

بخش دوم

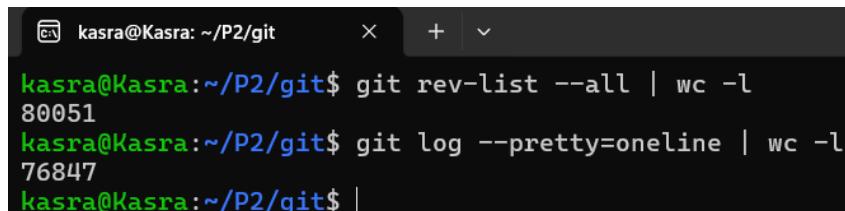
کسرا نوربخش 810100230

مقدمه

در این بخش با 2 ابزار مدیریت منبع کد و کنترل ورژن یعنی، git, mercurial آشنا شدیم.

تو پروژه چه خبره!

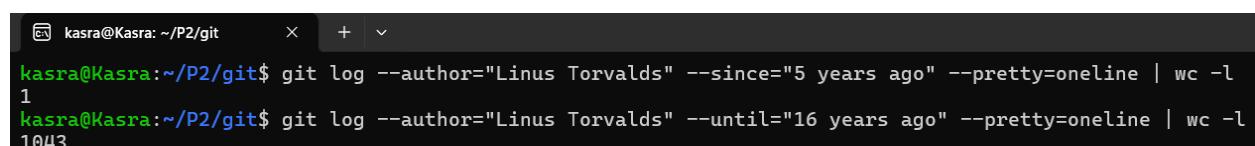
می دونستیم که برای کلون کردن یک پروژه باید از دستور git clone <> استفاده کنیم. برای شماردن تمامی کامیت ها از - | wc -l استفاده کردیم. همچنین می شد از دستور --git log --all | wc -l هم استفاده کنیم که جواب هایشان متفاوت بود به این دلیل که اولی، تمامی کامیت ها را از تمامی رفنس ها و برنج ها و تگ ها می شمارد اما دومی فقط کامیت هایی را که از HEAD فعلی قابل دسترس هستند که مشخصاً کمتر می باشد:



```
kasra@Kasra:~/P2/git$ git rev-list --all | wc -l
80051
kasra@Kasra:~/P2/git$ git log --pretty=oneline | wc -l
76847
kasra@Kasra:~/P2/git$ |
```

عکس 1: نتیجه خروجی دو دستور زده شده

برای بخش بعدی از git log --author="Linus Torvalds" --since="5 years ago" --pretty=oneline | wc -l استفاده شد و برای کامیت های گذشته اش از git log --author="Linus Torvalds" --until="16 years ago" --pretty=oneline | wc -l که نتیجه را مشاهده می کنیم:



```
kasra@Kasra:~/P2/git$ git log --author="Linus Torvalds" --since="5 years ago" --pretty=oneline | wc -l
1
kasra@Kasra:~/P2/git$ git log --author="Linus Torvalds" --until="16 years ago" --pretty=oneline | wc -l
1043
```

عکس 2: نتیجه خروجی دو دستور زده شده که مشاهده می شود Linus در روز های اول بیشتر فعال بوده است

تعداد کامیت های فایل مشخص گفته شده را با استفاده از - | wc -l و git log --oneline -- kernel/pid_namespace.c | wc -l از git log --diff-filter=A -- kernel/pid_namespace.c می زمان اضافه شدن این فایل را به پروژه با استفاده از زیر می شد استفاده کرد: توانستیم ببینیم. برای بدست آوردن تاریخ دقیق هم از دستور زیر می شد استفاده کرد:

```
.git log --diff-filter=A --format="%h %ad %an %s" --date=short -- kernel/pid_namespace.c
```

کانفیگ Git

برای این که نام و ایمیل رو به صورت local در کانفیگ گیت اضافه کنیم از دستورات زیر استفاده نمودیم:

```
git config user.name "Your Name"
```

```
git config user.email "your.email@example.com"
```



```
kasra@Kasra:~/P2/kasra-8101$ git config user.name "Kasra Noorbakhsh"
kasra@Kasra:~/P2/kasra-8101$ git config user.email "noorbakhshkasra@yahoo.com"
kasra@Kasra:~/P2/kasra-8101$ git config user.name
git config user.email
Kasra Noorbakhsh
noorbakhshkasra@yahoo.com
```

عکس ۳: نتیجه دو دستور زده شده

حالت local فقط برای یک مخزن خاص اعمال می شود و تنظیمات در فایل `git/config`. ذخیره می شوند. ضمناً این حالت دارای بالاترین اولویت است که اگر چند سطح مقدار داشته باشند، local بر بقیه غلبه می کند. حالت global رای همه مخازن کاربر فعلی اعمال می شود و تنظیمات در فایل `~/.gitconfig` ذخیره می شوند. در حالت system تنظیمات برای کل سیستم local > global (همه کاربران) می باشد و تنظیمات در `/etc/gitconfig` ذخیره می شوند. پس ترتیب اولویت به صورت:

`alias > system` می باشد. برای اضافه کردن alias اولی و دومی گفته شده، به ترتیب از:

```
git config --global alias.lg "log --oneline --all --graph"
```

```
git config --global alias.bb "bisect bad"
```

استفاده کردیم.

