Abort, continue, skip

Thursday, July 3, 2025 5:41 PM

✅ First: git rebase --abort

📌 Purpose:

If a **conflict** happens during a rebase and you don't want to continue or you can't resolve it, this command allows you to **cancel the rebase** and go back to how things were before it started.

🧪 Example:

1. You have two branches:
   - master
   - feature (with some new changes)

bash
نسختحرير
```
# Switch to the feature branch
$ git checkout feature
# Start the rebase process
$ git rebase master
```
🎯 A conflict happens:

bash
نسختحرير
```
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
```

🛠️ To cancel the rebase and go back to the original state:

bash
نسختحرير
```
$ git rebase --abort
```
📌 This will cancel the rebase and return the feature branch to how it was before the rebase started.

✅ Second: git rebase --continue

📌 Purpose:

After you **manually resolve a conflict**, you use this command to continue the rebase process.

🧪 Steps:

1. During git rebase master, a conflict occurs in file.txt.

bash
نسختحرير
```
CONFLICT (content): Merge conflict in file.txt
```
2. You open the file and **resolve the conflict manually**, removing the conflict markers:

text

نسختحرير
```
<<<<<<< HEAD
// from master
=======
// from feature
>>>>>>> commit-hash
```
3. After resolving, stage the file:

bash
نسختحرير
```
$ git add file.txt
```
4. Then continue the rebase:

bash
نسختحرير
```
$ git rebase --continue
```
🔥 Git will proceed to the next commit and continue the rebase.

✅ Third: git rebase --skip

📌 Purpose:

If a certain commit causes issues and **is not important**, you can **skip** it and continue with the rest.

🧪 Example:

1. During rebase:

bash
نسختحرير
```
$ git rebase master
```
2. A conflict occurs in a commit that modifies E.txt.
3. Instead of fixing it, you choose to ignore that commit:

bash
نسختحرير
```
$ git rebase --skip
```
🔁 Git will skip that commit entirely and continue with the next ones.

🎨 Diagram (Before and After)

💡 Before Rebase:

mathematica
نسختحرير
```
master:   A---B---C
              \
feature:       D---E
```

✅ During git rebase master from feature:

Git tries to replay D and E on top of C:

mathematica
نسختحرير

A---B---C---D'---E'
If:

- A conflict happens in E
- You run git rebase --skip

📌 Result:

mathematica
نسختحرير
A---B---C---D' ← E is removed

🧠 Summary:

Command	**Purpose**	**When to Use**
git rebase --abort	Cancel the rebase	When a conflict happens and you don’t want to continue
git rebase --continue	Resume after resolving a conflict	After fixing the conflict and staging the changes
git rebase --skip	Skip a problematic commit	If a commit causes issues and you decide to ignore it

✅ 4. squash (or s)

Thursday, July 3, 2025 5:45 PM

✅ **4. squash (or s)**

🔷 **Purpose**: Combines this commit (squash) with the previous one (above it), meaning their changes will be merged into a **single commit**.

🔶 **Use Case**: When you want to **clean up your commit history** by grouping related small commits into one meaningful commit.

✅ Example

Suppose you have:

bash
نسختحرير
```
pick f26f25c add v1
squash b1e9d46 add v2
```

This tells Git:

> "Take the changes from commit b1e9d46 and combine them with the previous commit f26f25c."

📝 What Happens Next?

After you save and close the interactive rebase, Git will open another editor window with the **combined commit message**, like:

bash
نسختحرير
```
# Please enter the commit message for your changes.
# This is a combination of 2 commits.
# The first commit's message is:
add v1
# The commit message for the squash commit is:
add v2
```

✏️ You can now edit this to:

bash
نسختحرير
```
Add version 1 and 2 features
```

Or keep both messages:

bash
نسختحرير
```
add v1
add v2
```

Once saved, these two commits will be turned into **one single commit** with the changes from both.

