ლინუქსის ფაილური სისტემა Ubuntu 18.04-სთვის

სარჩევი

შესავალი

ეს სახელმძღვანელო განკუთვნილია სტუდენტებისთვის, რომლებიც სწავლობენ Linux-ის ფაილურ სისტემას Ubuntu 18.04 ოპერაციულ სისტემაში. დოკუმენტი დეტალურად განმარტავს ფაილური სისტემის სტრუქტურას და, რაც ყველაზე მნიშვნელოვანია, ასახსნელებს თუ რომელი დირექტორიები რეალურად არსებობს მყარ დისკზე და რომლები არიან ვირტუალური (ოპერატიულ მეხსიერებაში).

რატომ არის ეს მნიშვნელოვანი:

- გესმით სად ინახება თქვენი მონაცემები რეალურად
- იცით რა ინფორმაცია გაქრება კომპიუტერის გამორთვისას
- გესმით როგორ მუშაობს Linux-ის ფილოსოფია "ყველაფერი ფაილია"
- შეძლებთ უკეთ გაიგოთ სისტემის არქიტექტურა

წინაპირობები:

- Ubuntu 18.04 დაინსტალირებული კომპიუტერი
- ტერმინალის გახსნის უნარი (Ctrl + Alt + T)
- ძირითადი ბრძანებების ცოდნა (არაა აუცილებელი, ყველაფერს ახსნით დოკუმენტში)

---

1. "ყველაფერი ფაილია" - ძირითადი კონცეფცია

1.1. რას ნიშნავს ეს პრინციპი

Linux-ის ერთ-ერთი ძირითადი ფილოსოფია არის: **"ყველაფერი ფაილია"** (Everything is a file).

ეს ნიშნავს, რომ Linux ყველაფერს, რასთანაც თქვენ მუშაობთ კომპიუტერში, წარმოადგენს როგორც ფაილს ან დირექტორიას. ეს აბსტრაქცია საშუალებას გაძლევთ ერთნაირად იმუშაოთ სხვადასხვა ტიპის რესურსებთან.

რატომ არის ეს მძლავრი:

- ერთი და იგივე ბრძანებები მუშაობს ფაილებზე, მოწყობილობებზე, პროცესებზე
- მარტივი და თანმიმდევრული ინტერფეისი
- ყველაფერზე შეგიძლიათ იმუშაოთ სტანდარტული ბრძანებებით (cat, echo, ls, და ა.შ.)

1.2. რა წარმოდგენილია ფაილების სახით

ტრადიციული ფაილები:

- 📄 ტექსტური დოკუმენტები (file.txt)
- 🖼️ სურათები (photo.jpg)
- 🎵 მუსიკა (song.mp3)
- 📼 ვიდეოები (movie.mp4)
- 📁 დირექტორიები (საქაღალდეები)

არატრადიციული "ფაილები":

- 🖨️ პრინტერი → `/dev/lp0`
- ⌨️ კლავიატურა → `/dev/input/event0`
- 💾 მყარი დისკი → `/dev/sda`

- 🖱️ მაუსი → `/dev/input/mouse0`
- 🔊 აუდიო → `/dev/snd/`
- 🌐 ქსელური კავშირი → `/proc/net/`
- 🔄 გაშვებული პროცესი → `/proc/1234/`
- 💻 CPU ინფორმაცია → `/proc/cpuinfo`
- 🔋 ბატარეის სტატუსი → `/sys/class/power_supply/`

მაგალითი:

```
# ჩვეულებრივი ფაილის კითხვა
cat document.txt

# პროცესორის ინფორმაციის "კითხვა"
cat /proc/cpuinfo

# ორივე შემთხვევაში იგივე ბრძანება - cat!
```

---

2. ფაილური სისტემის სტრუქტურა

2.1. ხის იერარქია

Linux-ის ფაილური სისტემა ორგანიზებულია როგორც **ხის სტრუქტურა**, რომელსაც აქვს ერთი ფესვი (root) და მისგან განშტოებული ტოტები.

ვიზუალური წარმოდგენა:

```
                / (ფესვი)
                |
     +----------+----------+
     |          |          |
   home/      etc/       var/
     |          |          |
  student/   passwd     log/
     |                     |
Documents/              syslog
```

2.2. ფესვის დირექტორია (`/`)

`/` (forward slash) არის **ფესვის დირექტორია** - ფაილური სისტემის უმაღლესი დონე.

მნიშვნელოვანი:

- Windows-ში ყოველ დისკს აქვს თავისი ფესვი: C:, D:, E:\
- Linux-ში არის **მხოლოდ ერთი ფესვი** `/`, და ყველა დისკი, USB, DVD ერთიანდება ამ ხის ქვეშ

მაგალითი:

```
# ფესვში გადასვლა
cd /

# ფესვის შიგთავსის ნახვა
ls /
```

2.3. ძირითადი დირექტორიები

ფესვის დირექტორიაში (`/`) გვხვდება შემდეგი ძირითადი საქაღალდეები:

```
/  (ფესვი - ROOT)
│
├── home/        👤 მომხმარებლების პირადი ფაილები
├── bin/         ⚙️ ძირითადი პროგრამები (ls, cp, mv)
├── sbin/        🔧 სისტემური ადმინისტრირების ბრძანებები
├── boot/        🚀 ჩატვირთვისთვის საჭირო ფაილები
├── dev/         🔌 მოწყობილობების ფაილები
├── etc/         ⚙️ კონფიგურაციის ფაილები
├── lib/         📚 ბიბლიოთეკები და მოდულები
├── media/       💿 გარე მოწყობილობები (USB, DVD)
├── mnt/         📌 დროებითი მონტირების ადგილი
├── opt/         📦 დამატებითი პროგრამები
├── proc/        🔄 პროცესების ინფორმაცია (ვირტუალური!)
├── root/        👑 root მომხმარებლის საქაღალდე
├── run/         🏃 გაშვებული პროცესების მონაცემები (ვირტუალური!)
├── srv/         🌐 სერვისების მონაცემები
├── sys/         💻 სისტემური ინფორმაცია (ვირტუალური!)
├── tmp/         🗑️ დროებითი ფაილები
├── usr/         👥 მომხმარებლის პროგრამები და ბიბლიოთეკები
└── var/         📊 ცვლადი მონაცემები (ლოგები, კეში)
```

ყურადღება: ზოგიერთი დირექტორია მონიშნულია როგორც "ვირტუალური" - მათ შესახებ დეტალურად ვისაუბრებთ მე-4 თავში.

