- git checkout dev

Sposta il working directory (la cartella del progetto) sul branch chiamato dev.
Aggiorna i file del progetto allo stato dell'ultimo commit presente in quel branch.
Cambia anche l'HEAD, cioè il puntatore al branch attivo, in modo che i nuovi commit verranno aggiunti su dev,(cioè se sono su main e scrivo git checkout dev, passo a lavorare su dev)

Se ho modifiche non salvate (non committate) che vanno in conflitto con i file presenti su dev, Git impedirà di fare il checkout finché non le gestisco (con git commit, git stash o git reset).
Nelle versioni più recenti di Git, si preferisce usare git switch dev, che è più chiaro e dedicato solo ai branch.

- git pull origin dev

Svolge in sequenza:
Fetch: scarica dal remoto (origin) gli aggiornamenti del branch dev.
È come fare git fetch origin dev.
Merge (o rebase, se configurato): unisce i cambiamenti del dev remoto con il branch su cui sono localmente.

Se mi trovo già sul branch dev locale:

git pull origin dev aggiorna il dev con quello remoto ed è equivalente a:
- git fetch origin dev
- git merge origin/dev

Se mi trovo su un altro branch (es. main):
git pull origin dev scarica dev dal remoto e prova a unirlo dentro main.

- git fetch origin

- **git fetch** → scarica gli aggiornamenti (nuovi branch, commit, tag) dal repository remoto, ma **non modifica il working directory** e **non unisce nulla**.

- **origin** → è il nome predefinito del mio remoto principale (di solito quello da cui ho fatto git clone).

Dopo il comando

- Vengono aggiornati i riferimenti locali a come stanno i branch remoti:
    - origin/main, origin/dev, ecc.
- **Il mio branch attuale rimane invariato**: i file non cambiano, l'HEAD non si muove.
- Per vedere cosa è cambiato si usa:

- git log --oneline -5
mostrerà gli ultimi 5 commit con i loro messaggi

- git log --oneline origin/feature/04-places..HEAD
mostrerà esattamente quali sono i commit che si stanno per inviare.

- git status
dirà esattamente cosa è successo ai file.

Dunque:

- git fetch origin = "Guarda cosa c'è di nuovo su *origin*, ma non tocca i miei file".
- git pull origin dev = "Scarica da *origin*/*dev* e lo unisce dentro il mio branch attuale".

- git switch -c feature/04-places --track origin/feature/04-places

- **git switch:** serve a cambiare branch (alternativa moderna a git checkout per i branch).
- **-c feature/04-places**: crea un nuovo branch locale chiamato feature/04-places.
- **--track origin/feature/04-places** dice a Git:
    - imposta il branch remoto origin/feature/04-places come *upstream* del nuovo branch locale;

- in questo modo, quando si farà git pull o git push dal mio branch locale, Git saprà a quale branch remoto riferirsi.

In conclusione:

1. Creo un branch locale feature/04-places.
2. Lo collego al branch remoto origin/feature/04-places.
3. Mi sposto su quel branch (sarò subito dentro feature/04-places).

- git switch --track origin/feature/04-places
Git in questo caso crea automaticamente il branch locale con lo stesso nome del remoto. Questo comando è l'equivalente più breve del seguente:

- git checkout -b feature/04-places origin/feature/04-places oppure
- git branch --track feature/04-places origin/feature/04-places

- **git add .**

Aggiunge tutti i file (se voglio aggiungere un file specifico git add file.txt,se voglio aggiungere tutti i file js git add * js)

- **git commit -m "feat (places):implementazione sezione Places con card di navigazione visive"**

- **git push**

Serve a caricare i commit locale dalla repository locale verso quella remota. Con il push, git confronta i commit locali con quelli del repository remoto, invia solo i nuovi commit che non esistono ancora nel remote ed aggiorna il repository remoto con le mie modifiche

Per verificare le mie credenziali:
git config --global user.name
git config --global user.email

Per settare le mie credenziali:
git config --global user.name "Tuo Nome"
git config --global user.email "tua@email.com"

Rimuove file dallo staging:
git reset file.txt

Annulla ultimo commit mantenendo le modifiche
git reset --soft HEAD~1

Annulla ultimo commit perdendo tutto
git reset --hard HEAD~1

Unisce una branch nella branch corrente
git merge feature-branch

- git commit --amend -m "sezione places spostata dopo la sezione notizie"

Se non ho fatto ancora il push e sto lavorando solo io su un branch personale, è possibile cambiare l'ultimo commit (non ne crea un altro) riscrive soltanto l'ultimo commit esistente con un nuovo messaggio, in questo caso l'hash del commit rimane uguale e l'head continua a puntare al commit modificato.

- **git commit --amend -m "refactor(places): sposta sezione places dopo news + add docs"**

- refactor = ristrutturazione codice
- feat = nuova funzionalità
- fix = correzione bug
- chore = task di manutenzione
- docs = documentazione
- style = formatting, spaziatura