سناریو بازی!

ابتدا مطابق با سناریو گفته شده کامیت های A و B را، با توجه به فرمات داده شده زدیم. دستورات این دو:

```
echo "initial setup" > main.txt
```

```
git add .
```

```
git commit -m "A: initial commit"
```

```
echo "preparing for feature branches" >> main.txt
```

```
git commit -am "B: base for features"
```

برای بخش بعدی به برنج جدید رفتیم و کامیت C را زدیم. دستورات این قسمت:

```
git checkout -b Feature1
echo "Feature1 base" > feature1.txt
git add .
git commit -m "C: start feature1"
```

برای بخش بعدی به برنج اصلی رفتیم و کامیت D را زدیم و برنج جدید دیگری ساختیم. دستورات این قسمت:

```
git checkout master
echo "updated core logic" >> main.txt
git commit -am "D: core enhancements"
git checkout -b Feature2
```

برای ویرایش پیام کامیت C با `--amend`

```
git checkout Feature1
git commit --amend -m "C: added feature1 file"
```

حال کامیت F را در جای اشتباه زدیم:

```
echo "accidentally added to Feature1" >> feature1.txt
git commit -am "F: mistakenly added to Feature1"
```

حال به رفع باگ از طریق یک برنج جدید و Merge رفتیم:

```
git checkout master
git checkout -b bugfix
echo "fixing bug" > bug.txt
git add .
git commit -m "G: bug fixed"
git checkout master
git merge bugfix
```

این یک bugfix است چون هیچ کامیت جدیدی در master از زمان انشعاب fast-forward merge انجام نشده.

انواع مرج ها:

فقط pointer جلو میره چون انشعابی وجود نداشت.

اگر هر دو برنج تغییرات مستقل داشته باشند، Git از Recursive (Three-way) استفاده می‌کند.

چند کامیت رو در یک کامیت ترکیب می‌کنند (تاریخچه حفظ نمی‌شود).

```
kasra@Kasra:~/P2/kasra-810100230$ git checkout master
Switched to branch 'master'
kasra@Kasra:~/P2/kasra-810100230$ git merge bugfix
Updating 88cf483..033fc64
Fast-forward
  bug.txt | 1 +
  1 file changed, 1 insertion(+)
  create mode 100644 bug.txt
```

عکس 4: خود دستور مرج هم می‌نویسد که Fast-forward می‌باشد

برای کامیت F به Feature1 Feature2 و حذفش از Feature2 دستورات زیر را زدیم:

```
git log -oneline
```

```
git checkout Feature2
```

```
git cherry-pick df0d6f6
```

```
git checkout Feature1
```

```
git reset --hard HEAD~1
```

انواع سوییچ ها در git reset

--soft HEAD عقب می‌رمه working dir و staging دست نخورده باقی می‌مانند.

--mixed HEAD و staging عقب می‌رمه working dir دست نخورده باقی می‌مانند.

--hard همه عقب می‌رنه، بدیش اینه که همه تغییرات پاک می‌شن!

برای افزودن کامیت K و تگ‌گذاری نسخه v0.1 از دستورات زیر استفاده کردیم:

```
echo "Feature1 complete" >> feature1.txt
```

```
git commit -am "K: Feature1 complete"
```

```
git tag v0.1
```

دستورات زده شده برای ادامه توسعه در main با کامیت A:

```
git checkout master
```

```
echo -e "Line1\nLine2\nLine3\nLine4" > newfile.txt
```

```
git add -p
```

```
git commit -m "L: added first part of new file"
```

برای stash کردن دو خط باقیمانده و کار روی Feature2 این هارو زدیم:

```
git stash push -m "remaining 2 lines"
```

```
git checkout Feature2
```

```
echo "Feature2 finished" >> feature2.txt
```

```
git add .
```

```
git commit -m "M: Feature2 done"
```

```
git checkout master
```

```
git merge Feature2 -m "N: merged Feature2"
```

```
git tag v0.2
```

انواع سوییچ های stash

تغییرات رو اعمال می کنے ولی در stash نگه می دارند.

تغییرات رو اعمال و بعدش از stash حذف می کنند.

```
kasra@Kasra:~/P2/kasra-8101$ git log --graph --oneline --all
*   d27bdb4 (HEAD -> master, tag: v0.2) N: merged Feature2
| \
| * 2c60aea (Feature2) M: Feature2 done
| * 52fc1e0 F: mistakenly added to Feature1
| | * 63e844d (refs/stash) On master: remaining 2 lines
| |
| / \
|   * 414c860 index on master: 9ac53ad L: added first part of new file
|   |
|   / \
|     * 9ac53ad L: added first part of new file
|     * 033fc64 (bugfix) G: bug fixed
|   /
|   *
*   88cf483 D: core enhancements
|   * cc15ca1 (tag: v0.1, Feature1) K: Feature1 complete
|   * bc7a519 C: added feature1 file, prev message was wrong
|
* 5d794bb B: base for features
* 12aa43a A: initial commit
```

عکس 5: شمایی از اتفاقاتی که افتاد مطابق با صورت پروژه!!!

