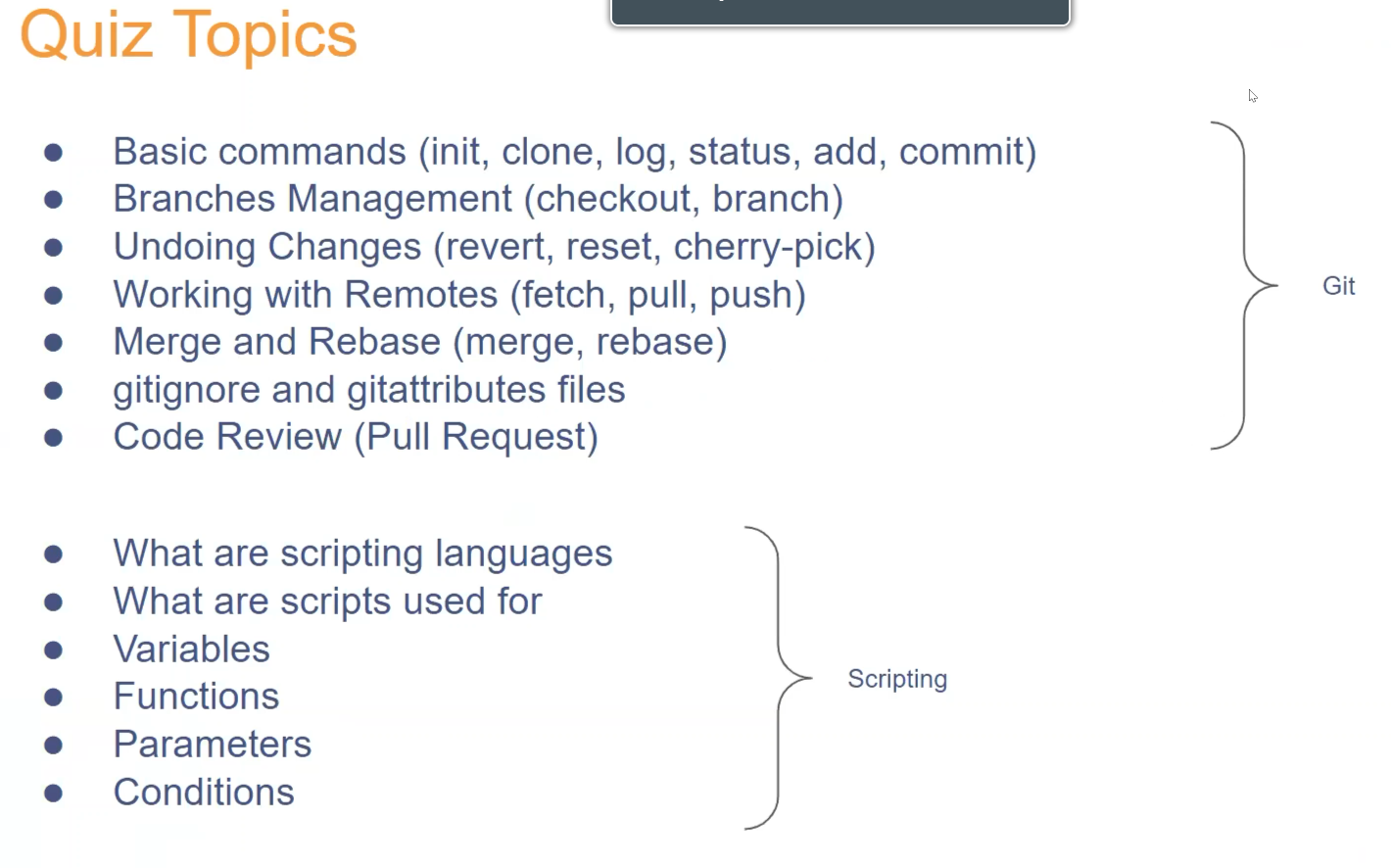
**DevOps קורס**

**Kickoff Meeting – 22/05/2022**

* **הסבר על השבוע השני** – דובר על השבוע שעבר , דובר על הפרויקט שיוגש השבוע , השבוע הפוקוס יהיה על **Source Control (Git)**
* **Quiz –** עומדים על ממוצע 90 😊 , ה**Quiz** יכלול את הנושאים הבאים :



**מפגש שלישי – 22.05.2022**

* **Version Control Systems** – מערכת לניהול רשומות ושינוים בקבצים שמתבצעים לאורך הדרך (בתיקיות , בפרויקט וכו) , לדוגמא **SVM , TFS , Git , Mercurian**.

בעבר עבדו על אותם קבצים ותיקיות ולכן היו מלא בעיות בנושא קונפליקטים בקוד וכו , המטרה היא להביא פיתרון לנושא הקונפליקטים וניהול גרסאות הקוד על מנת לייצר יישור קו בנושא הפיתוח הסופי , בנושא ניהול הסטוריית קבצים והקוד והצגת הבדלים בין הגרסאות והשינוים , כמובן גם גיבוים.

* **Distributed Version Control VS Centralized Version Control**

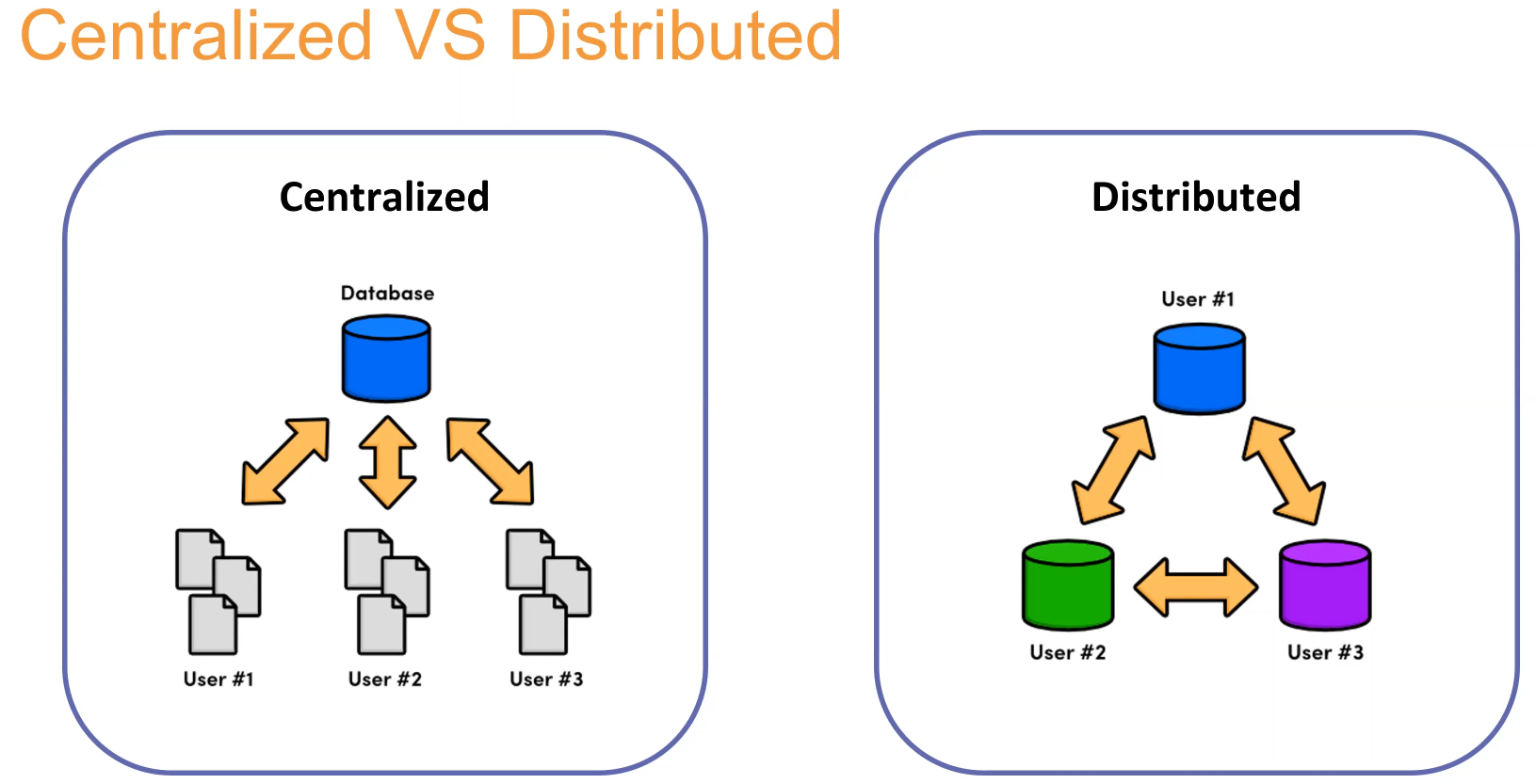
קיים מודל מסוג **Centralized Version Control** המאפשר פיתוח ועדכון שינוים מול שרת מרכזי אחד , כל מפתח מבצע שינוים בעמדת הקצה שלו (המחשב הפרטי שלו לצורך העניין) ולאחר מכן מבצע עדכון בשרת המרכזי (**Git Server**)

