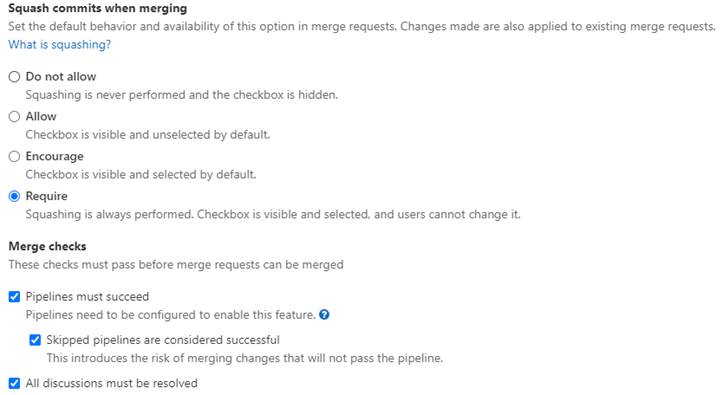


**אז מה זה דורש מכם?**

* לשים לב שבכל merge request / pull request שלכם ישנם רק שינויים הקשורים לנושא שלו,  
  בלי שינויים נוספים כמו תיקון באג קטן או סתם beautify על איזה קובץ, רק שינויים שקשורים לfeature או לbug.
* לוודא שכל הפרויקטים מעודכנים לעבודה עםrebase וsquash בMR
* אם אתם רוצים להתעדכן מול הmaster (רצוי) תעשו ממנו pull. ואם צריך גם תפתרו conflictים עם הmaster.
* אין צורך לעשות reset או squash או כל דרך "לעבוד על הגיט" לפני המרג עצמו, מספיק רק במרג'וג'
* יש לעדכן את הmaster לקוד הכי מעודכן לפני הפסקת העבודה עם שאר הבראנצ'ים
* יש לעדכן את הci/cd כך שיהיה אפשר לעשות deploy לכל סביבה בנפרד ללא תלות אחד בשני ורק מהrelease branches
* ויש לעבוד מאוד מסודר ובהתאמה עם כולם, אחרת סתם תהרסו אחד לשני

**שינויים שיש לבצע בהגדרות בגיט:**





**הסטנדרטים שיהיו:**

* שמות branchים- יהיה מורכב מprefix, מספר משימה (באג או פיצ'ר) ואז שם התכולה, (לדוגמא:  feat/4897-add-contact-mail):
  + פיצ'ר – feat
  + באג – fix
  + שינוי תשתיתי – chore
  + ברנאצ' לניסויים – junk
  + ריפקטור – ref
  + תיקון דחוף לסביבה המבצעית – hotfix
  + עבודה על ci/cd – ci
  + עיצוב בלבד (לא ברמה של פיצ'ר – style
  + גרסא – release/v-1.2.0
  + אם יש לכם רעיון לעוד סוג, דברו איתי 😉
  + שימו ❤ שם הבראנצ' צריך להיות באורך קצר ולא סיפור באורך 100 תווים
* הcommit של המרג' יראה כך:  
  feat(add contact mail) #5264 : add mail in profile page  
  add validation of email
* הcommitים שיהיו לכם בזמן עבודה, תשדלו שהם יהיו כמה שיותר מסודרים, עבורכם.
* **סטנדרטים של PR/MR** :
  + - כאשר התחלתם לעבוד על משימה, לאחר יצירת הבראנצ', יש לפתוח merge request או pull request כטויטא (WIP או draft) ורק כאשר סימתם אותם, להוריד מהמצב הזה, אחרי שכל הTestים עברו במידה ויש.
    - כותרת: מכילה את כותרת התכולה, מספר משימה
    - התיאור, יכיל:
      * הסבר על התכולה
      * הערות נוספות שהבודק צריך לדעת/מסמכים
      * קישור למשימה
      * תלויות: קישורים למרג'ים הקשורים למשימה (שצריך לבדוק ביחד)
      * תמונה או Gif של הנעשה בתכולה (אם אפשר)
    - הreviewer של המשימה:
      * בפתיחת המרג', צריך לקרוא את תיאורו, ואת המשימה ולוודא הבנה, אם יש צורך, יש לשאול את הPM.
      * בדיקה שהוולידציות של המרג' עברו, בין אם זה בילד מסוים או קונפליקטים וכו
      * לעבור לבראנצ' המשימה, (אם זה כמה מרג'ים יש לעבור בכל הפרויקטים המעורבים), לבדוק שהתכולה באמת התבצעה כמו שצריך, אם התבצע שינוי במשהו מרכזי/ תשתיתי, יש לוודא שהכל ממשיך לתפקד.
      * בדיקת הקוד:
        + אני אישית ממליצה לחשוב קודם איך את/ה היית ממש/ת את זה, כדי לבוא בראש נקי ולא להיות מקובעים לקוד הנבדק
        + בעת הבדיקה יש לשים לב שהקוד נמצא באיזון בין קריא ליעיל
        + לא לפחד להעיר, מותר לעלות כל דבר לדיון (מכבד כמובן)
        + יש להעיר על קוד שלא ברור למה נצא שם/ לא ברור או קריא / הערות או console log שנשאר, חלוקה לפונקציות או קומפוננטות לא נכונה וגם הצעה איך לעשות דברים בדרך אחרת
        + מומלץ להוסיף להערות אימוגים ולכתוב בדרך נחמדה, נא לא לקחת דברים אישית ועם אגו
        + זכרו! סקרי קוד נועדו לשפר את הסוקר וגם את הנסקר, ללמוד זה טוב 💪
    - לאחר מתן הערות, מתבצעים התיקונים או הדיונים
    - **רק מי שכתב את הערה עושה resolve להערה, אחרי שראה כי התיקון התבצע לפי כוונתו**
    - לאחר מכן, מתבצע אישור הPR רק על ידי רש"צ או באישורו, לא לשכוח למחוק את הבראנצ'!

יום קסום ✨