---

3. ფიზიკურ დისკზე არსებული დირექტორიები

ეს დირექტორიები **რეალურად** ინახება თქვენს მყარ დისკზე (HDD ან SSD). მათში შენახული ინფორმაცია **შენარჩუნდება** კომპიუტერის გამორთვის შემდეგაც.

3.1. `/home/` - მომხმარებლების საქაღალდეები

🟢 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** მომხმარებლების პირადი ფაილები

სტრუქტურა:

```
/home/
├── student1/
│   ├── Documents/        # დოკუმენტები
│   ├── Downloads/        # ჩამოტვირთული ფაილები
│   ├── Desktop/          # სამუშაო მაგიდა
│   ├── Pictures/         # სურათები
│   ├── Videos/           # ვიდეოები
│   ├── Music/            # მუსიკა
│   ├── .bashrc           # ფარული კონფიგურაციის ფაილი
│   └── .bash_history     # ბრძანებების ისტორია
├── student2/
│   └── ...
└── student3/
    └── ...
```

პრაქტიკული მაგალითები:

```
# თქვენს საწყის დირექტორიაში გადასვლა
cd ~
# ან
cd /home/studentname/

# ყველა ფაილის ნახვა (ფარულების ჩათვლით)
ls -la

# თქვენი დოკუმენტების ნახვა
ls ~/Documents/
```

რატომ არის დისკზე: თქვენი პირადი მონაცემები (ფოტოები, დოკუმენტები, ფაილები) უნდა შენარჩუნდეს კომპიუტერის გამორთვის შემდეგაც. წარმოიდგინეთ თუ ყოველ ჯერზე გადატვირთვისას წაიშლება ყველაფერი!

3.2. `/etc/` - კონფიგურაციის ფაილები

🔘 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** სისტემისა და პროგრამების კონფიგურაციის ფაილები

მნიშვნელოვანი ფაილები:

```
/etc/
├── passwd              # მომხმარებლების სია
├── shadow              # პაროლები (დაშიფრული)
├── group               # ჯგუფების ინფორმაცია
├── fstab               # ფაილური სისტემების მონტირების ცხრილი
├── hostname            # კომპიუტერის სახელი
├── hosts               # IP მისამართების ლოკალური ბაზა
├── network/            # ქსელის კონფიგურაცია
├── ssh/                # SSH სერვერის კონფიგურაცია
├── apache2/            # Apache ვებ-სერვერის პარამეტრები
└── apt/                # პაკეტების მენეჯერის კონფიგურაცია
```

პრაქტიკული მაგალითები:

```
# მომხმარებლების სიის ნახვა
cat /etc/passwd

# კომპიუტერის სახელის ნახვა
cat /etc/hostname

# ქსელის კონფიგურაციის რედაქტირება (საჭიროებს sudo)
sudo nano /etc/network/interfaces
```

რატომ არის დისკზე: სისტემის კონფიგურაცია (როგორ უნდა იმუშაოს ქსელი, რა პარამეტრები აქვს სერვერს, და ა.შ.) უნდა შენარჩუნდეს გადატვირთვის შემდეგაც. წინააღმდეგ შემთხვევაში, ყოველ ჯერზე თავიდან უნდა გაკონფიგურიროთ ყველაფერი!

3.3. `/bin/` და `/sbin/` - პროგრამები

🔘 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** სისტემური პროგრამები და ბრძანებები

`/bin/` - **Binary (ძირითადი ბრძანებები):**

```
/bin/
├── ls          # ფაილების სიის ჩვენება
├── cp          # ფაილის კოპირება
├── mv          # ფაილის გადატანა
├── rm          # ფაილის წაშლა
├── cat         # ფაილის წაკითხვა
├── mkdir       # დირექტორიის შექმნა
├── bash        # Bash shell პროგრამა
├── grep        # ტექსტის ძებნა
└── echo        # ტექსტის გამოტანა
```

`/sbin/` - System Binary (ადმინისტრატორის ბრძანებები):

```
/sbin/
├── ifconfig    # ქსელის კონფიგურაცია
├── reboot      # გადატვირთვა
├── shutdown    # გამორთვა
├── fdisk       # დისკების პარტიციონირება
├── mount       # ფაილური სისტემის მონტირება
└── iptables    # firewall კონფიგურაცია
```

განსხვავება:

- `/bin/` - ყველა მომხმარებელს შეუძლია გამოიყენოს
- `/sbin/` - მხოლოდ root (ადმინისტრატორს) ან sudo-თი

პრაქტიკული მაგალითები:

```
# ნახეთ რა პროგრამებია /bin/-ში
ls /bin/

# ნახეთ სად მდებარეობს ls პროგრამა
which ls
# გამომავალი: /bin/ls

# ნახეთ რა ბრძანებებია /sbin/-ში
ls /sbin/
```

რატომ არის დისკზე: სისტემური პროგრამები მუდმივად უნდა იყოს ხელმისაწვდომი. თუ ისინი RAM-ში იქნებოდა, ყოველ გამორთვაზე წაიშლებოდა და სისტემა ვერ ჩაიტვირთებოდა!

