

بخش اول

کسرا نوربخش 810100230

مقدمه

در این بخش به کار با ادیتور Vim پرداختیم.

توانایی کار با Vim

برای این بخش، یک فایل tmp.text ساختم و داخل آن vim کردم تا با آن کار کنم.

جابجایی در ادیتور Vim

پایین: کلید ز، بالا: کلید k، راست: کلید l و چپ: کلید h.

برای جلو رفتن به ابتدای کلمه چهارم: 3w.

برای به انتهای کلمه فروند آمدن: 4e و سپس 4 کلمه به عقب: 4b. Vim رفتاری عجیب دارد که وقتی می زیم 4e، 4 کلمه جلو می رود اما کرسر یک کاراکتر قبل تر را نشان می دهد گرچه در واقعیت بر روی کاراکتر آخر کلمه چهارم قرار گرفته است.

برای رفتن به خط اول فایل: gg.

برای رفتن به خط آخر فایل: G.

برای رفتن به خط 18 اوم: 18G.

نوشتن و پاک کردن

ابتدا 15G و سپس A برای رفتن به آخر خط در حالت نوشتن.

ابتدا 10G و سپس V برای مد ویژوال و سپس W برای انتخاب کلمه و سپس U برای uppercase.

با 0 می توانیم به ابتدای خط برویم، با d2wi کلمه را پاک می کنیم و وارد insert مد می شویم.

با dw کلمه را کات می کنیم، با 8GP به خط 8 اوم می رویم و آن را در ابتدا پیست می کنیم.

.R overstrike رفتن به مد

ابتدا 3G، سپس 3w و در نهایت D برای پاک کردن تا انتهای خط.

ابتدا 2yy، سپس G برای رفتن به انتهای فایل و در نهایت 3p برای پیست کردن.

ابتدا `v` برای ورود به مد `visual`, سپس `ew` برای انتخاب کلمه فعلی و کلمه بعدش، `u` برای کپی و `ggP` برای رفتن به ابتدا و پیست کردن.

ابتدا `v` برای ورود به مد `visual`, سپس `z` برای انتخاب خط ها، `/#/#/` برای اضافه کردن `#` به اول و در نهایت `//#/#/` برای `uncomment` کردن.

با استفاده از `r<char>` حرفی رو که زیر `cursor` هست را جانشینی می کنیم با `.char`.

سرچ کردن و اجرای کامند

با `/word` می توانیم `word` را جست و جو کنیم و با `n` به جلو و با `N` در نتایج به عقب حرکت کنیم.

با `\cword` سرج می کنیم.

با استفاده از `OLDWORD :%s/OLDWORD/NEWWORD/gic` را جایگزین `NEWWORLD` می کنیم.

در داخل `vimrc` خط `set hlsearch` را اضافه کردیم.

```

kasra@Kasra: ~/P1
eqfafhgkfgkfg
dgh kasra dg
fhrskasrahfghftg

gh
d
tthaa dgh
d
j HkasraGFHhdjsjkk JDDJD Dddddd
tt F

kasrakasra kasra kasra kasra kasra
fdghggfhdf810100230

aa aa aa
aa
eqfsfhgjfghfg
aa
eqfsfhgjfghfg
aa
eqfsfhgjfghfg
~
/kasra

```

عکس ۱: نتیجه خروجی

برای اجرای کامند لینوکسی بدون خارج شدن از Vim از `!command` استفاده می کنیم.

```
kasra@Kasra: ~/P1
kasra@Kasra:~/P1$ vim temp1.txt
[No write since last change]
temp1.txt

Press ENTER or type command to continue
[No write since last change]
Sat Apr 19 17:55:53 +0330 2025

Press ENTER or type command to continue|
```

عکس 2: نتیجه خروجی دستور Vim در date

ذخیره، مقایسه و کار با window tab

با استفاده از **W**: می توانیم تغییرات را ذخیره کنیم.

ابتدا **V** برای انتخاب 6 خط متوالی، **U** برای کپی کردن، **W** برای پیست کردن و به مانند بالا سیو **.:w newfile** با نام دلخواه:

ابتدا **V** 15، سپس **y**، **sp**: برای پنجره جدید **Ctrl+w w** برای رفتن به پنجره جدید و **P** برای پیست.

ابتدا **W** برای رفتن به پنجره جدید و سپس **dw** برای پاک کردن کلمه و **CW** برای تغییر کلمه **w for-diff**: برای ذخیره سازی با نام دلخواه.

diffget برای انتقال تغییرات از پنجره چپ به پنجره راست و **diffput** برای انتقال تغییرات از پنجره راست به پنجره چپ.