در بخش آخر هم که ریپو ساخته شد و به پروژه اضافه شد و تمامی تغییرات در اون push شدن.

آشنایی با Mercurial

بعد از نصب می بایستی که hg --version را بزنیم که در صورت پروژه به اشتباه با یک - آمده بود. برای این بخش دستورات زیر را به ترتیب زدیم:

```
hg init repo
```

```
cd repo
```

```
echo "file" > file.txt
```

```
hg add file.txt
```

```
hg commit -m "c0"
```

```
echo "c1" >> file.txt
```

```
hg commit -m "c1"
```

```
echo "c2" >> file.txt
```

```
hg commit -m "c2"
```

```
echo "c3" >> file.txt
```

```
hg commit -m "c3"
```

```
hg update -r 1    # c1
```

```
hg branch green
```

```
echo "green1" >> file.txt
```

```
hg commit -m "green1"
```

```
echo "green2" >> file.txt
```

```
hg commit -m "green2"
```

```
hg update default
```

```
hg update -r 3    # c3
```

```
echo "c4" >> file.txt
```

```
hg commit -m "c4"
```

```
hg update -r 3
hg branch blue
echo "blue1" >> file.txt
hg commit -m "blue1"
echo "blue2" >> file.txt
hg commit -m "blue2"
```

```

o  changeset:  6:56addae35246
|/ parent:    3:7a5329b1cdd0
|   user:      Kasra Noorbakhsh <noorbakhshkasra@yahoo.com>
|   date:      Wed Apr 23 16:51:47 2025 +0330
|   summary:   c4

o  changeset:  5:642c765f56c1
|/ branch:    green
|   user:      Kasra Noorbakhsh <noorbakhshkasra@yahoo.com>
|   date:      Wed Apr 23 16:51:29 2025 +0330
|   summary:   green2

o  changeset:  4:7d7df48d22fd
|/ branch:    green
|   parent:   1:dc639c6f87b7
|   user:      Kasra Noorbakhsh <noorbakhshkasra@yahoo.com>
|   date:      Wed Apr 23 16:51:24 2025 +0330
|   summary:   green1

o  changeset:  3:7a5329b1cdd0
|/ user:      Kasra Noorbakhsh <noorbakhshkasra@yahoo.com>
|   date:      Wed Apr 23 16:50:23 2025 +0330
|   summary:   c3

o  changeset:  2:950156d769df
|/ user:      Kasra Noorbakhsh <noorbakhshkasra@yahoo.com>
|   date:      Wed Apr 23 16:50:19 2025 +0330
|   summary:   c2

o  changeset:  1:dc639c6f87b7
|/ user:      Kasra Noorbakhsh <noorbakhshkasra@yahoo.com>
|   date:      Wed Apr 23 16:50:14 2025 +0330
|   summary:   c1

o  changeset:  0:78d834802abc
|/ user:      Kasra Noorbakhsh <noorbakhshkasra@yahoo.com>
|   date:      Wed Apr 23 16:50:02 2025 +0330
|   summary:   c0
```

عکس 6: بخشی از شمایی از اتفاقاتی که افتاد مطابق با صورت پروژه!!! (با استفاده از دستور hg log --graph)

برای مابقی دستوراتی که می توانند گراف مانند خروجی دهنند از چت جی پی تی پرسیده شد که در آنها موجود می باشد.

سوالات تشریحی

1. برای استفاده از ابزارهای git blame و git bisect، فرض می کنیم فایلی داریم که دچار باگ شده است. ابتدا با استفاده از git blame filename بررسی می کنیم که کدام کامیت مسئول اضافه کردن خط مورد نظر است. با دستور git bisect start شروع می کنیم، نسخه‌ای که خطای دارد را با git bisect bad و نسخه‌ای که سالم بوده را با git bisect good مشخص می کنیم. با اجرای تست‌ها روی نسخه‌های میانی، گیت به ما کمک می کند تا دقیقاً مشخص کنیم در کدام کامیت خطای خداه دارد.

2. مفهومی در مدیریت پروژه‌ها و مهندسی نرم‌افزار است که پس از وقوع یک باگ یا حادثه فنی، تیم فنی بدون گوشزد و یا مقصراً دانستن افراد، علت اصلی مشکل را بررسی می کند. هدف از این رویکرد، افزایش یادگیری سازمانی، بهبود فرآیندها و پیشگیری از تکرار باگ‌ها در آینده است. تمرکز اصلی بر تحلیل سیستمی و ساختن روحیه همکاری مثبت است.

3. دستور rebase برای بازنویسی هیستوری کامیت‌ها استفاده می‌شود تا ساختار شاخه‌ها ساده‌تر و تمیزتر به نظر برسد. برخلاف merge که تاریخچه شاخه‌ها را حفظ می‌کند، rebase باعث می‌شود انگار تغییرات ما بعد از آخرین کامیت برنج مقصد نوشته شده‌اند. در محیط‌های حساس تیم فنی طور استفاده از rebase روی برنج‌های عمومی رو توصیه نمی‌کنم چون هیستوری را تغییر می‌دهد و ممکن است موجب کانفلیکت شود. اما برای برنج‌های شخصی یا قبل از مرج نهایی به منظور تمیز نگه داشتن هیستوری پروژه، مفید است.