אך לעומת זאת הדבר מהווה קושי בנושא עבור מי שעובד בצורה מנותקת רשת (קיים בארגונים מסוימים)

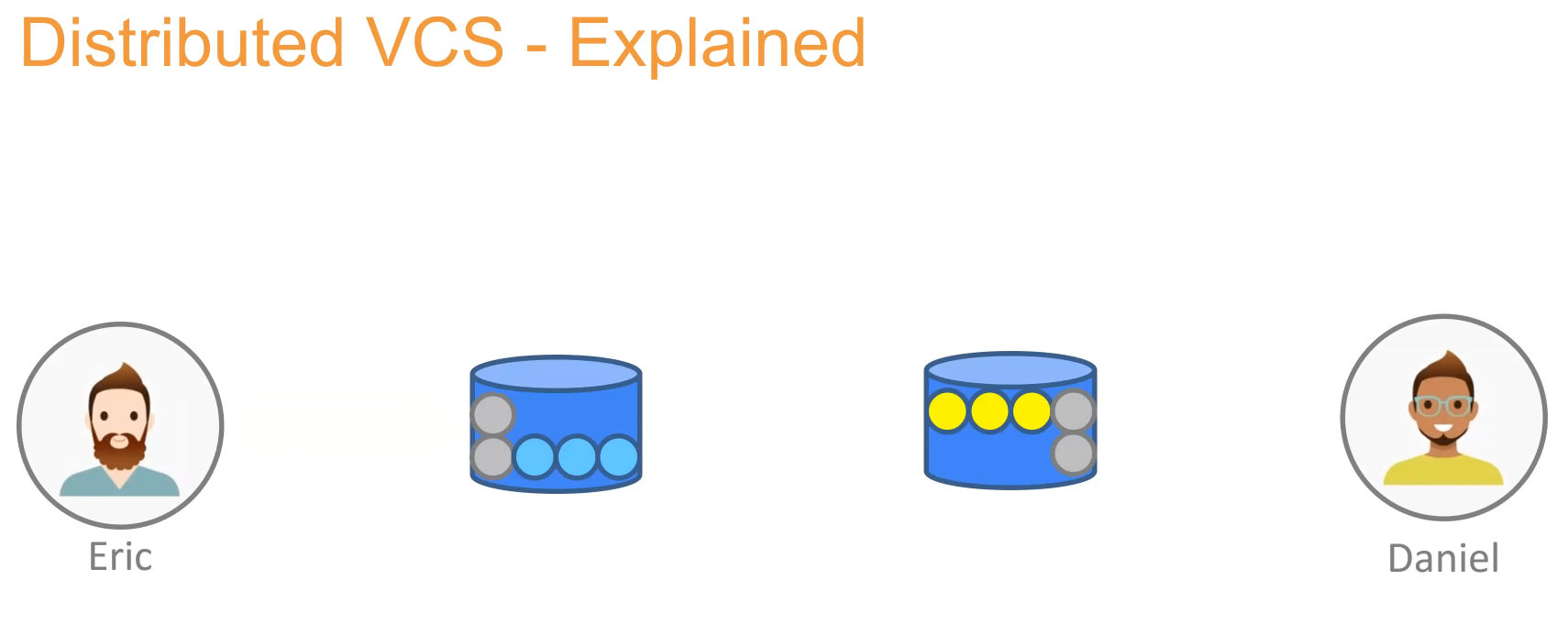
כל ההסטוריה של הקבצים והעבודה היתה ממורכזת במקום אחד – **השרת**.

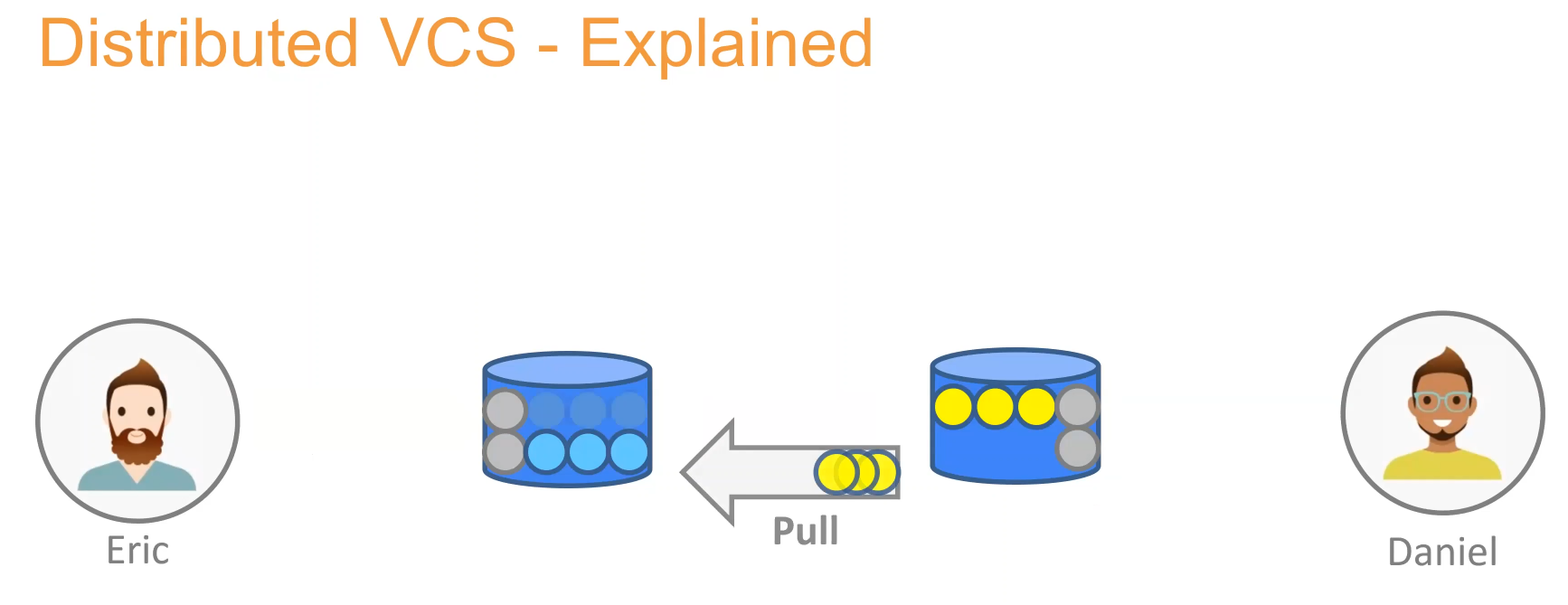
לעומת זאת , במודל **Distributed Version Control** כל לקוח קצה (מפתח בחברה) מוגדר כ"שרת" והדבר מאפשר לכל מפתח למשוך קוד (**Pull**) ממפתח אחר בחברה , לבצע שינויים על אותו הקוד ולאחר מכן לבצע עדכון (**Commit , Push**) על הקוד לאותו מפתח באופן חוזר.