Vim کانفیگ

برای تعریف کلیدهای میانبر سفارشی Key mapping، در بخش الف دستور گفته شده را اضافه کردیم؛ زین پس می توانیم با **\W** فایل را سیو بکنیم. برای بخش ب هم به همین صورت عمل کردیم.

```

kasra@Kasra: ~/P1      + | -
qfafhgkfgkfg
eqfafhgkfgkfg
dgh
fhrskasrahfghftg

dooroood dorod dooroodd dodododo
d
tthaa mostafa
tt F

temp1.txt          1,1      Top
qfafhgkfgkfg
eqfafhgkfgkfg
dgh
fhrskasrahfghftg

dooroood dorod dooroodd dodododo
d
tthaa mostafa
tt F

temp1.txt          1,1      Top temp1.txt        4,1      All
:/kasra

```

The terminal window shows the contents of the file 'temp1.txt'. The file contains several lines of text, some of which are highlighted in yellow. The lines are:

- qfafhgkfgkfg
- eqfafhgkfgkfg
- dgh
- fhrskasrahfghftg**
- dooroood dorod dooroodd dodododo
- d
- tthaa mostafa
- tt F
- temp1.txt** 1,1 Top
- qfafhgkfgkfg
- eqfafhgkfgkfg
- dgh
- fhrskasrahfghftg**
- dooroood dorod dooroodd dodododo
- d
- tthaa mostafa
- tt F
- jdsjdsndsjdjasra** **kasra** **kasra** **kasra** **kasra**
- fdghggfhdhf810100230**
- dgh nana dg
- fhrskasrahfghftg**
- ~
- ~
- ~
- ~
- :/kasra

عکس 3: نتیجه خروجی دستور این بخش

کانفیگ کردن ادیتور

نمایش خطوط با استفاده از `set ignorecase`, نادیده گرفتن حروف کوچک و بزرگ با استفاده از `set number`, هایلایت کردن را پیش تر اضافه کرده بودیم، سایز قب را تبدیل به 4 کردن: `set shiftwidth=4`, `set tabstop=4` و `syntax enable` و در نهایت مورد آخر با استفاده از `expandtab` و `colorscheme desert` امکان پذیر است.

```

1 qfafhgkfgkfg
2 eqfafhgkfgkfg
3 dgh
4 fhrskasrahfghftg
5
6 dooroood dorod dooroodd dodododo
7 d
8 tthaa mostafa
9 tt F
10 FF FF FF FT gdhdfF
11
12 jdsjdsndsjd j jasrakasra kasra kasra kasra
13
14 fdghggfhdhf 81010 0230
15
16
17 dgh nana dg
18 fhrskasrahfghftg
19
~ ~ ~ ~
:/f          2,1      All

```

The terminal window shows the contents of the file 'temp1.txt' after applying the configuration changes. The file now includes line numbers (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19) and syntax highlighting. The lines are:

- qfafhgkfgkfg
- eqfafhgkfgkfg
- dgh
- fhrskasrahfghftg**
- dooroood dorod dooroodd dodododo
- d
- tthaa mostafa
- tt F
- FF FF FF FT gdhdfF**
- jdsjdsndsjd j jasrakasra kasra kasra kasra**
- fdghggfhdhf 81010 0230**
- dgh nana dg
- fhrskasrahfghftg**
- ~
- ~
- ~
- ~
- ~
- :/f

عکس 4: نتیجه خروجی تنظیمات این بخش

افزودن پلاگین به Vim

در بخش الف، نصب ها انجام شد. برای باز کردن NERDTree در حالت Toggle: برای باز کردن NERDTree: برای پنجره فعلی NERDTreeFind در کنار پنجره فعلی NERDTreeToggle: برای بستن همچنین در بخش ب، میانبر گفته شده ایجاد شد تا با \7 باز و بسته کنیم.

میانبر ها:

O باز کردن فایل یا پوشه انتخاب شده

t باز کردن فایل در تب جدید

A باز کردن فایل در پنجره افقی جدید

S باز کردن فایل در پنجره عمودی جدید

P رفتن به پوشه والد

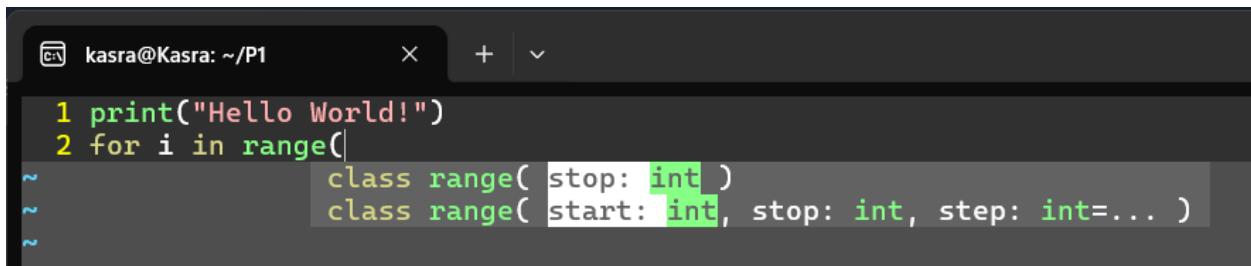
Z تازه سازی پوشه

? نمایش راهنمای کلیدهای میانبر

```
" Press ? for help
.. (up a dir)
/home/kasra/P1/
for-diff
temp1.txt
1 qfafhgkfgkfg
2 eqfafhgkfgkfg
3 dgh
4 fhrskasrahfghftg
5
6 dooroood dorod doorooodd dodododo
7 d
8 tthaa mostafa
9 tt F
10 FF FF FFGFT gdhdFF
11
12 jdsjdsndsjd j jasrakasra kasra kasra kasra
13
14 fdghggfhdhf 81010 0230
15
16
17 dgh nana dg
18 fhrskasrahfghftg
19
```

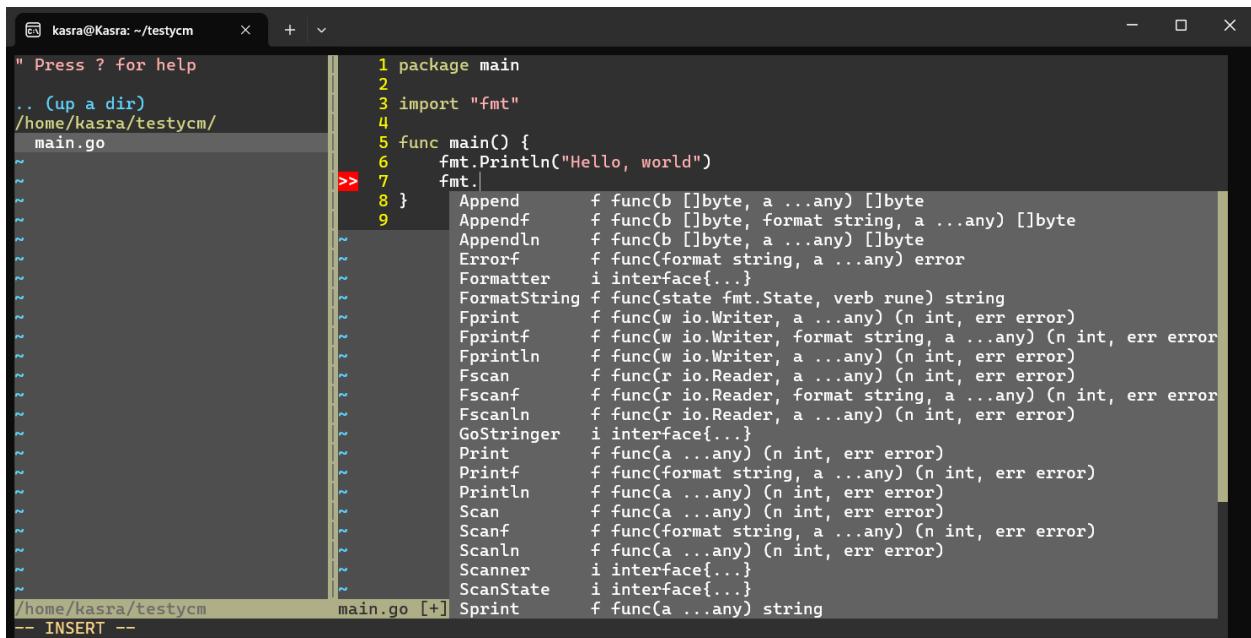
عکس 5: نتیجه خروجی با افزودن پلاگین NERDTree