✅ 5. fixup (or f)

Thursday, July 3, 2025 8:32 PM

✅ **5. fixup (or f)**
🔷 **Purpose**: Like squash, it **combines this commit with the one above it**, **but discards this commit's message**.
🔶 **Use Case**: When you want to **merge commits** and **keep only the original (first) message**, ignoring the message of the fixup commit.

✅ Example

bash
نسختحرير
pick f26f25c add v1
fixup b1e9d46 add v2
This tells Git:

> "Apply the changes from b1e9d46 into f26f25c, but **don't keep** the commit message from b1e9d46."

🎯 Result

- The **final single commit** will contain the **changes of both** commits.
- The **message will be only**:

  csharp
  نسختحرير
  add v1

No editor opens because Git doesn't ask you to edit messages.

💡 When to Use

- You corrected a typo, formatting, or bug shortly after a commit.
- You want a **clean history** and don't care about the intermediate commit message.

🧠 الشرح بالعربي:

✅ **fixup** مثل "دمج التعديلات" معناها squash، لكن بدون الاحتفاظ برسالة الـ commit المدموج.
🔧 **الاستخدام**: لو عندك commit رئيسي، وبعده commit تصحيحي، وعاوز تدمجهم وتحتفظ برسالة الأساسي فقط.
📌 **النتيجة**: يتم الدمج في commit واحد، برسالة add v1 فقط.

✅ 7. break

Thursday, July 3, 2025 9:08 PM

✅ **7. break**

🔷 **Purpose**: This command **pauses the rebase at that exact point**, allowing you to **manually inspect**, **debug**, or make changes before continuing.

🔶 **Use Case**: When you want to **pause intentionally** during an interactive rebase to:

- Inspect the working directory.
- Run manual tests or experiments.
- Change files before continuing.

✅ Example

```
pick f26f25c add v1
break
pick b1e9d46 add v2
```

⛔ What happens?

- Git applies the first commit f26f25c.
- Then it **pauses** the rebase **before** applying b1e9d46.

You'll see a message like:

```
Stopped at f26f25c... add v1
You can amend the commit now, with
  git commit --amend
Once you're satisfied with your changes, run
  git rebase --continue
```

🧪 What can you do during the break?

- **Inspect files**:

  ```
  cat somefile.js
  ```

- **Test the code manually**:

  ```
  bash
  نسختحرير
  npm start
  ```

- **Make edits**:

  ```
  bash
  نسختحرير
  code .
  ```

- **Stage and amend if needed**:

  ```
  bash
  نسختحرير
  git add .
  git commit --amend
  ```

- **Then continue**:

  ```
  bash
  نسختحرير
  git rebase --continue
  ```

🧠 الشرح بالعربي:

✅ **break** معناها "قف عند هذه النقطة" أثناء rebase.

🔧 **الاستخدام:**

لو حابب توقف العملية يدويًا بعد commit معين علشان:

- تفحص الملفات.
- تشغّل اختبارات يدويًا.

- تعدل على ملفات.
- تعمل debug لأي شيء.

📌 **بعد التوقّف**:
بإمكانك تعديل أي شيء، وبعدها تكمل بـ:

```
bash
نسختحرير
git rebase --continue
```

✅ سيناريو عملي: تعديل يدوي باستخدام break

📁 المشروع الحالي

افترض أنك في مشروع يحتوي على الكود التالي في الملف math.js:

```
js
نسختحرير
// math.js
function add(a, b) {
  return a + b;
}
module.exports = add;
```

📜 سجل الكوميتات:

```
bash
نسختحرير
git log --oneline
```

```
sql
نسختحرير
d123456 (HEAD -> master) add math function
c234567 fix typo
b345678 add test file
a456789 initial commit
```

🧪 الهدف

نريد أن نوقف عملية الـ rebase بعد commit fix typo **يدويًا** لنفحص الكود ونجرب تشغيله قبل متابعة rebase.

🔁 الخطوات

1. ابدأ rebase تفاعلي:

```
bash
نسختحرير
git rebase -i HEAD~3
```

2. في المحرر، عدّل إلى :

```
pick b345678 add test file
pick c234567 fix typo
break
pick d123456 add math function
```

3. ماذا سيحدث؟

- Git سيطبق:
  - add test file
  - fix typo
- ثم سيتوقف **يدويًا** عند نقطة break.