3.4. `/boot/` - ჩატვირთვის ფაილები

🟤 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** ოპერაციული სისტემის ჩასატვირთად საჭირო ფაილები

სტრუქტურა:

```
/boot/
├── vmlinuz-4.15.0-XX-generic     # Linux kernel (ბირთვი)
├── initrd.img-4.15.0-XX-generic # საწყისი RAM disk
├── config-4.15.0-XX-generic      # kernel-ის კონფიგურაცია
├── System.map-4.15.0-XX-generic # სიმბოლური ცხრილი
└── grub/                          # GRUB ჩატვირთვის მენეჯერი
    ├── grub.cfg                  # GRUB კონფიგურაცია
    └── fonts/                    # შრიფტები
```

რა არის Kernel: Linux kernel არის ოპერაციული სისტემის "გული" - პროგრამა რომელიც მართავს აპარატურას და რესურსებს.

პრაქტიკული მაგალითები:

```
# ნახეთ რა ფაილებია /boot/-ში
ls -lh /boot/

# ნახეთ kernel-ის ვერსია
uname -r
# გამომავალი: 4.15.0-213-generic

# ნახეთ GRUB-ის კონფიგურაცია
sudo cat /boot/grub/grub.cfg
```

რატომ არის დისკზე: ჩატვირთვის დროს კომპიუტერს სჭირდება ამ ფაილები **დისკიდან** წასაკითხად, რადგან RAM ჯერ კიდევ ცარიელია! კომპიუტერი პირველ რიგში კითხულობს `/boot/` -ს და ატვირთავს Linux kernel-ს მეხსიერებაში.

3.5. `/lib/` - ბიბლიოთეკები

🟤 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** პროგრამების საერთო კოდის ბიბლიოთეკები

რა არის ბიბლიოთეკა: ბიბლიოთეკა არის კოდის ნაწილი, რომელსაც ბევრი პროგრამა იზიარებს. ეს თავიდან აიცილებს კოდის დუბლირებას.

სტრუქტურა:

```
/lib/
├── x86_64-linux-gnu/    # 64-ბიტიანი ბიბლიოთეკები
│   ├── libc.so.6        # C სტანდარტული ბიბლიოთეკა
│   ├── libm.so.6        # მათემატიკური ბიბლიოთეკა
│   └── libpthread.so.0  # ძაფების (threads) ბიბლიოთეკა
└── modules/             # Kernel მოდულები (დრაივერები)
    └── 4.15.0-XX-generic/
```

მაგალითი: წარმოიდგინეთ 10 პროგრამას სჭირდება "დათვლის" ფუნქცია. ეს ფუნქცია ინახება `/lib/`-ში და ყველა პროგრამა იყენებს მას, ნაცვლად იმისა რომ თითოეულმა თავისი კოპია ჰქონდეს.

რატომ არის დისკზე: პროგრამებს მუდმივად სჭირდებათ ბიბლიოთეკები მუშაობისთვის. ისინი უნდა იყოს ხელმისაწვდომი სისტემის მუშაობის მთელი პერიოდის განმავლობაში.

3.6. `/usr/` - მომხმარებლის პროგრამები

🟢 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** დაინსტალირებული პროგრამები და მათი ბიბლიოთეკები

სტრუქტურა:

```
/usr/
├── bin/          # დამატებითი პროგრამები (python, gcc, firefox)
├── sbin/         # დამატებითი ადმინისტრატორის ბრძანებები
├── lib/          # პროგრამების ბიბლიოთეკები
├── include/      # C/C++ header ფაილები
├── share/        # საერთო მონაცემები
│   ├── doc/      # დოკუმენტაცია
│   ├── man/      # სახელმძღვანელოები (manual pages)
│   ├── icons/    # ხატულები
│   └── fonts/    # შრიფტები
└── local/        # ლოკალურად დაინსტალირებული პროგრამები
    ├── bin/
    └── lib/
```

განსხვავება `/bin/` და `/usr/bin/` შორის:

- `/bin/` - აუცილებელი სისტემური ბრძანებები
- `/usr/bin/` - დამატებითი პროგრამები

პრაქტიკული მაგალითები:

```
# ნახეთ რა პროგრამებია დაინსტალირებული
ls /usr/bin/

# Python-ის მდებარეობა
which python3
# გამომავალი: /usr/bin/python3

# დოკუმენტაციის წაკითხვა
man ls
# ეს კითხულობს /usr/share/man/-დან
```

რატომ არის დისკზე: თქვენ ინსტალირებული პროგრამები (Firefox, LibreOffice, Python, და ა.შ.) უნდა შენარჩუნდეს. წინააღმდეგ შემთხვევაში ყოველ გადატვირთვაზე თავიდან მოგიწევდათ ყველაფრის ინსტალაცია!

3.7. `/var/` - ცვლადი მონაცემები

🟢 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** ცვლადი და დინამიური მონაცემები

სტრუქტურა:

```
/var/
├── log/                    # ლოგ ფაილები
│   ├── syslog              # სისტემის ლოგი
│   ├── auth.log            # ავთენტიფიკაციის ლოგი
│   ├── kern.log            # kernel-ის ლოგი
│   └── apache2/            # ვებ-სერვერის ლოგები
├── cache/                  # კეში მონაცემები
│   └── apt/                # პაკეტების კეში
├── tmp/                    # დროებითი ფაილები
├── www/                    # ვებსაიტის ფაილები
│   └── html/               # HTML ფაილები
├── mail/                   # ელექტრონული ფოსტა
└── spool/                  # დროებითი სამუშაო ფაილები
    └── cron/               # დაგეგმილი ამოცანები
```

პრაქტიკული მაგალითები:

```
# სისტემის ლოგების ნახვა
tail -f /var/log/syslog
# ნახავთ რეალურ დროში რა ხდება სისტემაში

# ლოგებში შეცდომების ძებნა
grep -i "error" /var/log/syslog

# რამდენ ადგილს იკავებს /var/
du -sh /var/
```

რატომ არის დისკზე: ლოგ ფაილები გადამწყვეტი მნიშვნელობისაა პრობლემების დიაგნოსტიკისთვის. თუ სისტემა დაიქრაშა და გადატვირთდა, ადმინისტრატორს სჭირდება ძველი ლოგები იმის გასაგებად რა მოხდა. ამიტომ ლოგები უნდა შენარჩუნდეს დისკზე.