برای بخش ج هم youcompleteme را در نظر گرفتیم که یک ماشین تکمیل کد می باشد. هنگام نوشتن کد، پس از تایپ چندین حرف از یک کلمه، YouCompleteMe پیشنهادهایی برای تکمیل کلمه نشان می دهد. ما می توانیم با استفاده از کلید Tab پیشنهادات را مرور کنیم و گزینه مورد نظر خود را انتخاب کنیم. YouCompleteMe از زبان های برنامه نویسی Python, JavaScript, C/C++, Python مختلف مانند Go و JavaScript را نصب نمودیم. این پلاگین امکاناتی مشابه IDE ها مانند تکمیل خودکار، پیشنهادات مربوط به توابع، انواع داده ها و پارامترها را در Vim به ارمغان می آورد. وقتی در حال نوشتن یک دستور هستیم، می تواند پیشنهادات متنی و داکیومنت های مرتبط را نمایش دهد.



```
kasra@Kasra: ~/P1
1 print("Hello World!")
2 for i in range(
~         class range( stop: int )
~         class range( start: int, stop: int, step: int=... )
```

عکس 6: نتیجه خروجی با افزودن پلاگین YouCompleteMe برای زبان Python



```
" Press ? for help
.. (up a dir)
/home/kasra/testycm/
main.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, world")
7     fmt._
>> 8 }     Append      f func(b []byte, a ...any) []byte
~ 9 Appendf      f func(b []byte, format string, a ...any) []byte
~ Appendln     f func(b []byte, a ...any) []byte
~ Errorf       f func(format string, a ...any) error
~ Formatter     i interface{...}
~ FormatString f func(state fmt.State, verb rune) string
~ Fprint        f func(w io.Writer, a ...any) (n int, err error)
~ Fprintf       f func(w io.Writer, format string, a ...any) (n int, err error)
~ Fprintln      f func(w io.Writer, a ...any) (n int, err error)
~ Fscan         f func(r io.Reader, a ...any) (n int, err error)
~ Fscanf        f func(r io.Reader, format string, a ...any) (n int, err error)
~ Fscanln      f func(r io.Reader, a ...any) (n int, err error)
~ GoStringer   i interface{...}
~ Print         f func(a ...any) (n int, err error)
~ Printf        f func(format string, a ...any) (n int, err error)
~ Println      f func(a ...any) (n int, err error)
~ Scan          f func(a ...any) (n int, err error)
~ Scanf         f func(format string, a ...any) (n int, err error)
~ Scanln        f func(a ...any) (n int, err error)
~ Scanner       i interface{...}
~ ScanState    i interface{...}
~ Sprint        f func(a ...any) string
/main.go [+] Sprint
```

عکس 7: نتیجه خروجی با افزودن پلاگین YouCompleteMe برای زبان Go به همراه NERDTree

استفاده از ماکروها در Vim

متن را در فایل farsy.txt قرار دادیم و سپس با زدن `qa` و سپس ماکرو نوشته شده و در نهایت زدن `@t` آن را اجرا نمودیم. البته مخالف با صورت سوال این ماکرو را با نام `a` نوشته و اجرا نمودیم، اگر می خواستیم نامش `t` باشد در ابتدا باید را می زدیم.

```

1 Poet: Ferdowsi | Years: 940-1020 | Famous Work: Shahnameh
2 Poet: Omar Khayyam | Years: 1048-1131 | Famous Work: Rubaiyat
3 Poet: Nezami Ganjavi | Years: 1141-1209 | Famous Work: Khamsa
4 Poet: Attar | Years: 1145-1220 | Famous Work: Conference of the Birds
5 Poet: Rumi | Years: 1207-1273 | Famous Work: Masnavi
6 Poet: Saadi | Years: 1210-1292 | Famous Work: Gulistan and Bustan
7 Poet: Hafez | Years: 1315-1390 | Famous Work: Divan of Hafez

```

عکس 8: نتیجه بعد از اجرای ماکرو

سوالات تشریحی

سوال اول: اهمیت استفاده از یک افزونه NERDTree در مدیریت فایل‌های پروژه این است که به کاربران اجازه می‌دهد تا به راحتی به ساختار فایل‌ها و دایرکتوری‌ها دسترسی داشته باشند. در هنگام کار روی پروژه‌های بزرگ، این افزونه می‌تواند به صرفه‌جویی در زمان کمک کند (به مانند ساختار `vs code` که استفاده می‌کنیم می‌باشد)، زیرا شما می‌توانید بدون نیاز به خروج و بازگشت به ترمینال، به سرعت فایل‌ها را جستجو، باز و ویرایش کنید. همچنین، از آنجا که ساختار دایرکتوری‌ها را نمایش می‌دهد، مدیریت فایل‌ها و حرکت میان دایرکتوری‌ها بسیار آسان‌تر می‌شود.

