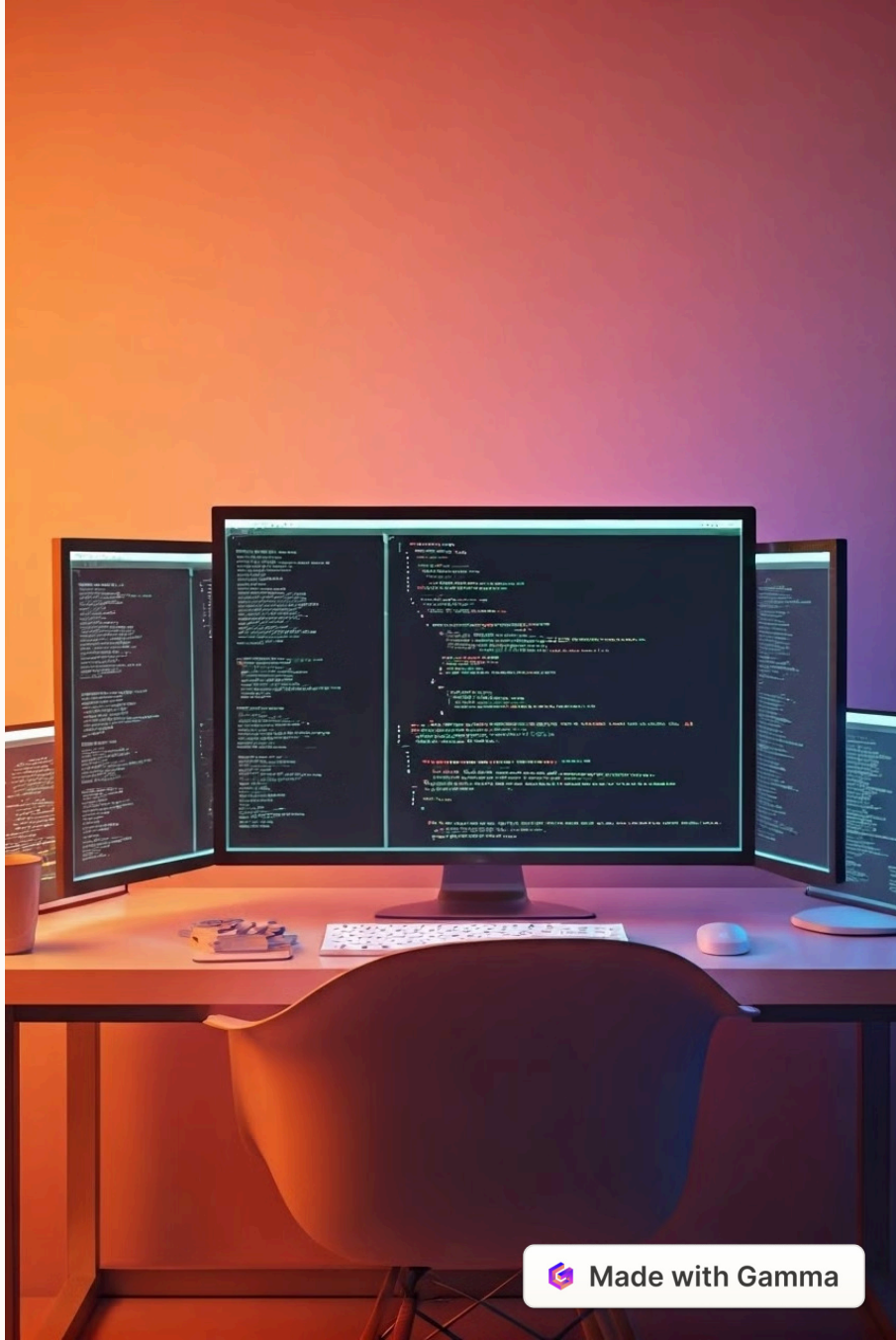
מבוא ל-Git: מערכת בקרת גרסאות לפיתוח תוכנה

ברוכים הבאים למצגת על Git, כלי חיוני לכל מפתח תוכנה. במצגת זו, נסקור את היסודות של Git, מדוע הוא חשוב, וכיצד להשתמש בו בצורה יעילה. בין אם אתם מתחילים או מפתחים מנוסים, מצגת זו תספק לכם את הידע הנדרש לשימוש ב-Git בפרויקטים שלכם.

by David Bar Or



מדוע חשוב להשתמש ב-Git בפיתוח תוכנה?