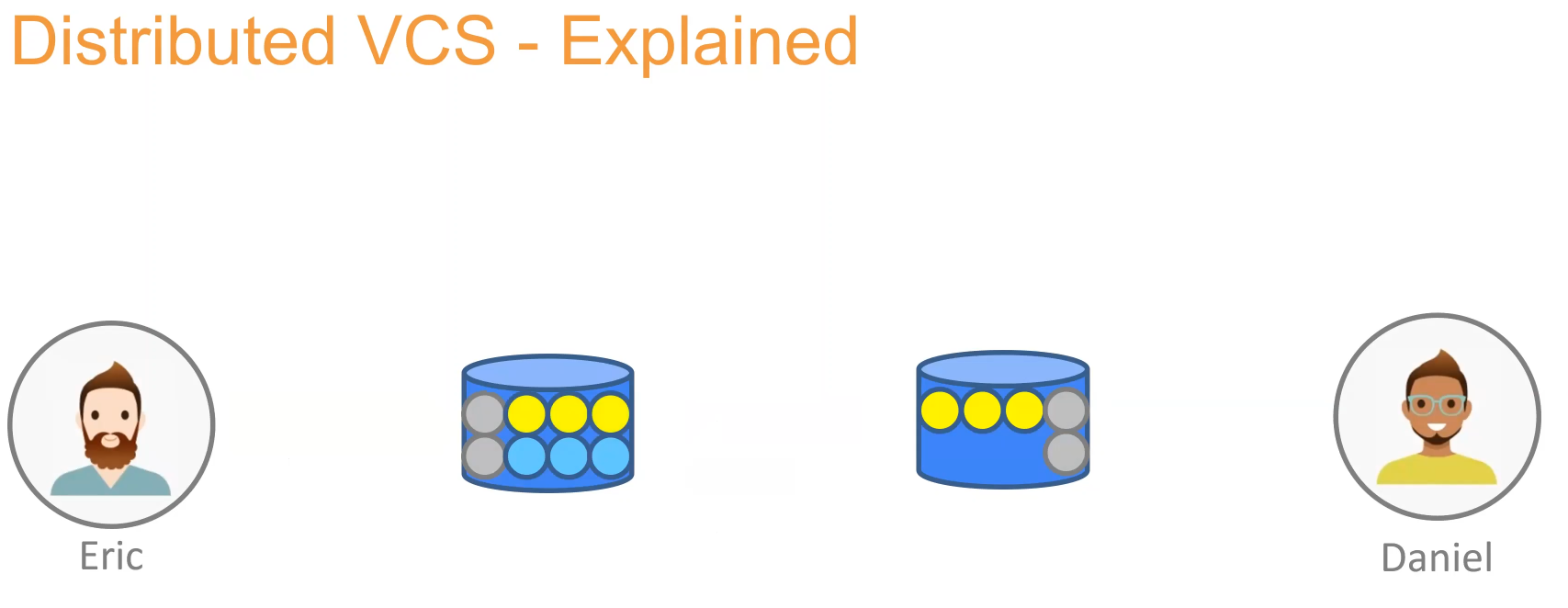
הדבר פחות מקובל כיום היות וכולם עובדים מול שרת מרכזי אחד , לכל מקרה **Git** מאפשר עבודה בתצורה שכזו.