سوال دوم: برخی از افزونه‌ها و تنظیمات می‌توانند ویژگی‌های بیشتری را به Vim اضافه کنند، اما در عین حال ممکن است موجب کاهش سرعت عملکرد شوند. برای مثال، افزونه‌هایی که وظایف سنتی مانند syntax highlighting پیش‌رفته یا autocomplete انجام می‌دهند، ممکن است مصرف منابع را افزایش دهند و باعث کندی شوند. در حالی که تنظیمات ساده‌تری که نیاز به منابع کمی دارند، ممکن است عملکرد سریع‌تر و کارآتری را ارائه دهند. بنابراین، در انتخاب تنظیمات و افزونه‌ها باید تعادلی میان عملکرد و قابلیت‌ها برقرار کرد.

سوال سوم: نقی جان درود، می‌دونم به Nano عادت کردی و خیلی سادست اما Vim، بہت این امکان را می‌دهد که با استفاده از دستورات مختلف، ویرایش متن را به طور بسیار مؤثری انجام بدی. ویژگی‌هایی مثل پشتیبانی از ماکروها، جستجو و جایگزینی پیش‌رفته، پشتیبانی از حالت‌های مختلف Insert Mode و Command Mode، و قابلیت‌های سفارشی‌سازی پیش‌رفته Vim این امکان را می‌دهند که در پروژه‌های پیچیده و در مواجهه با حجم بالای داده‌ها، بسیار قدرتمندتر از Nano عمل کند. به عنوان مثال، در Vim می‌توانی تنها با چند دستور ساده تغییرات زیادی را در فایل‌ها اعمال کنی، در حالی که در Nano این امر به طور قابل توجهی زمان بیشتری می‌برد. پس یبار دیگه فکر کن و یکم یوتوب رو شخم بزن شاید نظرت عوض شد!

سوال چهارم: مزایای استفاده از ماکرو در Vim این است که ماکروها به ما این امکان را می‌دهند که یک سری از دستورات ویرایشی را ضبط کرده و سپس آن‌ها را در هر جایی از فایل یا پروژه دوباره اجرا کنیم. این ویژگی به ویژه زمانی مفید است که بخوایم یک سری عملیات مشابه را روی بخش‌های مختلف متن انجام بدیم. به جای تکرار دستی دستورات، می‌توانیم ماکروها را ضبط کنیم و با استفاده از یک کلید میانبر یا دستور آن‌ها را دوباره اجرا کنیم. این کار می‌تواند باعث صرفه‌جویی در زمان و کاهش خطای انسانی در عملیات‌های تکراری شود.

سوال پنجم: مشکلاتی که ممکن است حین Recording و Replaying مکروها ایجاد شوند شامل تغییرات ناخواسته در موقعیت یا متن بین دو اجرای ماکرو است. اگر حین ضبط ماکرو، موقعیت‌تان تغییر کند یا متن به گونه‌ای تغییر کند که ساختار آن با زمانی که ماکرو ضبط شده تطابق نداشته باشد، ممکن است اجرای مجدد ماکرو نتایج غیرمنتظره‌ای را به دنبال داشته باشد. این اتفاق خصوصاً در مورد ویرایش‌های پیچیده‌تر که شامل جابجایی‌های متعدد یا تغییرات گسترده در فایل‌ها می‌شود، می‌تواند مشکل‌ساز شود. در قسمت انجام ماکرو‌ها چون که حواسمن به این موضوع بود به این مشکل بر نخوردیم.

سوال ششم: ویرایشگرهای Modeless به این معنا هستند که کاربر می‌تواند بدون تغییر حالت‌ها و با یک جریان پیوسته از دستورات و تایپ، کار کند. در این ویرایشگرهای، هر کلیدی که فشرده شود، به یک دستور مشخص تبدیل می‌شود. از طرف دیگر، ویرایشگرهای Modal از حالت‌های مختلف برای انجام عملیات‌های مختلف استفاده می‌کنند. در حالت Insert کاربر می‌تواند متنی وارد کند، در حالی که در حالت Command Mode دستورات مختلفی برای ویرایش متن وارد می‌شود.

سوال هفتم: Vim سه حالت اصلی دارد: Normal Mode، Insert Mode و Visual. در حالت Normal Mode می‌توانیم متنی را وارد کنیم. در Normal Mode می‌توانیم دستورات مختلفی برای جستجو، جابجایی، حذف و ویرایش متن وارد کنیم. در Visual Mode می‌توانیم متنی را انتخاب کرده و سپس آن را کپی، کات یا جایگزین کنیم. این حالت‌ها به ما این امکان را می‌دهند که به طور مؤثری از Vim برای ویرایش فایل‌ها استفاده کنیم. نحوه رفتنه به هر Mode ای و خارج شدن از آن هم در عکس صورت پروژه آمده است.