גיבוי ואבטחה

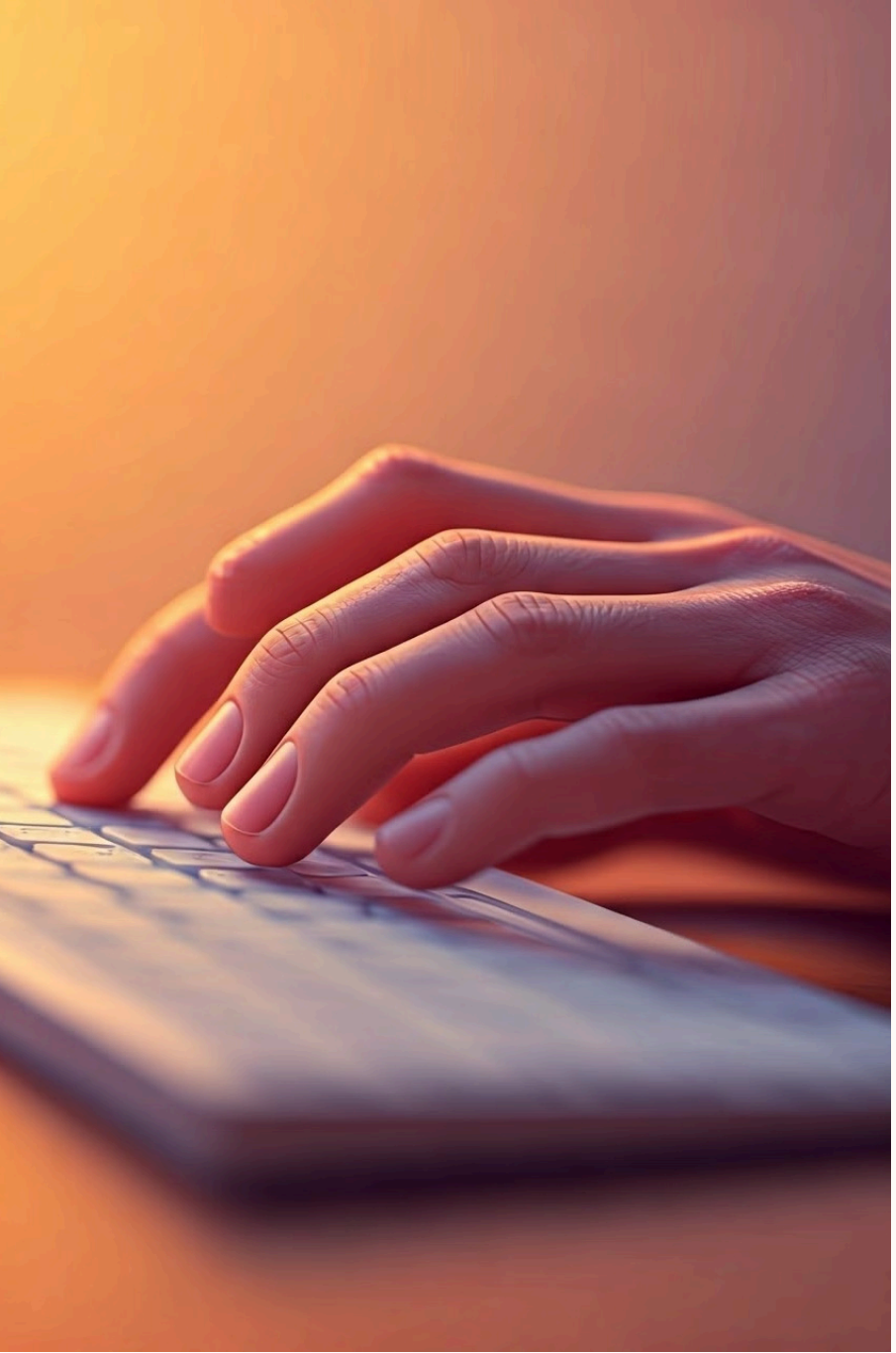
מספק גיבוי לקוד ומגן מפני אובדן נתונים, עם Git אפשרות לשחזור גרסאות במקרה של תקלות.