3.8. `/opt/` - დამატებითი პროგრამები

🟢 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** გარე კომპანიების დიდი პროგრამული პაკეტები

მაგალითები:

```
/opt/
├── google/
│   └── chrome/          # Google Chrome
├── skype/               # Skype
├── lampp/               # XAMPP (Apache, MySQL, PHP პაკეტი)
└── teamviewer/          # TeamViewer
```

რატომ განსხვავებული დირექტორია: ზოგიერთი კომპანია ამჯობინებს თავისი პროგრამები ერთ ადგილას იყოს შეკრული, `/usr/`-ში გაფანტვის ნაცვლად.

რატომ არის დისკზე: ისევე როგორც `/usr/`, ეს არის დაინსტალირებული პროგრამები რომლებიც უნდა შენარჩუნდეს.

3.9. `/root/` - ადმინისტრატორის საქაღალდე

🟢 **მდებარეობა:** მყარ დისკზე
📄 **რა ინახება:** root მომხმარებლის პირადი ფაილები

⚠️ **არ აგვერიოთ:**

- `/` (ფესვის დირექტორია) - მთელი სისტემის სათავე
- `/root/` - root მომხმარებლის home დირექტორია

სტრუქტურა:

```
/root/
├── .bashrc              # root-ის shell კონფიგურაცია
├── .bash_history        # root-ის ბრძანებების ისტორია
└── scripts/             # ადმინისტრატორის სკრიპტები
```

რატომ არის განცალკევებული: root არის სისტემის ადმინისტრატორი და მისი ფაილები განცალკევებულია უსაფრთხოებისთვის.

რატომ არის დისკზე: ადმინისტრატორის კონფიგურაციები და სკრიპტები უნდა შენარჩუნდეს გადატვირთვის შემდეგაც.

---

4. ვირტუალური დირექტორიები (RAM-ში)

ეს დირექტორიები **არ არსებობს** ფიზიკურად მყარ დისკზე! ისინი იქმნება **ოპერატიული მეხსიერებაში (RAM)** სისტემის ჩატვირთვისას და **ქრებიან** გამორთვისას.

🟢 **მთავარი პრინციპი:** ეს დირექტორიები არიან "ფანჯრები" სისტემის მიმდინარე მდგომარეობაში.

4.1. `/proc/` - პროცესების ინფორმაცია

🟢 **მდებარეობა:** RAM-ში (არა დისკზე!)
📄 **რა ინახება:** პროცესების და სისტემის რეალურ დროში ინფორმაცია

სტრუქტურა:

```
/proc/
├── 1/                  # პროცესი PID=1 (init/systemd)
│   ├── cmdline         # როგორ გაეშვა პროცესი
│   ├── status          # პროცესის სტატუსი
│   ├── mem             # მეხსიერების გამოყენება
│   └── fd/             # ღია ფაილების დესკრიპტორები
├── 1234/               # პროცესი PID=1234
├── 5678/               # პროცესი PID=5678
├── cpuinfo             # პროცესორის ინფორმაცია
├── meminfo             # მეხსიერების ინფორმაცია
├── version             # kernel-ის ვერსია
├── uptime              # რამდენ ხანია ჩართული სისტემა
├── loadavg             # სისტემის დატვირთვა
├── filesystems         # მხარდაჭერილი ფაილური სისტემები
└── net/                # ქსელის ინფორმაცია
    ├── dev             # ქსელური ინტერფეისები
    └── route           # მარშრუტიზაციის ცხრილი
```

პრაქტიკული მაგალითები:

```
# პროცესორის სრული ინფორმაცია
cat /proc/cpuinfo
# ნახავთ: მოდელი, ბირთვების რაოდენობა, სიჩქარე, cache

# მეხსიერების ინფორმაცია
cat /proc/meminfo
# ნახავთ: სულ რამდენია RAM, რამდენი თავისუფალია, რამდენი გამოიყენება

# რამდენ ხანია ჩართული სისტემა
cat /proc/uptime
# პირველი რიცხვი = წამები ჩართვიდან

# ქსელის სტატისტიკა
cat /proc/net/dev
# ნახავთ რამდენი ბაიტი გაიგზავნა/მიიღო თითოეულმა ინტერფეისმა

# kernel-ის ვერსია
cat /proc/version

# სისტემის დატვირთვა (load average)
cat /proc/loadavg
```

🔬 ექსპერიმენტი - პროცესების დინამიკა:

```
# ტერმინალი 1:
# ნახეთ რამდენი პროცესია ახლა
ls /proc/ | grep -E '^[0-9]+$' | wc -l
# დაითვალეთ რიცხვი

# ტერმინალი 2:
# გახსენით Firefox
firefox &

# ტერმინალი 1:
# ხელახლა ნახეთ პროცესები
ls /proc/ | grep -E '^[0-9]+$' | wc -l
# ახლა მეტია! Firefox-მა ახალი პროცესები შექმნა

# იპოვეთ Firefox-ის PID
ps aux | grep firefox
# დაიმახსოვრეთ PID (მაგ: 12345)

# ნახეთ Firefox-ის ინფორმაცია
cat /proc/12345/cmdline
ls /proc/12345/
```

რატომ არის RAM-ში: პროცესები იქმნებიან და ქრებიან ყოველ წამს. ეს ინფორმაცია იცვლება რეალურ დროში და საჭიროებს სწრაფ წვდომას. RAM ბევრად სწრაფია ვიდრე დისკი!