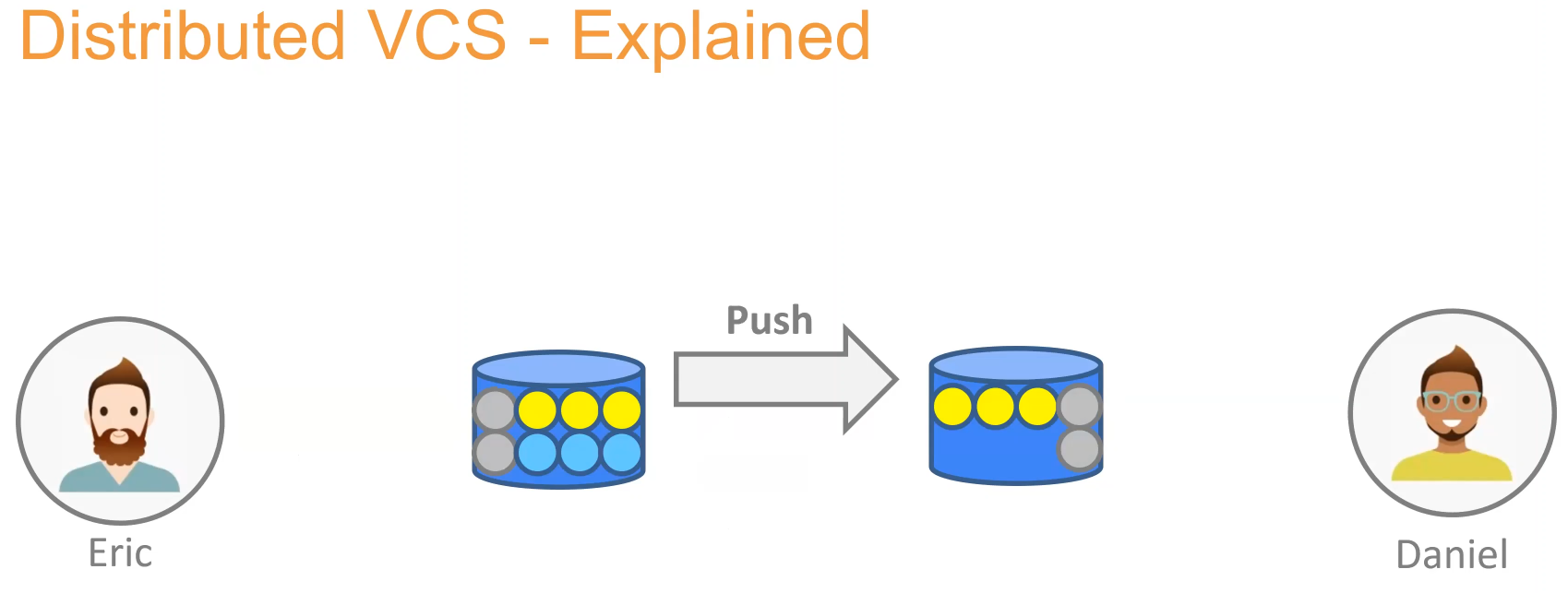


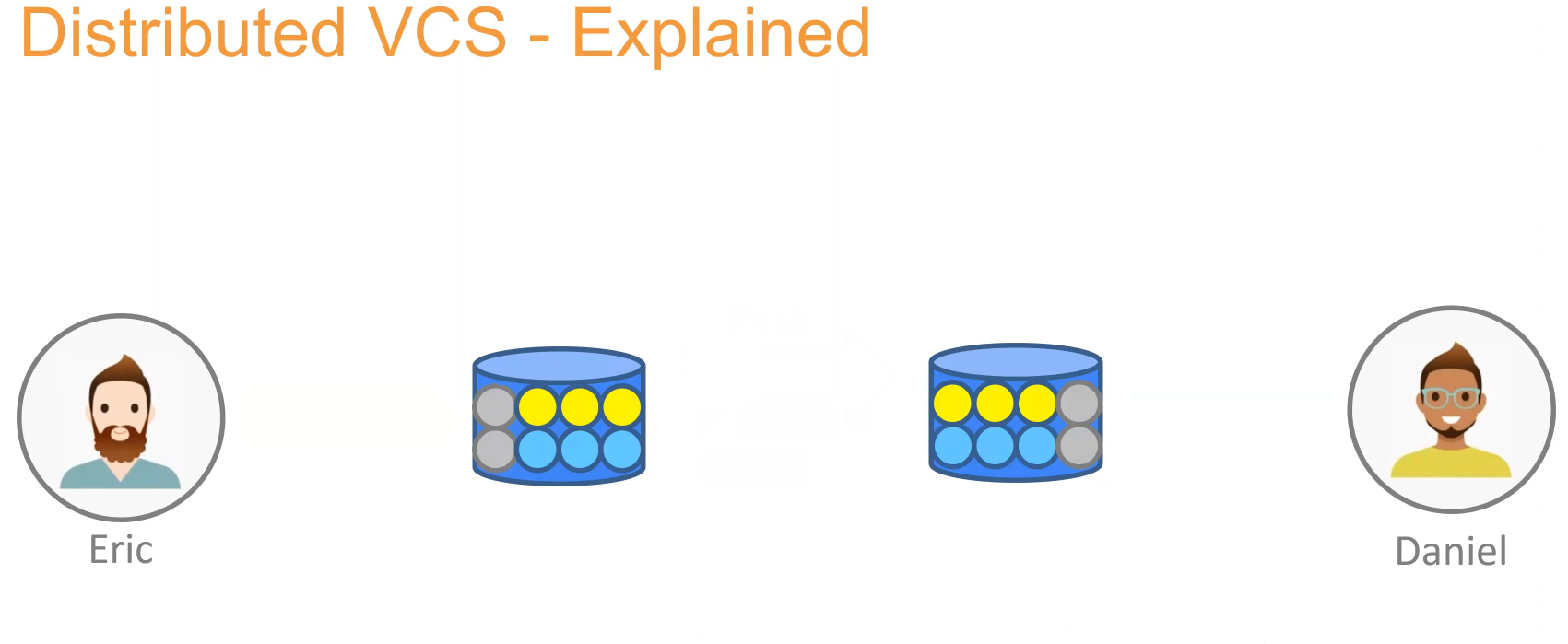
דוגמא למודל **Distributed**

****

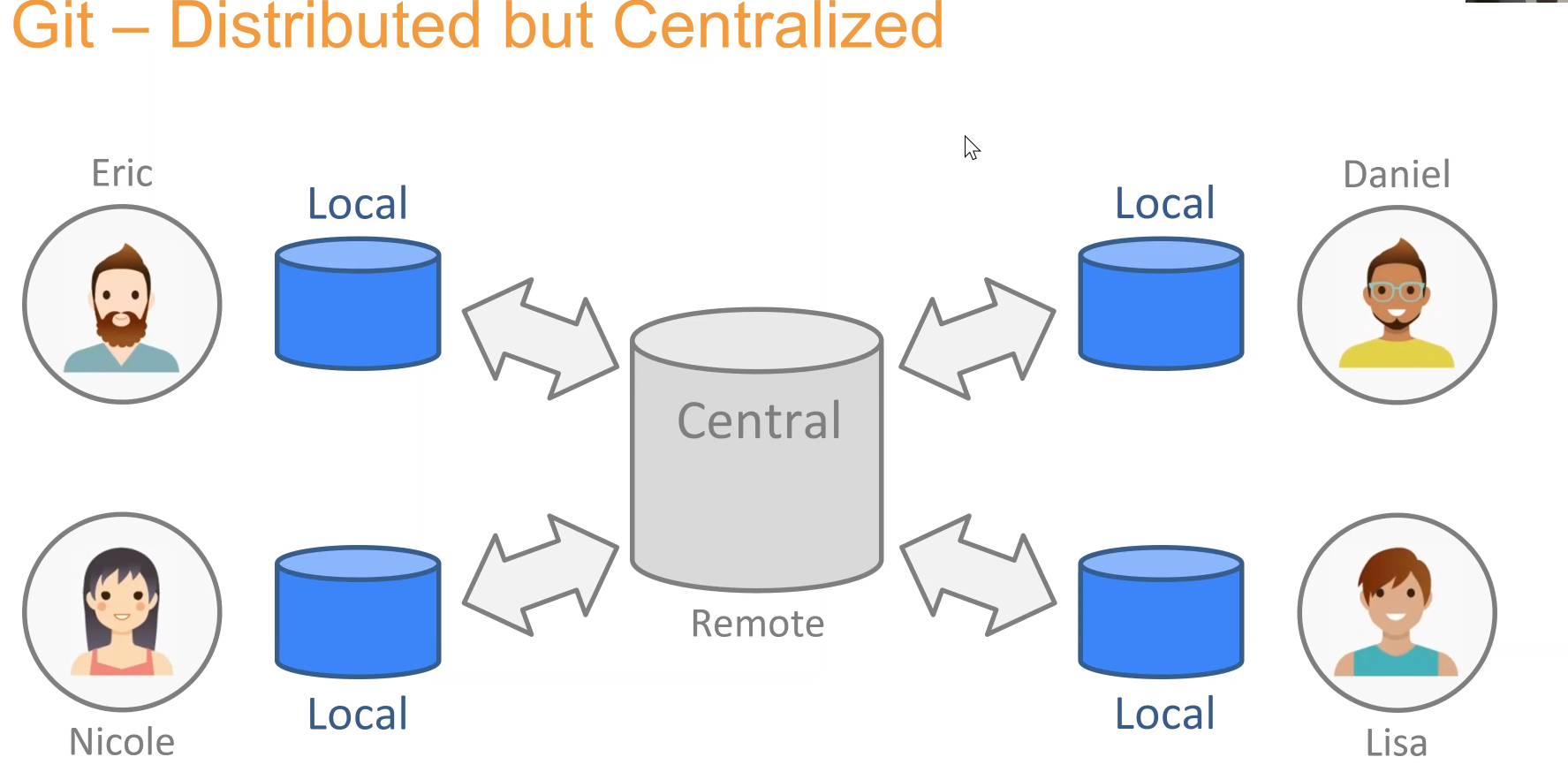
****

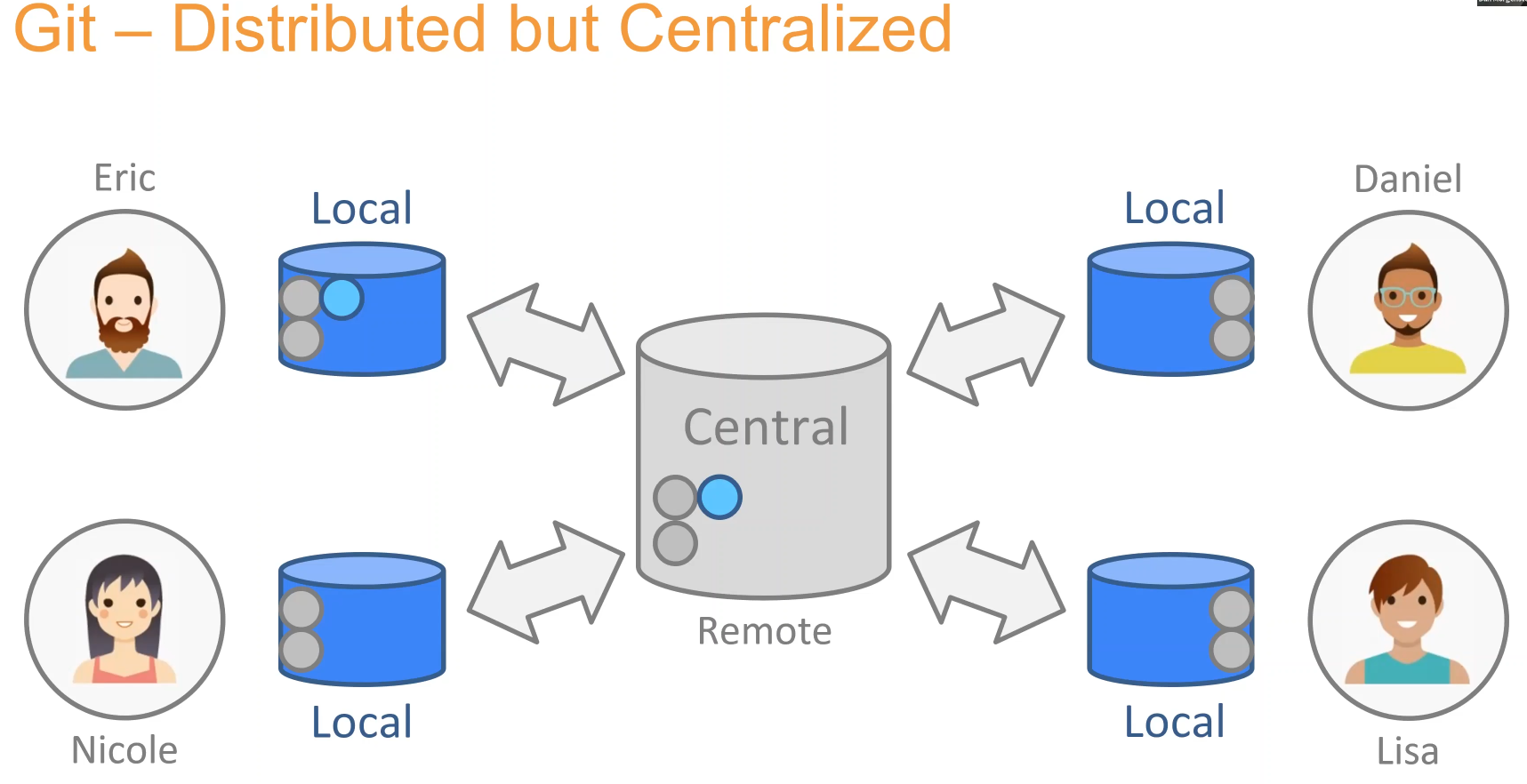
****

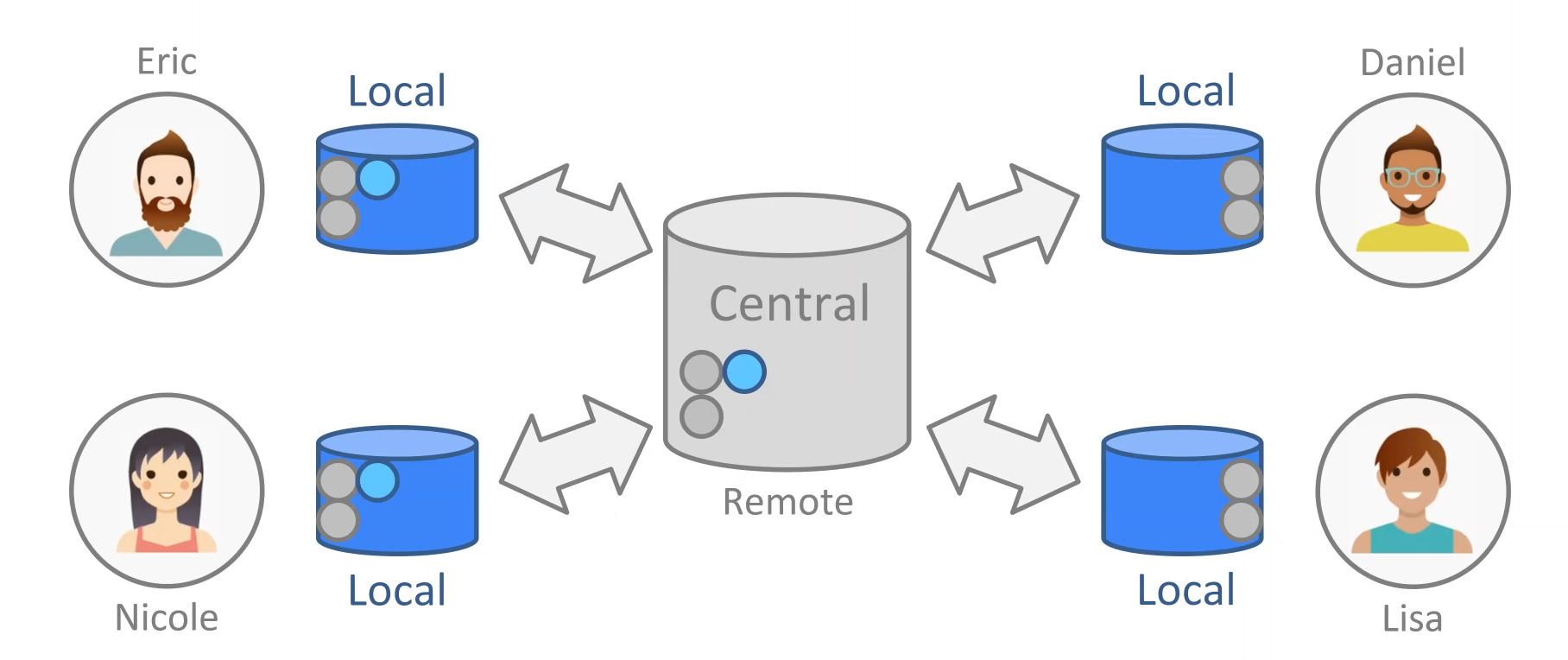
****

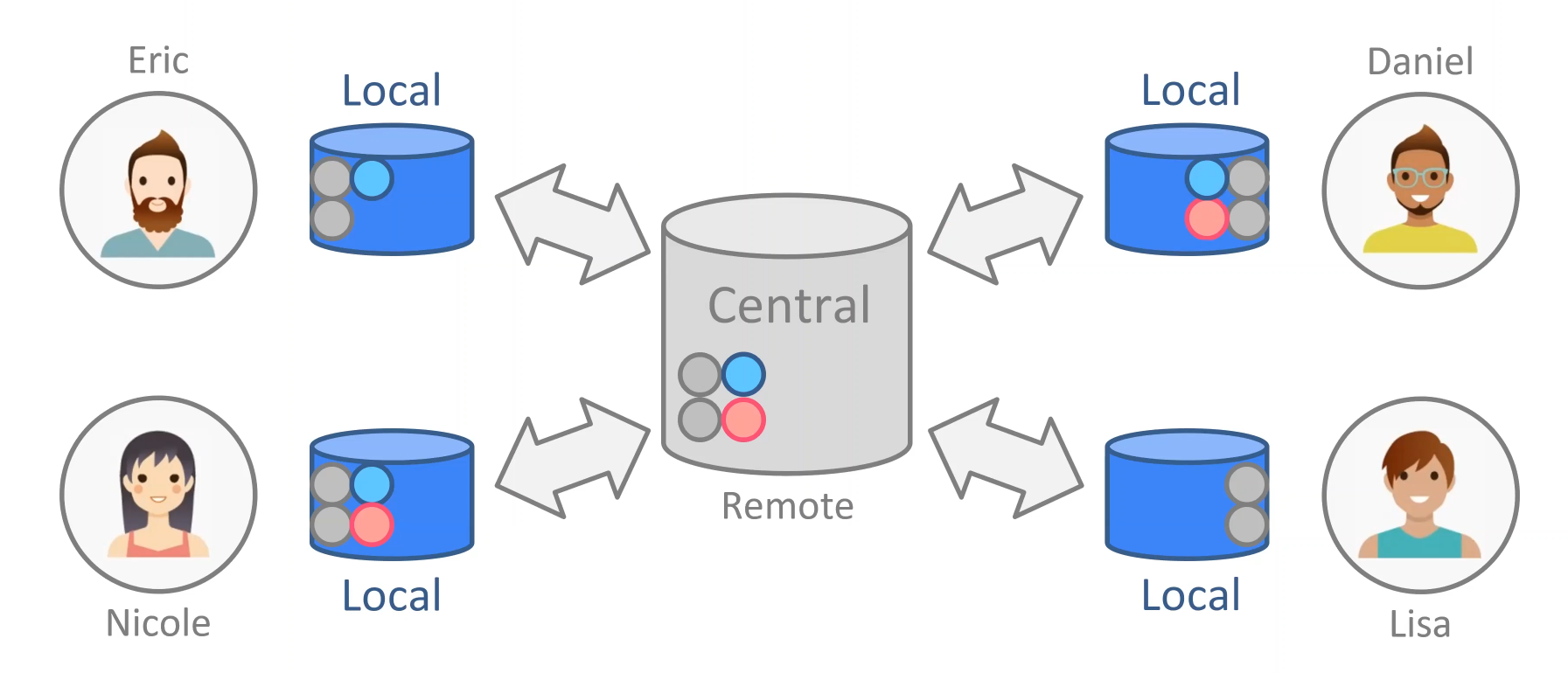
****

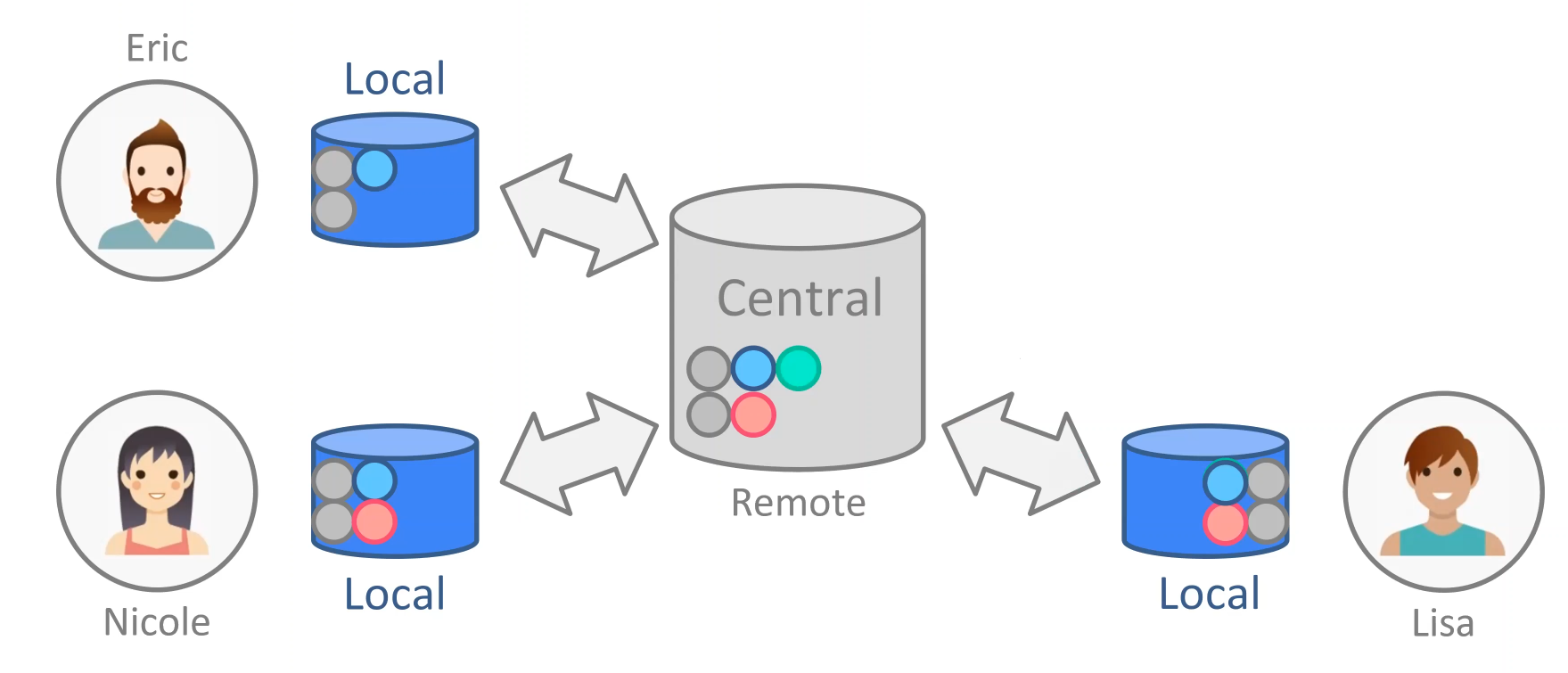
דוגמא למודל **Centralized**

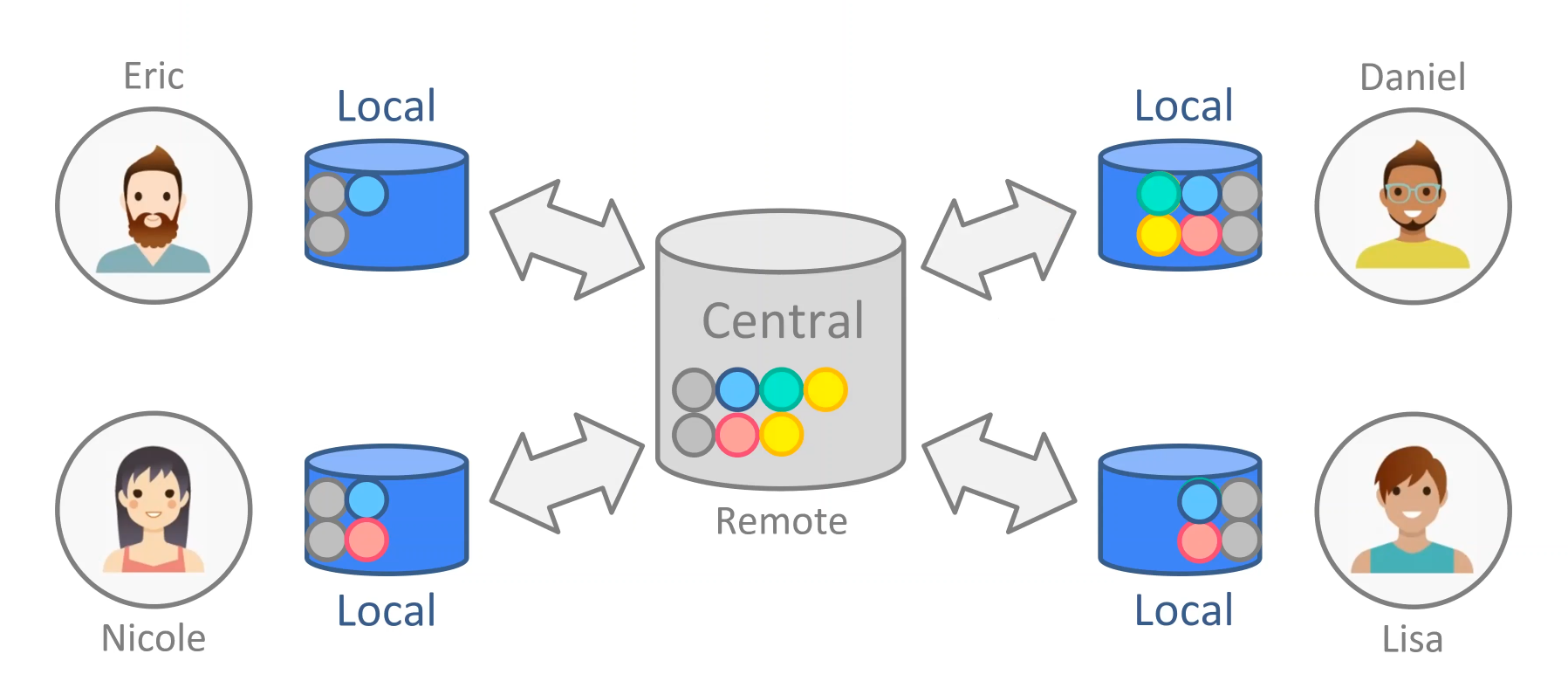