שיתוף פעולה

תומך בשיתוף פעולה בין מפתחים, מאפשר Git עבודה מקבילית ומיזוג שינויים בצורה מסודרת.

ניהול גרסאות

מאפשר ניהול גרסאות קל ויעיל, מאפשר Git חזרה לגרסאות קודמות ושמירה על היסטוריה מלאה של הפרויקט.



עבודה בסיסית Git: init, add, commit, push, pull

1

git init

יצירת מאגר Git חדש.

2

git add

הוספת שינויים לאזור ההמתנה.

3

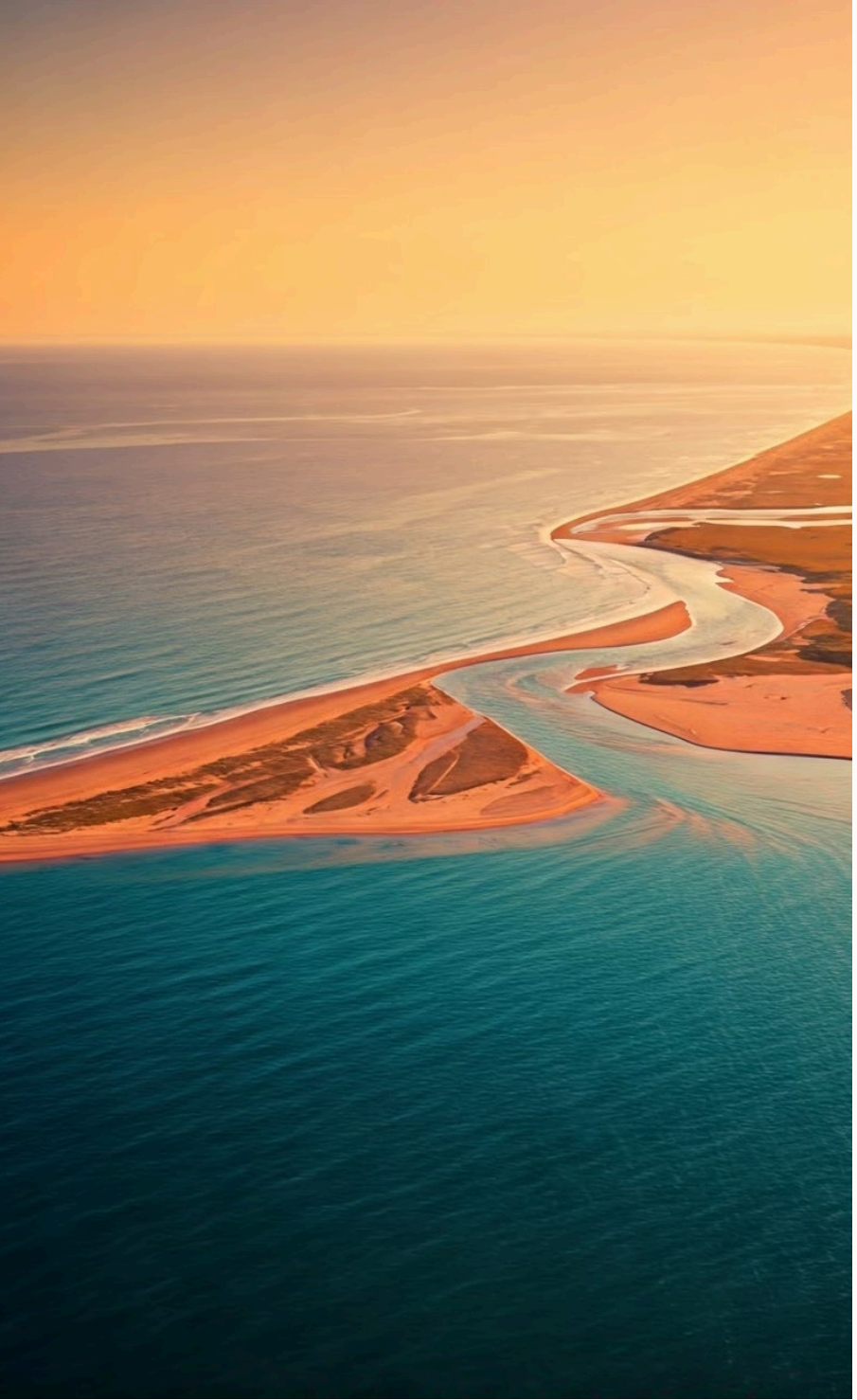