რა მოხდება გამორთვისას: ❌ **სრულად გაქრება!** პროცესები ჩერდება, `/proc/` ცარიელდება.

4.2. `/sys/` - სისტემური და აპარატურის ინფორმაცია

🔘 **მდებარეობა:** RAM-ში (არა დისკზე!)
📄 **რა ინახება:** აპარატურის და kernel-ის პარამეტრები

სტრუქტურა:

```
/sys/
├── block/                # დისკები და პარტიციები
│   ├── sda/
│   │   ├── size          # დისკის ზომა (სექტორებში)
│   │   ├── removable     # მოქრევადია? (0=არა, 1=კი)
│   │   └── queue/        # I/O პარამეტრები
│   └── sdb/              # USB stick (თუ ჩართულია)
├── class/
│   ├── net/              # ქსელური ინტერფეისები
│   │   ├── eth0/         # Ethernet
│   │   └── wlan0/        # WiFi
│   ├── power_supply/     # ელკვება და ბატარეა
│   │   └── BAT0/
│   │       ├── capacity      # ბატარეის პროცენტი
│   │       ├── status        # Charging/Discharging
│   │       └── voltage_now   # ვოლტაჟი
│   └── backlight/        # ეკრანის სიკაშკაშე
│       └── intel_backlight/
│           ├── brightness    # მიმდინარე სიკაშკაშე
│           └── max_brightness
├── devices/              # ყველა მოწყობილობა
└── kernel/               # kernel-ის პარამეტრები
```

პრაქტიკული მაგალითები:

```
# დისკის ზომა
cat /sys/block/sda/size
# გამომავალი: რიცხვი სექტორებში
# ბაიტებში გადაყვანა: რიცხვი * 512

# დისკი მოქრევადია?
cat /sys/block/sda/removable
# 0 = არა (მყარი დისკი)
# 1 = კი (USB stick)

# ბატარეის მდგომარეობა
cat /sys/class/power_supply/BAT0/capacity
# მაგ: 85 (ნიშნავს 85%)

cat /sys/class/power_supply/BAT0/status
# Charging - იტენება
# Discharging - იხარჯება
# Full - სავსეა

# ეკრანის სიკაშკაშის ნახვა
cat /sys/class/backlight/intel_backlight/brightness

# ეკრანის სიკაშკაშის შეცვლა (საჭიროებს root უფლებებს)
echo 500 | sudo tee /sys/class/backlight/intel_backlight/brightness

# ქსელური ინტერფეისის მდგომარეობა
cat /sys/class/net/eth0/operstate
# up - ჩართულია და მუშაობს
# down - გამორთულია

# CPU-ს მიმდინარე სიხშირე
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

🔬 ექსპერიმენტი - USB-ს მონიტორინგი:

```
# ჩართვამდე:
ls /sys/class/block/
# ნახავთ: sda, sda1, sda2 (თქვენი მყარი დისკი)

# ჩართეთ USB stick

# ხელახლა ნახეთ:
ls /sys/class/block/
# ახლა დამატებით: sdb, sdb1 (USB)

# USB-ს ინფორმაცია:
cat /sys/class/block/sdb/size          # ზომა
cat /sys/class/block/sdb/removable     # 1 = მოქრევადია
```

რატომ არის RAM-ში: `/sys/` გვიჩვენებს აპარატურის **მიმდინარე მდგომარეობას**. ბატარეის დონე, CPU-ს სიხშირე, ქსელის სტატუსი - ყველაფერი იცვლება რეალურ დროში. Kernel აწარმოებს ამ ინფორმაციას "ფაილების" სახით RAM-ში.

რა მოხდება გამორთვისას: ❌ **გაქრება!** გადატვირთვისას ხელახლა შეიქმნება ახალი მონაცემებით.

4.3. `/dev/` - მოწყობილობების ფაილები

🔘 **მდებარეობა:** RAM-ში (არა დისკზე!)
📄 **რა ინახება:** სპეციალური ფაილები რომლებიც წარმოადგენენ მოწყობილობებს

სტრუქტურა:

```
/dev/
├── sda             # პირველი მყარი დისკი
├── sda1            # პირველი პარტიცია
├── sda2            # მეორე პარტიცია
├── sdb             # მეორე დისკი (USB)
├── tty1            # პირველი ვირტუალური კონსოლი
├── pts/            # ვირტუალური ტერმინალები
│   ├── 0           # პირველი ტერმინალი
│   └── 1           # მეორე ტერმინალი
├── null            # "შავი ხვრელი" - ყველაფერს ყლაპავს
├── zero            # უსასრულო ნულები
├── random          # შემთხვევითი ბაიტები
├── urandom         # შემთხვევითი ბაიტები (სწრაფი)
├── stdin           # სტანდარტული შემომავალი (0)
├── stdout          # სტანდარტული გამომავალი (1)
├── stderr          # შეცდომების გამომავალი (2)
└── input/          # შეტანის მოწყობილობები
    ├── mouse0      # მაუსი
    └── event0      # კლავიატურა
```

ფაილების ტიპები:

```
$ ls -l /dev/sda /dev/null /dev/pts/0
brw-rw---- 1 root disk   8, 0 Jan 18 10:00 /dev/sda
crw-rw-rw- 1 root root   1, 3 Jan 18 10:00 /dev/null
crw--w---- 1 user tty  136, 0 Jan 18 10:00 /dev/pts/0
 ↑
 ├─ b = block device (დისკები)
 └─ c = character device (ტერმინალები, სერიული პორტები)
```

დისკების სქემა:

```
/dev/sda     = მთელი პირველი დისკი
/dev/sda1    = პირველი პარტიცია
/dev/sda2    = მეორე პარტიცია
/dev/sda5    = მეხუთე პარტიცია (logical)

/dev/sdb     = მეორე დისკი (USB stick)
/dev/sdb1    = USB-ს პარტიცია
```

პრაქტიკული მაგალითები:

```
# დისკების სია
ls -l /dev/sd*
# ნახავთ ყველა მყარ დისკს და პარტიციას

# დისკის დეტალური ინფორმაცია
sudo fdisk -l /dev/sda

# შემთხვევითი 10 ბაიტი
cat /dev/random | head -c 10 | xxd

# შემთხვევითი პაროლის გენერაცია
cat /dev/urandom | tr -dc 'A-Za-z0-9' | head -c 16
# გამომავალი: 16 სიმბოლოიანი შემთხვევითი პაროლი

# ყველაფრის "გადაგდება" შავ ხვრელში
echo "ეს არ გამოჩნდება" > /dev/null
ls /nonexistent 2> /dev/null  # შეცდომების დამალვა

# უსასრულო ნულები
head -c 100 /dev/zero | xxd
# 100 ნულოვანი ბაიტი

# თქვენი ტერმინალის იდენტიფიკაცია
tty
# მაგ: /dev/pts/0

# სხვა ტერმინალში წერა
# ტერმინალი 1:
tty
# გამომავალი: /dev/pts/0

# ტერმინალი 2:
echo "გამარჯობა!" > /dev/pts/0
# ტექსტი გამოჩნდება ტერმინალი 1-ში!
```

🔬 ექსპერიმენტი - /dev/null გაგება:

```
# 1. შექმენით ფაილი
echo "მნიშვნელოვანი ინფორმაცია" > important.txt
cat important.txt
# ნახავთ: მნიშვნელოვანი ინფორმაცია

# 2. "გადააგდეთ" /dev/null-ში
cat important.txt > /dev/null
# არაფერი არ გამოჩნდება!

# 3. ფაილი კვლავ არსებობს
cat important.txt
# ნახავთ: მნიშვნელოვანი ინფორმაცია

# 4. /dev/null ცარიელია
cat /dev/null
# არაფერი

# 5. პრაქტიკული გამოყენება
find / -name "*.txt" 2> /dev/null
# ეძებს .txt ფაილებს, შეცდომებს "ყლაპავს"
```

როგორ მუშაობს:

1. Kernel ქმნის `/dev/` დირექტორიას RAM-ში
2. `udev` სისტემა **ავტომატურად** ქმნის მოწყობილობების ფაილებს
3. როცა USB ჩაერთვება → ახალი ფაილი `/dev/sdb` ჩნდება
4. როცა USB გამოერთვება → ფაილი ქრება

რატომ არის RAM-ში: მოწყობილობები ერთვება და გამოერთვება დინამიურად. USB-ს ჩართვისას მაშინვე უნდა გამოჩნდეს `/dev/sdb`, ამიტომ `/dev/` უნდა იყოს მოქნილი და სწრაფი - RAM იდეალურია ამისთვის.

რა მოხდება გამორთვისას: ❌ **გაქრება!** (მაგრამ გადატვირთვისას ხელახლა შეიქმნება)

4.4. `/run/` - გაშვებული პროცესების მონაცემები

🟤 **მდებარეობა:** RAM-ში (tmpfs)
📄 **რა ინახება:** მიმდინარე სესიის რან-თაიმ მონაცემები

სტრუქტურა:

```
/run/
├── user/
│   └── 1000/            # User ID 1000-ის სესია
│       ├── systemd/     # systemd სესიის მონაცემები
│       └── pulse/       # PulseAudio
├── lock/                # Lock ფაილები
├── utmp                 # ვინ ჩართულია ახლა
├── systemd/             # systemd-ს მონაცემები
├── nginx.pid            # Nginx-ის Process ID
├── apache2.pid          # Apache-ის Process ID
└── udev/                # udev-ის დროებითი მონაცემები
```

პრაქტიკული მაგალითები:

```
# ვინ ჩართულია სისტემაში ახლა
who
# ან
w

# თქვენი User ID
id -u
# მაგ: 1000 (ჩვეულებრივ პირველი მომხმარებელი)

# გაშვებული სერვისების PID ფაილები
ls /run/*.pid
# nginx.pid, sshd.pid, და ა.შ.

# PID-ის წაკითხვა
cat /run/nginx.pid
# გამომავალი: 1234 (Nginx პროცესის ID)
```

რატომ არის RAM-ში: ეს მონაცემები საჭიროა მხოლოდ **მიმდინარე სესიისთვის**. გადატვირთვის შემდეგ ახალი PID-ები მიიღება, ძველები აღარ არის აქტუალური.

რა მოხდება გამორთვისას: ❌ **გაქრება!**

4.5. `/tmp/` - დროებითი ფაილები

🟡 **მდებარეობა:** დამოკიდებულია კონფიგურაციაზე!
📄 **რა ინახება:** დროებითი ფაილები

⚠️ **Ubuntu 18.04-ში:** `/tmp/` შეიძლება იყოს **RAM-ში (tmpfs)** ან **დისკზე** - დამოკიდებულია სისტემის კონფიგურაციაზე.

როგორ შევამოწმოთ:

```
# შემოწმება სად არის /tmp/
df -h /tmp/

# თუ გამომავალი შეიცავს "tmpfs":
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           3.9G  120M  3.8G   4% /tmp
# → RAM-ში არის!