4. تفاوت اصلی بین fetch و pull در این است که fetch فقط تغییرات را از ریپو ریموت دریافت می‌کند و آن‌ها را در برنچی جدا ذخیره می‌کند بدون اینکه آن‌ها را با برنج محلی ادغام کند. در مقابل، pull همان fetch به علاوه‌ی merge است، یعنی تغییرات را دریافت کرده و بلافصله با برنج فعلی ادغام می‌کند. fetch مناسب برای وقتی هست که می‌خوایم اول بررسی کنیم چه تغییراتی انجام شده و بعدش یا مرج بکنیم با نه.

5. Git و Mercurial هر دو سیستم‌های کنترل ورژن‌ان، ولی Git به دلیل سرعت، انعطاف‌پذیری، پشتیبانی گستره‌ده و ابزارهای جانبی قدرتمندتر، در پروژه‌های open source و تجاری خیلی محبوب‌تر است. Git از مدل برنچینگ و مرج پیشرفته‌ای برخوردار است، در حالی که Mercurial سادگی بیشتری در UI دارد. امکان کار با ریپو‌های ریموت را دارد و با استفاده از hg-git متصل شد، اما به صورت دیفالت این پشتیبانی وجود ندارد و معمولاً پیچیدگی بیشتری نسبت به Git دارد.

لینک های استفاده شده

برای پیش برد بخش سناریو بازی از ChatGPT کمک گرفته شد که لینک آن:

<https://chatgpt.com/share/6808d639-a0c0-800d-9a11-e949ec5787d6>

چت کامل:

عکس گراف رو بالاتر فرستادم باید این کارها رو بکنیم: در کامیت دوم B تصمیم میگیریم که با اضافه کردن یک فیچر به پروژه ورژن اولی هی اون رو ایجاد کنیم. برنچی با Feature1 رو به پروژه اضافه کنیم و یک فایل برای این فیچر در کامیت C اضافه کنیم. سپس به برنج اصلی برگردین تا روند رو ادامه ب دین. کامیت D و سپس یک برنج برای Feature2 اضافه کنید. حالا فرض کنیم که متوجه اشتباهی در مسیح کامیت C میشیم. بنابراین به برنج Feature1 برین و مسیح این کامیت رو با استفاده از Feature2 تغییر بدین. کامیت F را به اشتباه در ادامه برنج Feature1 اضافه کنیم. فرض کنین فراموش کردن که به برنج Feature2 git commit --ammend درست منقول کنیم و سپس به کمک git reset این کامیت رو حذف کنیم. توضیح بدین که از کدوم حالت استفاده میکنیم و به اختصار در مورد تفاوت soft و hard در git reset mixed و fast forward هست یا نه و در مورد انواع مرج به اختصار توضیح بدین. سپس با استفاده از قابلیت cherry-pick در گیت، کامیت F رو به برنج انجام شده از نوع patch در نهایتا کامیت K رو اضافه کنین و نسخه‌ی اولی هی کد رو با v0.1 تگ بزنین 10. به کامیت A برگردین تا توسع اضافه کنین اما در مورد دو خط دیگه مطمئن نیستید و میخواید در working directory اونها رو نگه دارین. بدون حذف کردن این خطوط، کامیت A رو به پروژه خیلی عالی! این سناریو گام به گام هم مفاهیم اصلی Git رو پوشش میده و هم مهارت کار با دستورات مهم رو تمرین می‌کنه. من طبق مراحل گفته شده بہت دستورهای دقیق و توضیح‌های لازم رو میدم.

گام اول: کامیت A و B

bash

CopyEdit

mkdir project

cd project

git init

```
echo "initial setup" > main.txt
```

```
git add .
```

```
git commit -m "A: initial commit"
```

```
echo "preparing for feature branches" >> main.txt
```



git commit -am "B: base for features"

C گام دوم: ساخت **Feature1** و کامیت

bash

CopyEdit

git checkout -b Feature1

echo "Feature1 base" > feature1.txt

git add .

git commit -m "C: start feature1"

D برگشت به **main** و ادامه روند با کامیت **Feature2** و ساخت

bash

CopyEdit

git checkout master

echo "updated core logic" >> main.txt

git commit -am "D: core enhancements"

git checkout -b Feature2

--amend پیام کامیت C با ویرایش

bash

CopyEdit

git checkout Feature1

git commit --amend -m "C: added feature1 file"

E اشتباه در F: انجام روی **Feature1** به جای **Feature2**

bash

CopyEdit

echo "accidentally added to Feature1" >> feature1.txt

git commit -am "F: mistakenly added to Feature1"

رفع باگ از طریق یک برنج جدید و Merge

bash

CopyEdit

git checkout master

git checkout -b bugfix

echo "fixing bug" > bug.txt

git add .

git commit -m "G: bug fixed"

git checkout master

git merge bugfix

این یک **fast-forward merge** است چون هیچ کامیت جدیدی در master از زمان انشعاب bugfix اجام نشده.