git commit

שמירת השינויים במאגר המקומי.

4

git push

העלאת השינויים למאגר מרוחק.



מושגי יסוד: מאגר, שלוחה, סניף, מיזוג

מאגר (Repository)

אוסף של קבצים והיסטוריה של שינויים.

שלוחה (Remote)

מאגר מרוחק שאליו ניתן לדחוף ולמשוך שינויים.

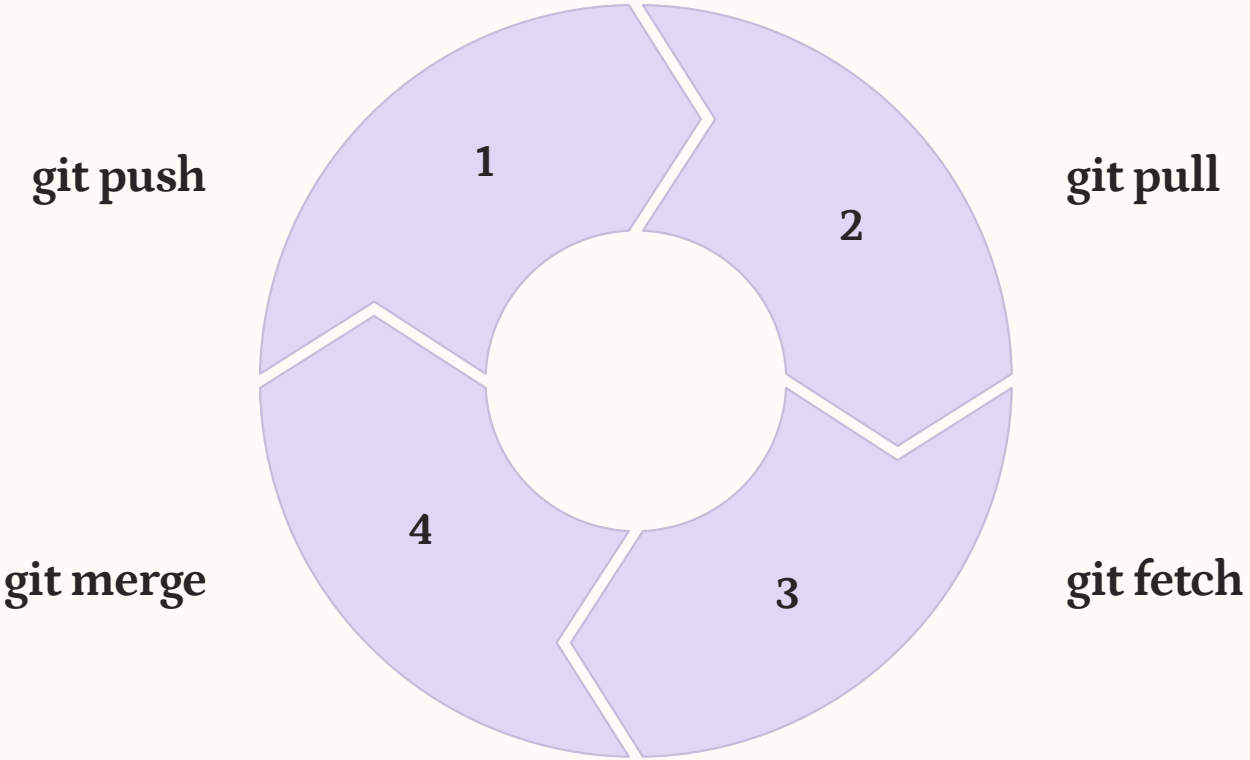
ענף (Branch)

גרסה נפרדת של הקוד, מאפשרת עבודה מקבילית.