# თუ გამომავალი შეიცავს "/dev/sda":
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        50G   20G   28G  42% /tmp
# → დისკზეა!
```

პრაქტიკული მაგალითები:

```
# დროებითი ფაილის შექმნა
echo "დროებითი მონაცემები" > /tmp/mytemp.txt

# ბევრი პროგრამა იყენებს /tmp/-ს
ls -lh /tmp/
# ნახავთ სხვადასხვა დროებით ფაილებს

# ფაილის შექმნა უნიკალური სახელით
mktemp
# გამომავალი: /tmp/tmp.XxXxXxXx (უნიკალური სახელი)

# დროებითი დირექტორია
mktemp -d
# გამომავალი: /tmp/tmp.XXXXXXXXXX/
```

რა მოხდება გამორთვისას:

- თუ **RAM-ში**: ❌ **სრულად გაქრება**
- თუ **დისკზე**: 🗑️ **შეიძლება წაიშალოს** სისტემის პოლიტიკის მიხედვით (ჩვეულებრივ 10 დღის შემდეგ)

---

5. რატომ ჩანან ვირტუალური დირექტორიები რეალურად

5.1. Linux-ის აბსტრაქციის მექანიზმი

Linux იყენებს **VFS** (Virtual File System) - ვირტუალურ ფაილურ სისტემას.

რას აკეთებს VFS:

1. ქმნის **ერთიან ინტერფეისს** ყველა ტიპის ფაილური სისტემისთვის
2. მომხმარებელს აძლევს **ილუზიას** რომ ყველაფერი ფაილია
3. **მალავს** განსხვავებებს ფიზიკურ დისკსა და ვირტუალურ სისტემებს შორის

მაგალითი:

```
თქვენ: cat /proc/cpuinfo
        ↓
VFS: "უჰმ, /proc/ არის procfs ტიპის..."
        ↓
Kernel: "აჰა, ეს არ არის ფაილი, ეს არის CPU-ს ინფორმაცია!"
        ↓
Kernel აგენერირებს ინფორმაციას რეალურ დროში
        ↓
VFS აბრუნებს თქვენ როგორც "ფაილის შიგთავსს"
        ↓
თქვენ ხედავთ: processor: 0, model name: Intel...
```

5.2. ფაილური სისტემის ინტერფეისი

ყველა ინტერაქცია ფაილებთან ხდება **სტანდარტული ოპერაციებით**:

- `open()` - ფაილის გახსნა
- `read()` - წაკითხვა
- `write()` - ჩაწერა
- `close()` - დახურვა

ეს მუშაობს ყველაფერზე:

```
# რეალური ფაილი
cat /home/student/file.txt
# → open("/home/student/file.txt")
# → read(ფაილიდან მონაცემების წაკითხვა დისკიდან)

# ვირტუალური "ფაილი"
cat /proc/cpuinfo
# → open("/proc/cpuinfo")
# → read(kernel აგენერირებს CPU ინფოს)

# მოწყობილობის "ფაილი"
cat /dev/urandom | head -c 10
# → open("/dev/urandom")
# → read(kernel აგენერირებს შემთხვევით ბაიტებს)
```

თქვენთვის ყველა ერთნაირად გამოიყურება!

5.3. Kernel-ის როლი

Linux Kernel არის "თარჯიმანი":

1. **ფიზიკური ფაილებისთვის:**

```
cat file.txt → Kernel → წაიკითხე დისკიდან სექტორი 12345
```

2. **ვირტუალური ფაილებისთვის:**

```
cat /proc/meminfo → Kernel → გამოითვალე მეხსიერების მდგომარეობა
                           → დააფორმატე ტექსტად
                           → დაუბრუნე როგორც "ფაილის შიგთავსი"
```

3. **მოწყობილობებისთვის:**

```
cat /dev/sda → Kernel → წაიკითხე ბაიტები მყარი დისკიდან
echo "hi" > /dev/pts/0 → Kernel → გამოაჩინე "hi" ტერმინალში
```

ილუზია სრულყოფილია! თქვენ არასოდეს ხედავთ ამ სირთულეს - ყველაფერი "ფაილივითაა".

---

6. როგორ შევამოწმოთ რომელი დირექტორია სად არის

6.1. `df` ბრძანება

`df` (Disk Free) - აჩვენებს ფაილური სისტემების დისკზე მდებარეობას და ზომას.

ძირითადი გამოყენება:

```
# ყველა ფაილური სისტემის ნახვა ადამიანისთვის გასაგები ფორმატით
df -h
```

გამომავლის მაგალითი:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        50G   20G   28G  42% /
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           3.9G  1.2M  3.9G   1% /run
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sda5       100G   50G   46G  53% /home
tmpfs           790M   16K  790M   1% /run/user/1000
```

როგორ ვიკითხოთ:

- **Filesystem** - ფაილური სისტემის ტიპი
  - `/dev/sdaX` = რეალური დისკის პარტიცია (🟡 დისკზე)
  - `tmpfs` = დროებითი ფაილური სისტემა (🟤 RAM-ში)
- **Size** - სულ ზომა
- **Used** - გამოყენებული
- **Avail** - თავისუფალი
- **Use%** - გამოყენების პროცენტი
- **Mounted on** - სად არის მიმაგრებული

კონკრეტული დირექტორიის შემოწმება:

```
# /tmp/ სად არის?
df -h /tmp/

# /proc/ სად არის?
df -h /proc/

# /home/ სად არის?
df -h /home/
```

6.2. `mount` ბრძანება

`mount` - აჩვენებს ყველა მონტირებულ ფაილურ სისტემას.