****

****

****

****

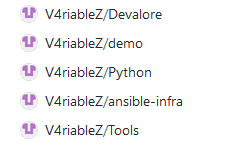




* שרת **Git** – שרת **Git** הינו שרת שתפקידו הוא אחסון גרסאות קוד אשר המפתחים מעדכנים (**Push**) ומושכים (**Pull** או **Fetch**) ממנו גרסאות , שרת **Git** מאפשר גם לראות הסטוריית קוד ועדכונים קודמים (**Git log…**) וכמובן מציג את ה**Branch**ים של ה**Repository** אותו המפתחים מעדכנים (**Git branch** יציג את הבראנצ'ים , **Git checkout -b “branch name”** יעביר אותנו על הבראנצ')
* **Central Repository** – כאשר אנו מבצעים תהליך **Clone** מ**Repository** קיים , אנחנו מקימים **Repository** מקומי שמכיל רק את הקבצים , ללא הסטוריית קבצים (**Commit**ים וכו)

הדבר תורם לצורך סנכרון מול ה**Repo** , נוכל לקחת לדוגמא את הפרויקט בשבוע הראשון , ביצענו **Clone** לכתובת שסופקה לנו ובכך התעדכנו מול הגרסא האחרונה של המוצר שפותח , לא יכלנו לראות גרסאות קודמות של הקוד אלא רק להוריד אותו אלינו.