انواع Merge

- **Fast-forward:** فقط pointer جلو میره چون انشعابی وجود نداشته.

- **Recursive (Three-way):** اگر هر دو برنج تغییرات مستقل داشته باشند، Git merge commit استفاده می‌کنه.

- **Squash Merge:** چند کامیت رو در یک کامیت ترکیب می‌کنه (تاریخچه حفظ نمی‌شود).

کامیت Feature1 به Feature2 و حذفش از Feature1 Cherry-pick

- . پیدا کردن هش کامیت:

bash

CopyEdit

git log --oneline

- . فرض کنیم هش abc123 هست:

bash

CopyEdit

git checkout Feature2

git cherry-pick abc123

- . حذف کامیت F از Feature1:

bash

CopyEdit

git checkout Feature1

git reset --hard HEAD~1

🔍 git reset:

- HEAD--soft: عقب می‌رده working dir و staging دست نخورده.

- HEAD--mixed: عقب می‌رده working dir و staging دست نخورده.

- HEAD--hard: همه عقب می‌رن؛ هشدار: تغییرات پاک می‌شن!

افزودن کامیت K و تگ‌گذاری نسخه v0.1

bash

CopyEdit

```
echo "Feature1 complete" >> feature1.txt
```

```
git commit -am "K: Feature1 complete"
```

```
git tag v0.1
```

ادامه توسعه در main با کامیت I

bash

CopyEdit

```
git checkout master
```

```
echo -e "Line1\nLine2\nLine3\nLine4" > newfile.txt
```

```
git add -p # فقط Line1 و Line2 را انتخاب کن
```

```
git commit -m "L: added first part of new file"
```

Feature2 کردن دو خط باقی‌مانده و کار روی stash

bash

CopyEdit

```
git stash push -m "remaining 2 lines"
```

```
git checkout Feature2
```

```
echo "Feature2 finished" >> feature2.txt
```

```
git add .
```

```
git commit -m "M: Feature2 done"
```

```
git checkout master
```

git merge Feature2 -m "N: merged Feature2"

git tag v0.2

⌚ git stash:

- تغییرات رو اعمال می کنه ولی در stash نگه می داره.

- تغییرات رو اعمال و بعدش از stash حذف می کنه.

اگه خواستی با git log --oneline --graph کار رو ببینی.
در ادامه اگر بخوای مرحله به مرحله اجرا کنیم یا خطاهای خاصی دیدی، بگو تا دقیق‌تر راهنمایی‌ت کنم.

You said:

```
: kasra@Kasra:~/P2/kasra-810100230$ git log --oneline df0d6f6 (HEAD -> Feature1)
Feature1) F: mistakenly added to Feature1 bc7a519 C: added feature1 file, prev message was wrong 5d794bb B:
base for features 12aa43a A: initial commit kasra@Kasra:~/P2/kasra-810100230$ git checkout Feature2
Switched to branch 'Feature2' kasra@Kasra:~/P2/kasra-810100230$ git cherry-pick df0d6f6 CONFLICT
(modify/delete): feature1.txt deleted in HEAD and modified in df0d6f6 (F: mistakenly added to Feature1).
Version df0d6f6 (F: mistakenly added to Feature1) of feature1.txt left in tree. error: could not apply df0d6f6... F:
mistakenly added to Feature1 hint: After resolving the conflicts, mark them with hint: "git add/rm <pathspec>", then run hint: "git cherry-pick --continue". hint: You can instead skip this commit with "git cherry-pick --skip". hint: To abort and get back to the state before "git cherry-pick", hint: run "git cherry-pick --abort".
```

ChatGPT said:

عالی پیش رفتی! مشکلی که دیدی طبیعیه و بخارط اینه که در برنج Feature2 وجود نداره (احتمالاً حذف شده یا اصلاً ساخته نشده)، ولی کامیت F تلاش می کنه اون فایل رو تغییر بدنه، بنابراین Git به درستی بہت پیام conflict می ده.

راه حل مرحله به مرحله برای حل Conflict در cherry-pick:

1. وضعیت فعلی رو بررسی کن:

bash

EditCopy

git status

خواهی دید که conflict در feature1.txt هست.

2. اگر می خوای تغییرات فایل feature1.txt کامیت F رو نگه داری:

bash

EditCopy

git add feature1.txt

git cherry-pick --continue

این فایل رو به Feature2 اضافه می کنه حتی اگر او نجا قبلاً نبود.

. ۳. اگر نمی خوای این فایل رو وارد Feature2 کنی (مثالاً چون به اون مربوط نیست):

bash

CopyEdit

git cherry-pick --abort

و بعد راه حل جایگزین استفاده کن:

راه حل جایگزین: تغییر کامیت F به حذف فایل قبل از cherry-pick یا فقط محتوای لازم رو بگیر (✓)

اگر فقط یه خط خاص از کامیت F رو می خوای، می تونی:

(الف) یک برنج موقت بساز:

bash

CopyEdit

git checkout Feature1

git checkout -b temp-F

ب) تغییر مورد نظر رو دستی کپی کن (مثالاً خطی از feature1.txt رو)

پ) برگرد به Feature2 و اضافه شن کن:

bash

CopyEdit

git checkout Feature2

paste manually what you want into a relevant file (like feature2.txt)

git add .

git commit -m "F-prime: applied part of F to Feature2"

خلاصه:

- اگر Feature2 رو تو Feature1.txt را cherry-pick رو و می خواهی، abort کن.