מיזוג (Merge)

איחוד שינויים מסניפים שונים לסניף אחד.

עבודה מרובת משתמשים: push, pull, fetch, merge



בעבודה מרובת משתמשים, חשוב להבין את ההבדל בין **push** (דחיפת שינויים), **pull** (משיכת שינויים), **fetch** (אחזור שינויים) ו-**merge** (מיזוג שינויים). השתמשו ב-**push** כדי להעלות את השינויים שלכם למאגר המרוחק, ב-**pull** כדי לקבל את השינויים האחרונים מהמאגר המרוחק, ב-**fetch** כדי לבדוק אילו שינויים קיימים במאגר המרוחק, וב-**merge** כדי לשלב את השינויים מהמאגר המרוחק למאגר המקומי שלכם.

גלגול חזרה בזמן: checkout, revert, reset

1. **git checkout**
   מעבר לגרסה ספציפית של הקוד.
2. **git revert**
   יצירת שינוי שמבטל שינוי קודם.
3. **git reset**
   חזרה למצב קודם, עם או בלי שמירת השינויים.

מאפשרת מעבר בין **checkout** מאפשר לחזור אחורה בזמן ולתקן טעויות. הפקודה Git מחזירה את המאגר למצב קודם. חשוב **reset**-מבטלת שינוי קודם, ו **revert**, גרסאות להשתמש בפקודות אלו בזהירות ולוודא שאתם מבינים את ההשלכות של כל אחת מהן.

שמירת מצב: stash

1 שמירת שינויים זמנית

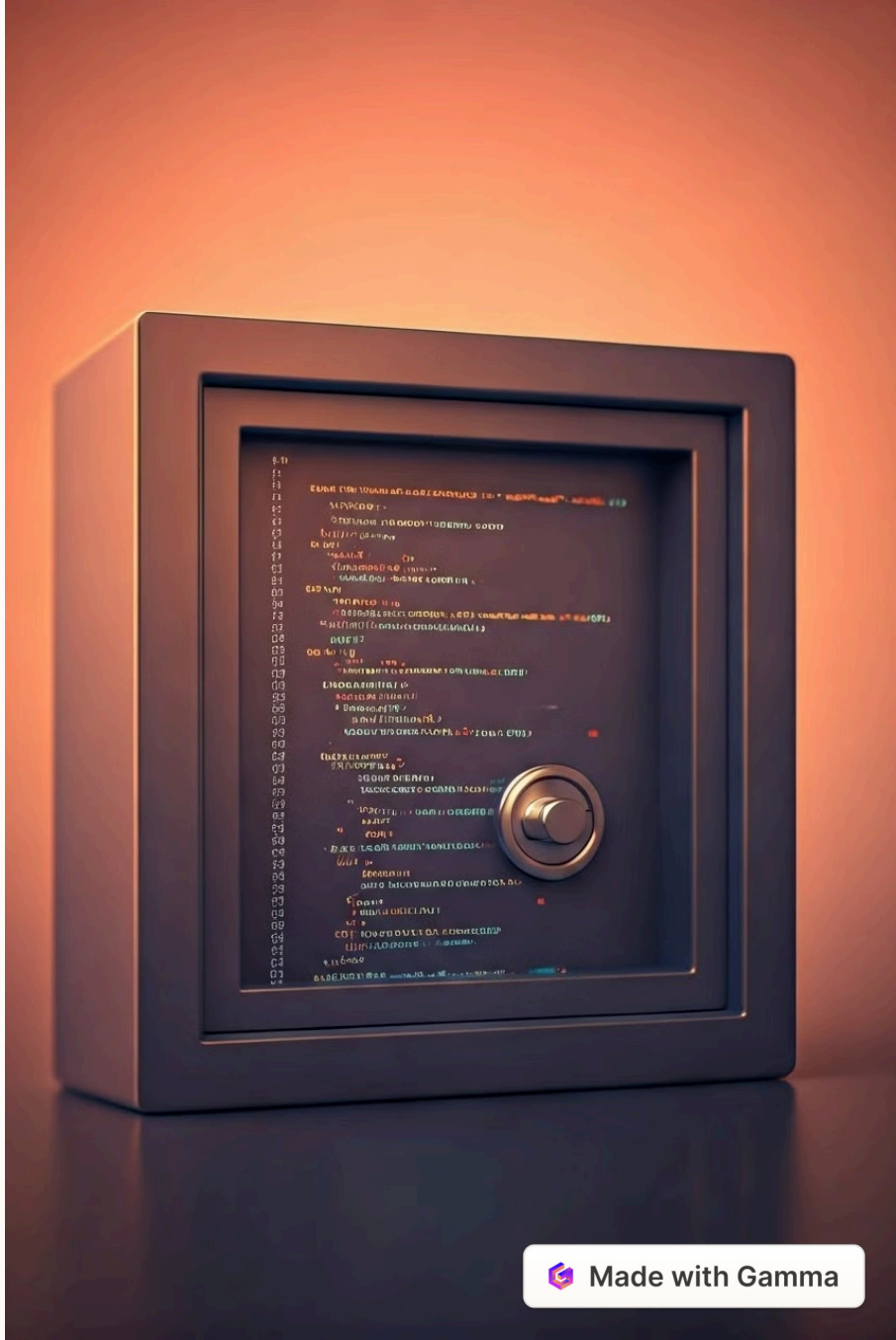
הפקודה **stash** מאפשרת לשמור שינויים זמניים שלא מוכנים עדיין ל-**commit**, ומאפשרת לעבור לעבוד על משימות אחרות.