* **Repository Managers** – מאין שירותים לניהול ואחסון של יותר מ – **Repository** אחד (יותר ממוצר אחד לדוגמא) , להלן דוגמא מה**Github** שלי

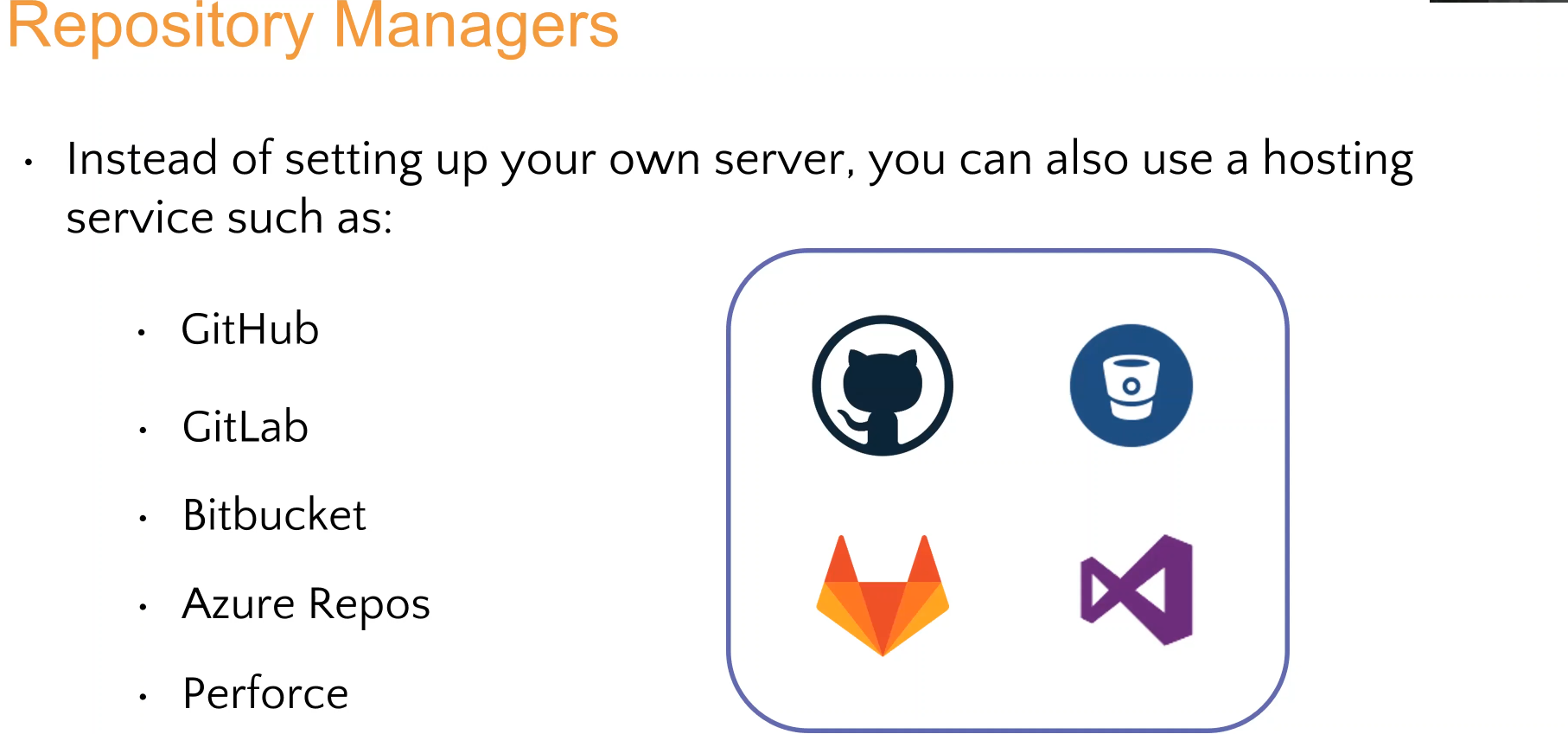


בתוך כל **Repository** (לדוגמא **Python** או **ansible-infra** , קיימים **master branch** וגם **branch**ים נוספים לאותו **Repository**.

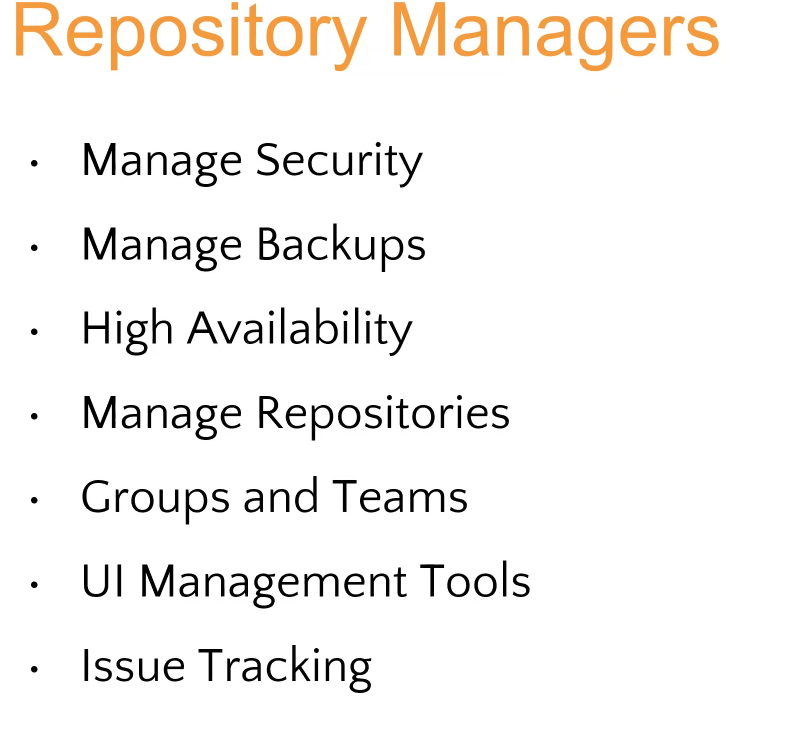
**אין קשר בין ה-Repositoryים המוצגים כלל , כל Repository הוא עצמאי לכל דבר**

במקרה המוצג ה**Repository**ים אינם חלק מ**Pipeline** קיים.