- اگر فایل لازم بود، add کن و continue -بزن.

- یا با برنج موقت فقط بخش موردنیاز F رو دستی وارد کن.

هر روشی رو انتخاب کردی بگو تا بر اساسش بریم گام بعدی) حذف F از Feature1 با reset و ادامه مراحل.)

You said:

قبل از کامیت آی که باید کار هارو بکنیم این مشکل هست.
 kasra@Kasra:~/P2/kasra-810100230\$ git add -p No changes.
 kasra@Kasra:~/P2/kasra-810100230\$ git commit -m "L: added first part of new file" On branch master
 Untracked files: (use "git add <file>..." to include in what will be committed) newfile.txt nothing added to commit
 but untracked files present (use "git add" to track)

ChatGPT said:

عالیه که دقیق جلو میری! این پیغام به این معنیه که فایل جدیدت (newfile.txt) هنوز به گیت معرفی نشده، یعنی هنوز **"tracked"** نیست، و چون - git add **"tracked"** فقط برای فایل‌های **(دنیال شده)** کار می‌کنه، چیزی برای نشون دادن نداره.

راه حل قدم به قدم:

1. اول باید فایل جدید رو به گیت معرفی کنی (بدون اینکه همه‌شو کامیت کنیم):

bash

CopyEdit

```
git add -N newfile.txt
```

◇ این دستور فایل رو به صورت **tracked** اضافه می‌کنه، ولی محتوایی به Staging نمی‌بره.

2. حالا که فایل **tracked** شده، می‌توانی از **git add -p** استفاده کنی:

bash

CopyEdit

```
git add -p
```

اینجا گیت تغییرات فایل newfile.txt را به صورت patch نشون می‌ده و ازت می‌پرسه که هر بخش (hunk) یا (hunk) را می‌خوای اضافه کنی یا نه.

3. بعد از انتخاب دو خط اول، کامیت کن:

bash

CopyEdit

```
git commit -m "L: added first part of new file"
```

و دو خط دیگه همچنان در working directory باقی می‌مونن.

نکته:

اگر می‌خوای بدون patch کردن فایل، دستی همون دو خط اول رو وارد کنی و بعد git add newfile.txt بزنی، این کل فایل رو استیج می‌کنه، که مد نظرت نیست.

اگر خواستی با stash (دادمه بدیم) قبیل از رفتن به Feature2 پگو تا گام بعدی رو دقیق برأت بچینم.

You said:

بعد از `git add -p` چجوری می تونم بگم فقط دو خط اول

ChatGPT said:

سوال خیلی خوبی پرسیدی چون `git add -p` دقیقاً برای همین موقع هاست ☺

وقتی `git add -p` و نتیجه Git تغییرات را به صورت تکه‌ای (hunk) نشون می‌ده و ازت می‌پرسه می‌خوای اون بخش رو **stage** کنی یا نه.

ولی نکته‌ی مهم:

گاهی ممکنه Git همه چهار خط رو توی یه hunk نشون بده ☺ چون خودش تصمیم می‌گیره که تغییرات چطوری تقسیم بشن.

راحل اگر همه ۴ خط در یک hunk باشه: ✎

وقتی Git ازت می‌پرسه:

kotlin

CopyEdit

Stage this hunk [y,n,q,a,d,e,?]?

تو اینجا `e**` رو بزن. (Edit).

۳ مرحله بعد:

یه ادیتور باز می‌کنه و چیزی شبیه به این نشون می‌ده:

diff

CopyEdit

`@@ -0,0 +1,4 @@`

+line 1

+line 2

+line 3

+line 4

اینجا فقط دو خط اول رو نگه دار و بقیه رو پاک کن:

diff

CopyEdit

`@@ -0,0 +1,2 @@`

+line 1

+line 2

فایل رو ذخیره کن و خارج شو (در nano با `Y`, بعد `Ctrl + X`, Enter).

حالا:

فقط دو خط اول stage شدن ●

دو خط دیگه هنوز unstaged هستن و توی working directory باقی میمونن.

حالا میتونی:

bash

CopyEdit

git commit -m "L: added first part of new file"

و مرحله رو کامل کنی.

اگه خواستی در گام بعدی وارد stash بشیم و اون دو خط باقیمانده رو حفظ کنیم قبل از رفتن به Feature2، بگو تا برات انجامش بدیم.

You said:

نهایتا در مرحله‌ی آخر، یک ریپازیتوری در اکانت گی تهاب خودتون با نام ECS-Git-Practice ایجاد کنین و اون رو به عنوان ریپازیتوری ریموت به پروژه اضافه کنین و برنچها و تگهای مختلف پروژه رو به ریپازیتوری ریموت اضافه کنین.