2 מעבר בין משימות

ניתן לשמור מספר **stashes** ולעבור ביניהם בקלות, מה שמקל על ניהול משימות מרובות.

3 שחזור שינויים

הפקודה **stash pop** מאפשרת לשחזר את השינויים שנשמרו ולחזור לעבוד עליהם.



התנגשויות קוד ופתרונן

כיצד לפתור התנגשויות?

יש לערוך את הקובץ הבעייתי, לבחור את השינויים הרצויים, ולסמן את ההתנגשות כפתורה.

מהי התנגשות קוד?

התנגשות קוד מתרחשת כאשר שני מפתחים משנים את אותו קטע קוד, ו- Git לא מצליח למזג את השינויים באופן אוטומטי.

התנגשויות קוד הן חלק בלתי נפרד מפיתוח תוכנה מרובה משתתפים. חשוב להבין כיצד לזהות ולפתור אותן כדי למנוע בעיות בהמשך.

מרוחק Git: GitHub, GitLab

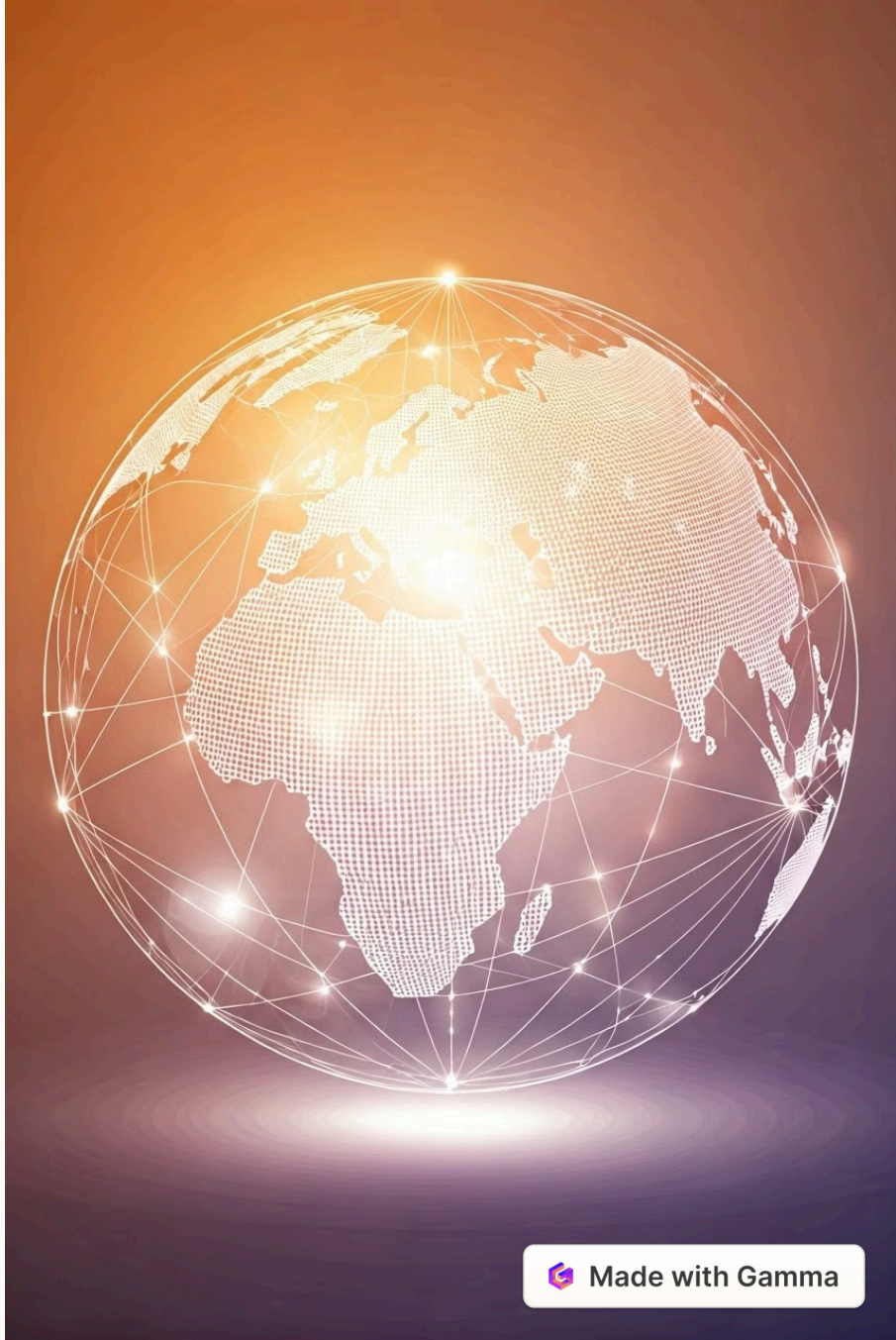
GitHub

פלטפורמת אירוח קוד פופולרית, המאפשרת שיתוף פעולה וניהול פרויקטים.

GitLab

פלטפורמה דומה ל-GitHub, עם דגש על כלי פיתוח ובדיקות אוטומטיות.

הן פלטפורמות אירוח קוד מרוחקות המאפשרות שיתוף פעולה וניהול פרויקטים. שתיהן מציעות מגוון רחב של כלים ושירותים, כגון ניהול גרסאות, מעקב אחר באגים, ובדיקות אוטומטיות.



סיכום והמלצות שימוש ב-Git

1 תכננו את העבודה שלכם

חלקו את המשימות לסניפים קטנים ומוגדרים היטב.

2 בצעו commit בתדירות גבוהה

שמרו את השינויים שלכם באופן קבוע, עם הודעות **commit** ברורות ותמציתיות.

3 תקשרו עם הצוות

שתפו פעולה עם חברי הצוות, ודאגו להבין את השינויים שלהם.

יכול לשפר את הפרודוקטיביות, Git-הוא כלי חיוני לפיתוח תוכנה מודרני. שימוש נכון ב Git להקל על שיתוף הפעולה, ולמנוע בעיות קוד. הקפידו על שימוש נכון בפקודות השונות, ותכננו את העבודה שלכם בצורה מסודרת.