4. عند التوقف، افحص الكود يدويًا :

```bash
نسختحرير
cat math.js
node test/test.js  # أو أي سكربت اختبار موجود
```

أو يمكنك التعديل مثلًا:

```bash
نسختحرير
nano math.js
# صحح مشكلة إن وجدت
```

5. بعد الانتهاء، أكمل rebase:

```
git add .
git commit --amend  # لو عملت تعديل
git rebase --continue
```

🧠 ملاحظات مهمة:

💡 متى تستخدم break؟

الحالة	هل break مفيد؟
تريد اختبار يدوي بعد commit	✅ نعم
تريد إيقاف rebase عند نقطة محددة	✅ نعم
تريد تعديل أو تصحيح ملفات يدويًا	✅ نعم
تريد دمج تلقائي بدون تدخل	❌ لا

Thursday, July 3, 2025 9:56 PM

✅ 8. drop (or d)

🔷 Delete this commit — it will be gone after rebase.

✅ When: You want to remove a mistake or unwanted commit.

plaintext
نسختحرير
```
drop 68b4a9b add v3
```
➡️ That commit will disappear permanently (unless recovered with reflog).

✅ 9. label

Thursday, July 3, 2025 10:16 PM

✅ **9. label**
🔷 **Purpose**: Assigns a **custom name** (like a temporary tag) to a **specific point** during an interactive rebase.
🔶 **Use Case**: When you want to **refer back** to this exact commit later in the rebase script—especially if you're doing **complex rebases** involving reset, merge, or drop.

🧪 Example

Let's say you have this rebase plan:

plaintext
نسختحرير
```
pick f26f25c add v1
label start-login
pick b1e9d46 add v2
pick c63b2ab add login
```
Here:

- label start-login attaches the name start-login to the commit f26f25c (add v1).
- Later, you can use this label in other commands like reset or merge.

✅ Real-world scenario: Use with reset

plaintext
نسختحرير
```
pick f26f25c add v1
label before-feature
pick b1e9d46 add buggy-feature
reset before-feature
pick c63b2ab add clean-feature
```

🔄 What happens?

- Git applies add v1.
- Applies add buggy-feature.
- reset before-feature moves **HEAD back** to add v1, as if buggy-feature never happened.
- Then applies add clean-feature.

🧠 This allows you to **test a commit**, and if it's not good, **rewind to a label**, and move forward from there.

✅ Real-world scenario: Use with merge

plaintext
نسختحرير
```
pick f26f25c base commit
label base
pick b1e9d46 added feature A
```

pick c63b2ab added feature B
merge base added features
This creates a **merge commit** between the current HEAD and the base label—useful to preserve history intentionally.

🧠 الشرح بالعربي:

✅ label لنقطة في "تسمية مؤقتة" معناها rebase، تقدر ترجع لها لاحقًا باستخدام reset أو merge.
🔧 **الاستخدامات:**

الاستخدام	**النتيجة**
label mypoint	يعطي اسم مؤقت للـ commit الحالي
reset mypoint	يرجعك لنقطة mypoint
merge mypoint	يعمل دمج مع mypoint

📌 مفيدة جدًا لو بتعمل drop أو reset أو merge معقد داخل rebase script.

Rebase

Thursday, July 3, 2025 10:17 PM

✅ 1. pick (or p)

🔷 Use the commit as-is (default).

✅ When: You want to keep the commit unchanged.

plaintext
نسختحرير
```
pick f26f25c add v1
```
➡️ Git applies the commit normally, no changes made.

✅ 2. reword (or r)

🔷 Use the commit, but change its message.

✅ When: You like the commit content, but want to edit the message.

plaintext
نسختحرير
```
reword b1e9d46 add v2
```
➡️ Git pauses and opens an editor:

nginx
نسختحرير
```
add v2
# You can edit this
```
🔁 After saving, rebase continues.

✅ 3. edit (or e)

🔷 Pause here to modify the commit's content or message.