ChatGPT said:

آفرین! رسیدیم به مرحله‌ی نهایی که قراره پروژه‌ت رو به GitHub پوش کنی 

در ادامه، قدم‌به‌قدم راهنماییت می‌کنم چطور:

1. به ریپازیتوری جدید توی GitHub بسازی.

2. اون رو به پروژه اضافه کنی.

3. همه‌ی برنچها و تگها رو پوش کنی.

مرحله ۱: ساخت ریپازیتوری در GitHub

1. وارد GitHub شو.

2. روی دکمه‌ی **New repository** کلیک کن.

3. اسم ریبو رو بذار : ECS-Git-Practice

4. گزینه‌ی **Initialize this repository with a README** رو خاموش بدار (چون ما لوكالی ریبو داریم.)

5. روی **Create repository** کلیک کن.

بعد از ساخت، URL بہت یه شبیه می‌ده:

 bash

CopyEdit

<https://github.com/<your-username>/ECS-Git-Practice.git>

مرحله ۲: اضافه کردن ریموت به پروژه لوکال

در پوششی پروژت (که تا اینجا همهی کارها روش انجام شده):

 bash

CopyEdit

`git remote add origin https://github.com/<your-username>/ECS-Git-Practice.git`

(یادت نره `<your-username>` خودتو بزاری!)

مرحله ۳: پوش کردن همهی برنجها و تگها

❖ پوش همهی برنجها:

 bash

CopyEdit

`git push origin --all`

❖ پوش همهی تگها:

 bash

CopyEdit

`git push origin --tags`

اگر از قبل یک برنج `main` روی گیت‌هاب ایجاد شده بود:

ممکنه پوش کردن `master` خطای چون GitHub برنج پیش‌فرض `main` هست. در این صورت، می‌تونی:

 bash

CopyEdit

`git push -u origin master`

یا اگر لازم بود `main` رو به `master` تغییر بدی، هم می‌شه انجام داد (فقط اگر خواستی).

اگه خواستی بدونی کدوم برنج‌ها، تگ‌ها یا `alias` ها رو اضافه کردی یا ساختار گراف نهایی چی شد، می‌تونی `git log --oneline --all --graph --decorate`

تا ببینیش.

بگو آگه دوست داری یه دستور برای گرفتن گراف پروژه‌ی نهاییت بدم یا فایل README هم بسازیم و پوش کنیم.

You said:

```
kasra@Kasra:~/P2/kasra-810100230$ git push origin --all Username for 'https://github.com': kasra-noorbakhsh
Password for 'https://kasra-noorbakhsh@github.com': remote: Support for password authentication was
removed on August 13, 2021. remote: Please see https://docs.github.com/get-started/getting-started-with-
git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of
authentication. fatal: Authentication failed for 'https://github.com/kasra-noorbakhsh/ECS-Git-Practice.git'/
```

ChatGPT said:

آره این خطأ طبيعية چون GitHub از اوتفیکیشن با پسورد پشتیبانی نمی‌کنه دیگه (از سال 2021 به بعد) ✗

راه حل: استفاده از **Personal Access Token (PAT)** ✓

اینجا برات کامل توضیح می‌دم چطوری به جای رمز عبور PAT بسازی و استفاده کنی! 🔒

۱. ساختن Token جدید ✨

وارد GitHub Settings → Developer Settings → Personal Access Tokens شو.

روی "Tokens (classic)" کلیک کن.

گزینه‌ی **Generate new token (classic)** رو بزن.

: "git push token" به اسم براش انتخاب کن (مثلاً)

Expiration: بذار مثلاً 30 روز یا custom.

Permissions: تیک بزن:

)repo (برای دسترسی به ریپازیتوری‌ها)

پایین صفحه روی **Generate token** کلیک کن.

توکن تولیدشده رو کپی کن و یه جایی ذخیره کن (فقط یک بار نشون داده میشه!)

۲. استفاده از Token به جای پسورد 🔑

حالا که می‌خوای push کنی، وقتی Git ازت Username و Password خواست:

Username: kasra-noorbakhsh •

Password: ون توکن طولانی که ساختی (Paste) کن!

برای راحتی بیشتر ⚡ :

تا مجبور نباشی هر بار توکن رو وارد کنی، Git می‌تونه اونو ذخیره کنه:

bash

CopyEdit

git config --global credential.helper cache

یا (در سیستم‌های جدیدتر و مطمئن‌تر):

bash

CopyEdit

git config --global credential.helper store

اگه خواستی می‌تونیم به جای SSH از HTTPS هم استفاده کنیم، که راحت‌تره برای push کردن‌های زیاد) فقط یه بار key setup می‌خواد).

بگو اگه می‌خوای روشن SSH رو هم یاد بگیری 😊

برای بخش آشنایی با ChatGPT از Mercurial کمک گرفتم.