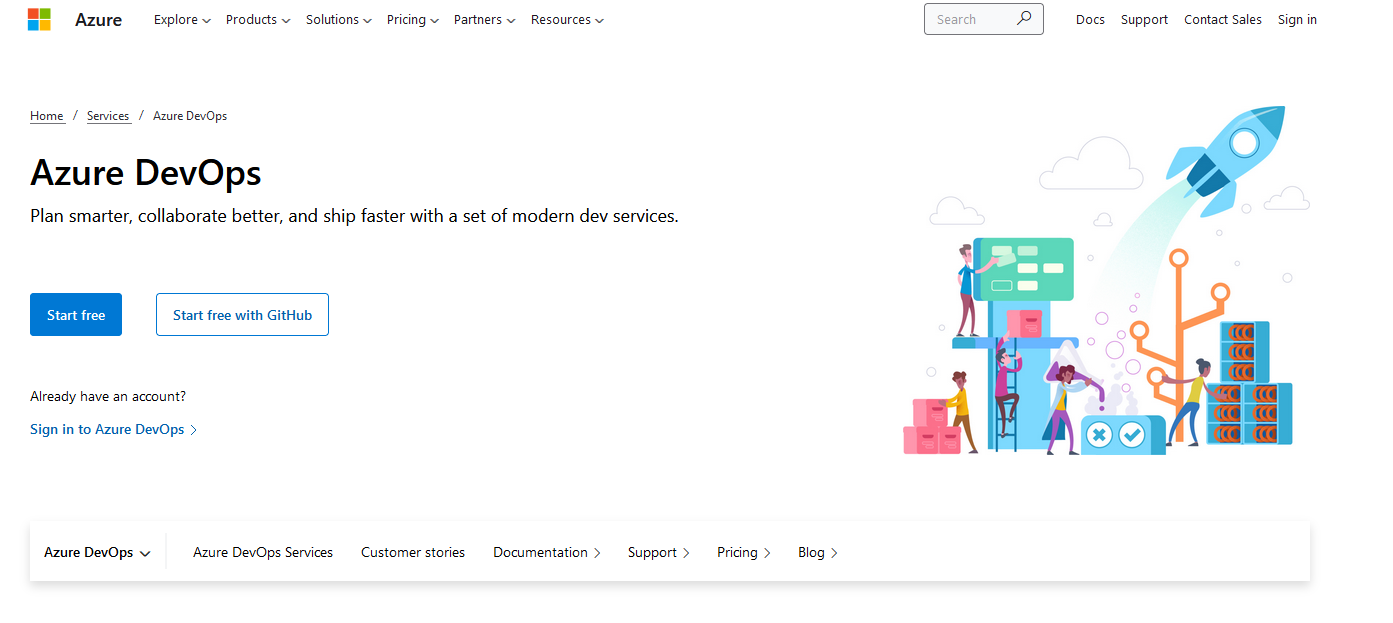
להלן שירותים נוספים לניהול **Repository**ים , ניתן להשתמש בהם כניהול בענן וגם במצב **Standalone** המקובל כיום בחברות הייטק.



להלן חלק מהשירותים הנכללים בתוך שירותי ניהול ה**Repository**ים

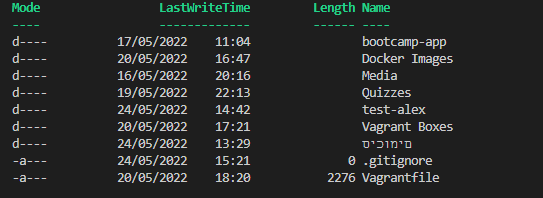


* ניתן להגדיר מי יוכל לגשת לאיזה **Repo** , מי יוכל לראות **Repo** ספציפי וכו...
* מתבצעים גיבוים של גרסאות הקוד והמוצרים באופן שוטף (במקרה הספציפי , בענן)
* השירות זמין בכל מקום ובכל זמן (שוב , בענן במקרה זה)
* ניתן להגדיר קבוצות פיתוח וקבוצות אבטחה (**Scrum master** וכו)
* מתאפשר מעקב אחר תקלות ובאגים בקוד (פתיחת באג \ **Issue** , דוגמא למצב כזה - <https://github.com/microsoft/vscode/issues> )
* ניתן גם לבצע מעקב אחר **Code Review** , הדבר מאפשר בדיקת קוד טרם העברתו אל ה **master branch**
* **Azure Devops** – לינק לשירות - <https://azure.microsoft.com/en-us/services/devops/>



השירות מאפשר שירותי ניהול גרסאות קוד (**Git Repository** ולא **Github**) , שירותי בדיקות (**Tests**) ובניית תהליכי פיתוח (**Pipeline**) ועוד...

* **.gitignore** – קובץ הנמצא בדרך כלל בתוך תיקיית האב של המוצר (לדוגמא עץ של תיקיות וקובץ **.gitignore**)



מטרת הקובץ הנה לאפשר מצב של קבצים לדוגמא לטובת בדיקות , לדוגמא **myfile1.txt** וכו , כדי לעבוד עם הקובץ – יש לרשום את שמותיהם המלאים של הקבצים בתוך הקובץ **.gitignore** , לדוגמא **myfile1.txt , myfile2.txt** בשורה חדשה כל קובץ ולאחרמכן כאשר מתבצע תהליך **push** אל ה**Repository** או לתוך ה**Branch** הייעודי , **Git** יתעלם מהקבצים אשר נרשמו בתוך קובץ ה**.gitignore** (במידה ונוצר על ידי המשתמש) ויעדכן רק את הקבצים בתוך התיקיה (הקבצים אשר יתעדכנו כוללים את הקבצים בתיקיית האב + תיקיות וקבצים מטה)

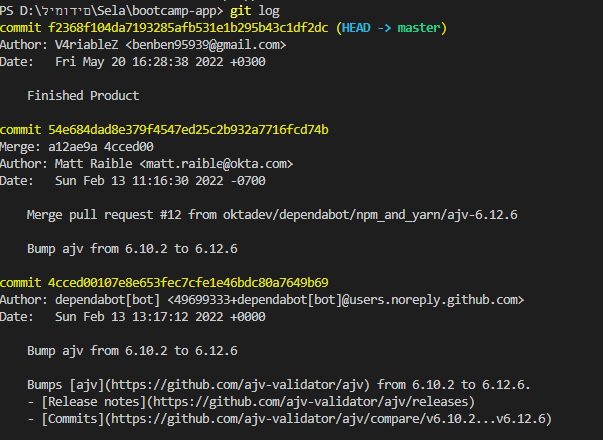
ניתן גם לבצע **ignore** לקבצים עם סיומות מיוחדים , כמו **.ps1** או **.exe** וכו.

* **היסטוריית Commitים ב-Git** – כל עדכון שאנו מבצעים ולאחר מכן מעדכנים בשרת המרכזי או במחשב המקומי (**Commit**) אנחנו בעצם מאפשרים מצב של הצבעה על העדכון הקודם.

לדוגמא , עדכנתי או יצרתי קובץ ולאחר מכן ביצעתי את הפעולות **Git add , Git commit**

אזי , באפשרותי לחזור לגרסא הקודמת של העדכון על ידי שימוש בפקודה **Git log** שבה מוצגים כלל ה**Commit**ים של אותו קובץ (במידה ויש יותר מ1).

לדוגמא :



ניתן לראות כי קיימים מספר **Commit**ים , בכל **Commit** בוצע שינוי אחר , לדוגמא **Bump ajv from 6.10.2 to 6.12.6** או לחליפין **Merge pull request**.

ניתן לחזור אחורה בזמן באופן תאורתי , ניתן לחזור **Commit** אחד אחורה במידה ואנו מעוניינים לבדוק עדכון או תהליך שבוצע טרם עדכון ה **Commit** האחרון בעזרת הפקודה **git checkout** ולאחר מכן את קוד ה**Commit** , לדוגמא **git checkout 54e684dad ,** יחזיר אותי אל העדכון הקודם טרם העדכון האחרון שבוצע.

**יש לשים לב כי הסימון HEAD -> master** או שם ה**Branch** הרלוונטי , יוחלף כאשר אנו נחזור לעדכון הקודם , היות ואנו בודקים גרסא שהיא אינה הגרסא האחרונה של הקוד , אם נבצע שינוים נוספים ונרצה לעדכן את השרת המרכזי (**Git server**) עלינו קודם כל לבצע את הפקודה **git pull** ולאחר מכן , כאשר יוצגו לנו קונפליקטים (התנגשויות קוד וקבצים וכו , **If any**) עלינו לפתור אותם ורק לאחר מכן נוכל לעדכן את השרת המרכזי.

**יש לשים לב כי לאחר ביצוע עדכון בשרת המרכזי נקבל שגיאה שאנו נמצאים גרסא אחת מעל הHEAD , הסבר למצב זה יוצג בשלב מאוחר יותר בסיכום.**

במידה ורצינו לבדוק **commit** מסוים אך לא ביצענו שום עדכון ואנו מעוניינים לחזור אל ה**commit** האחרון , נוכל לבצע זאת על ידי הפקודה **git checkout f2368f104** (**כמובן שהכל רלוונטי עבור האיור המוצג במסמך זה בלבד**)

**כל הCommitים נשמרים בתוך הנתיב .git הנסתר בתיקיית הRepo טרם העלאתם לשרת או הפעלה מחדש של מערכת ההפעלה.**