✅ When: You want to fix a bug or update the commit.

plaintext
نسختحرير
```
edit 68b4a9b add v3
```
➡️ Rebase pauses here. You then:

bash
نسختحرير
```
# Edit files
git add .
git commit --amend  # Edit message or files
git rebase --continue
```

✅ 4. squash (or s)

🔷 Combine this commit with the one above it. Messages merged.

✅ When: You want to clean history and combine related commits.

plaintext
نسختحرير
```
pick f26f25c add v1
squash b1e9d46 add v2
```
➡️ Editor opens to combine messages:

csharp
نسختحرير
```
# Combined commit message:
add v1
add v2
```

✅ 5. fixup (or f)

🔷 Like squash, but discard this commit's message.

✅ When: You want to combine commits and keep only the first commit's message.

plaintext
نسختحرير
```
pick f26f25c add v1
fixup b1e9d46 add v2
```
➡️ Result: One commit, message = add v1.

✅ 6. exec (or x)

🔷 Run a shell command at this point.

✅ When: You want to automate tests or commands.

plaintext
نسختحرير
```
exec npm test
```
➡️ Rebase will pause here to run npm test. Useful in CI pipelines.

✅ 7. break

🔷 Stop rebase manually and allow you to inspect or debug.

✅ When: You want to pause the rebase yourself.

plaintext
نسختحرير
```
break
```
➡️ You can inspect state, test something, etc. Then:

bash
نسختحرير
```
git rebase --continue
```

✅ 8. drop (or d)

🔷 Delete this commit — it will be gone after rebase.

✅ When: You want to remove a mistake or unwanted commit.

plaintext
نسختحرير
```
drop 68b4a9b add v3
```
➡️ That commit will disappear permanently (unless recovered with reflog).

✅ 9. label

🔷 Mark this point with a custom name (like a tag).

✅ When: You want to reference this point later in rebase.

plaintext
نسختحرير
```
label start-login
```
➡️ Can be used later with reset or merge.

✅ 10. reset

🔷 Move HEAD back to a label set earlier.

✅ When: You want to replay some commits again or change flow.

plaintext
نسختحرير
```
reset start-login
```
➡️ Rebase jumps back to the labeled point.

✅ 11. merge

🔷 Create a merge commit during rebase.

✅ When: You need a real merge (not fast-forward).

plaintext
نسختحرير
```
merge -C abc123 start-login
```
- -C abc123: Use the original message from merge commit abc123.
- start-login: The label to merge.

🧠 Advanced use for preserving merge history.

✅ 12. update-ref

🔷 **Track that a reference (like a branch) should be moved here.**
✅ **When: You want to update a branch/tag after rebase.**

```
plaintext
نسختحرير
update-ref refs/heads/feature-login
```

➡️ At the end of the rebase, the branch feature-login will point to this commit.

🔁 Example Rebase File Using All

```
plaintext
نسختحرير
reword f26f25c add v1     # Change commit message
edit b1e9d46 add v2       # Modify files or message
pick 68b4a9b add v3       # Keep as-is
fixup c63b2ab fix typo    # Merge into add v3
exec echo "Checkpoint done" # Run shell command
break                     # Pause here
drop 5ff1cc1 temp debug    # Delete unwanted commit
label start-clean         # Mark this point
reset start-clean         # Jump back
merge -C abc123 start-clean # Create a merge
update-ref refs/heads/dev   # Update branch to here
```

📚 Summary Table

Command	Purpose	Use Case
pick	Use commit as-is	Normal commit
reword	Change commit message	Fix typo or message clarity
edit	Modify content or message	Fix bug, update file
squash	Merge commit with previous	Combine related changes
fixup	Merge and discard message	Cleanup before pushing
exec	Run shell command	Tests or logging
break	Manually stop rebase	Debug or review
drop	Delete commit	Remove mistake
label	Name the point	Advanced scripting
reset	Move HEAD back to label	Replay commits differently
merge	Create merge commit	Preserve merge history
update-ref	Move ref to this commit	Update branch/tag post rebase