<https://chatgpt.com/share/6808ebca-65a0-800d-9640-22af892c1c99>

چت کامل:

Using hg init, hg add, hg commit, hg branch, hg update. I want to create the graph shown in the picture(each round is a commit)

ChatGPT said:

The graph you've shown is a commit history from Mercurial (hg), and each colored line indicates a separate branch. Here's how you can recreate it step-by-step using hg init, hg add, hg commit, hg branch, and hg update.

The graph represents:

- A main line of commits (gray). •
- A branch from an early commit (green) with two commits. •
- A branch from a later commit (blue) with two commits. •

✓ Step-by-step Mercurial Commands:

Step 1: Initialize and create the first commit

bash

CopyEdit

hg init repo

cd repo

echo "file" > file.txt

hg add file.txt

hg commit -m "c0"

Step 2: Add 3 more commits on the default branch**bash****CopyEdit**`echo "c1" >> file.txt``hg commit -m "c1"``echo "c2" >> file.txt``hg commit -m "c2"``echo "c3" >> file.txt``hg commit -m "c3"`

Now you have a linear history: c0 → c1 → c2 → c3

Step 3: Create green branch from c1**bash****CopyEdit**`hg update -r 1 # c1``hg branch green``echo "green1" >> file.txt``hg commit -m "green1"``echo "green2" >> file.txt``hg commit -m "green2"`

Step 4: Go back to default and continue to c4**bash****CopyEdit**`hg update default``hg update -r 3 # c3``echo "c4" >> file.txt``hg commit -m "c4"`

Step 5: Create blue branch from c3**bash****CopyEdit**`hg update -r 3`



hg branch blue

echo "blue1" >> file.txt

hg commit -m "blue1"

echo "blue2" >> file.txt

hg commit -m "blue2"

Step 6: Done!

You now have the structure:

Default: c0 → c1 → c2 → c3 → c4 •

Green: branch from c1, has 2 commits •

Blue: branch from c3, has 2 commits •

You can view the graph with:

bash

CopyEdit

hg log --graph

Would you like a shell script version of this?

4o

You said:

در مورد ابزارها و دستوراتی که یک گراف از کامیت‌های شما در mercurial نمایش میدن مثل hg log --graph تحقیق

ChatGPT said:

ابزارها و دستوراتی که برای نمایش گراف کامیت‌ها در Mercurial استفاده می‌شن، به شما کمک می‌کنن ساختار شاخه‌ها (branches)، مسیرهای انشعاب (merges)، و ادغام‌ها (divergence) را به صورت گرافیکی یا متنی مشاهده کنین.

در ادامه، معرفی کامل ابزارها و دستورات مربوطه:

: hg log --graph

این دستور گراف کامیت‌ها را به شکل متنی در ترمینال نمایش می‌دهد.

نمونه استفاده:

bash

CopyEdit

hg log --graph

گزینه‌های مفید:

bash

CopyEdit

```
hg log --graph --template "{rev}:{branch}:{desc|firstline}\n"
```

این خروجی‌ای فشرده و خواناتر می‌ده و نشون می‌ده که هر کامیت به چه شاخه‌ای تعلق دارد.

 ابزار گرافیکی (**hgk**) : اختیاری

hgk یک رابط گرافیکی برای مشاهده گراف کامیت‌هاست.

نصب:

bash

Edit

sudo apt install hgk

اجرا:

bash

Edit

hg view

توجه **hgk**: بر پایه **Git** ساخته شده، و گاهی به نصب **Git** هم نیاز دارد.

 ابزار تحت وب :

برای مشاهده گراف کامیت‌ها از طریق مرورگر.

اجرا:

bash

Edit

hg serve

سپس مرورگر رو باز کن و برو به:

arduino

Edit

http://localhost:8000

در بخش "changelog" یا "graph" ، گراف کامیت‌ها نمایش داده می‌شود.

 افزونه‌ها (**Extensions**)

1. Graphlog Extension

بعضی نسخه‌های **Mercurial** نیاز دارن این افزونه فعال باشند.

فعالسازی در: **~/.hgrc**

ini



CopyEdit

[extensions]

graphlog =

حالا می‌توانی از دستور استفاده کنی:

bash

CopyEdit

hg glog

ابزارهای مکمل (خارج از Mercurial)

نسخه گرافیکی برای TortoiseHg: •

در آن، تب "Revision History" گراف کامیت‌ها را نشون می‌دهد.

نرم‌افزار رایگان از Atlassian با پشتیبانی از Mercurial در نسخه‌های قدیمی SourceTree: •

Rhg (Rust implementation): • سریع‌تر و سبک‌تره ولی هنوز در حال توسعه است.

خلاصه:

ابزار / دستور توضیح مختصر

hg log --graph نمایش گراف متنی در ترمینال

hg log --graph --template نمایش سفارشی اطلاعات گراف

hg view رابط گرافیکی hgk برای گراف

hg serve + مشاهده گراف در مرورگر hg

TortoiseHg GUI رسمی Mercurial نسخه

graphlog افزونه hg glog برای دستور

اگر دوست داری، می‌تونم یک خروجی گراف مثال واقعی برای شبیه‌سازی کنم، یا یک اسکریپت آماده برای اجرای همه این‌ها در یک پروژه تمرينی بسازم.
دوست داری؟