ძირითადი გამოყენება:

```
# ყველა მონტირებული სისტემა
mount

# მხოლოდ tmpfs სისტემები (ვირტუალური)
mount | grep tmpfs

# მხოლოდ /dev/ სისტემები (ფიზიკური დისკები)
mount | grep "^/dev/"
```

tmpfs-ის მაგალითი:

```
$ mount | grep tmpfs
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=805340k)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=809216k)
```

განმარტება:

- `tmpfs on /run` - `/run/` დირექტორია არის tmpfs ტიპის (🟤 RAM-ში)
- `type tmpfs` - ფაილური სისტემის ტიპი
- `(rw,...)` - mount ოპციები

6.3. `ls -l` ბრძანება

ვირტუალური დირექტორიების ნიშანი:

```
ls -ld /proc/ /sys/ /dev/ /run/ /home/
```

გამომავალი:

```
dr-xr-xr-x 180 root root    0 Jan 18 10:00 /proc/
drwxr-xr-x  13 root root    0 Jan 18 10:00 /sys/
drwxr-xr-x  17 root root 3.9K Jan 18 10:00 /dev/
drwxr-xr-x  25 root root  800 Jan 18 10:00 /run/
drwxr-xr-x   5 root root 4.0K Jan 18 08:00 /home/
```

ყურადღება მიაქციეთ ზომას:

- `/proc/` → ზომა = **0** (🔘 ვირტუალური!)
- `/sys/` → ზომა = **0** (🔘 ვირტუალური!)
- `/dev/` → ზომა = **3.9K** (პატარა, 🔘 ვირტუალური)
- `/run/` → ზომა = **800** (პატარა, 🔘 ვირტუალური)
- `/home/` → ზომა = **4.0K** (🔘 რეალური დირექტორია)

ზომა 0 ან ძალიან პატარა → სავარაუდოდ ვირტუალურია!

6.4. პრაქტიკული ექსპერიმენტები

ექსპერიმენტი 1: /proc/ დინამიკა

```
# 1. ნახეთ /proc/ ზომა
du -sh /proc/
# გამომავალი: 0 (ვირტუალურია!)

# 2. ნახეთ პროცესების რაოდენობა
ls /proc/ | grep -E '^[0-9]+$' | wc -l
# დაიმახსოვრეთ რიცხვი

# 3. გახსენით ბევრი პროგრამა
firefox &
gedit &
gnome-calculator &

# 4. ხელახლა ნახეთ
ls /proc/ | grep -E '^[0-9]+$' | wc -l
# ახლა მეტია!

# 5. /proc/ ზომა კვლავ 0-ია
du -sh /proc/
# კვლავ 0! რადგან RAM-ში არის
```

ექსპერიმენტი 2: /tmp/ შემოწმება

```
# 1. შეამოწმეთ სად არის /tmp/
df -h /tmp/

# 2. შექმენით ფაილი
echo "ტესტი" > /tmp/test_file.txt

# 3. დაადასტურეთ რომ არსებობს
cat /tmp/test_file.txt

# 4. გადატვირთეთ (ან დაელოდეთ გადატვირთვას)
sudo reboot

# 5. შემოწმეთ ხელახლა
cat /tmp/test_file.txt
# თუ /tmp/ RAM-ში იყო: ფაილი აღარ არსებობს!
# თუ დისკზე იყო: ფაილი შეიძლება არსებობდეს
```

ექსპერიმენტი 3: ყველაფრის დადასტურება

```
# შევქმნათ სრული ანგარიში:

echo "=== ფიზიკურ დისკზე ==="
df -h | grep "^/dev/"

echo ""
echo "=== RAM-ში (tmpfs) ==="
df -h | grep "tmpfs"

echo ""
echo "=== ვირტუალური დირექტორიები ==="
mount | grep -E "proc|sysfs|devtmpfs"
```

სასარგებლო რჩევები

1. ნუ შეშფოთდებით /proc/, /sys/, /dev/ "ზომით"

- ეს დირექტორიები არ იკავებენ რეალურ ადგილს დისკზე
- `du -sh /proc/` ყოველთვის გაჩვენებთ 0-ს ან ძალიან პატარა ზომას

2. ფრთხილად იყავით /dev/-ში

- `/dev/sda` არის მთელი თქვენი მყარი დისკი
- არასწორი ბრძანება შეიძლება წაშალოს მთელი სისტემა
- არასოდეს გამოიყენოთ `dd` თუ არ იცით რას აკეთებთ

3. /tmp/ გამოიყენეთ დროებითი ფაილებისთვის

- კარგია დიდი დროებითი ფაილებისთვის (თუ /tmp/ დისკზეა)
- ან პატარა სწრაფი ფაილებისთვის (თუ RAM-ში არის)
- არ ინახოთ მნიშვნელოვანი რამ - შეიძლება წაიშალოს!

4. სისტემის მონიტორინგი

```
# რეალურ დროში ნახვა:
watch -n 1 cat /proc/meminfo      # მეხსიერება
watch -n 1 cat /proc/loadavg      # დატვირთვა
tail -f /var/log/syslog           # ლოგები
```

5. დისკის ადგილის გათავისუფლება

- `/var/log/` ჩასტუმბეთ ძველი ლოგები: `sudo journalctl --vacuum-time=7d`
- `/tmp/` გაწმინდეთ: `rm -rf /tmp/*` (ფრთხილად!)
- `/var/cache/apt/` გაწმინდეთ: `sudo apt clean`

6. სწრაფი შემოწმება

```
# რა იკავებს ბევრ ადგილს?
sudo du -sh /* 2>/dev/null | sort -hr | head -10

# რომელ პარტიციას მთავრდება ადგილი?
df -h | grep -E "8[0-9]%|9[0-9]%|100%"
```

7. ვირტუალური დირექტორიების სწავლა

- ექსპერიმენტირეთ /proc/ და /sys/-ში - ვერაფერს დააზიანებთ!
- ბევრ ფაილს /proc/sys/-ში შეუძლია პარამეტრების შეცვლა
- გამოიყენეთ `cat` ინფორმაციის სანახავად

8. უსაფრთხოების რჩევები

- არასოდეს გაუშვათ: `rm -rf /`
- ფრთხილად იყავით root უფლებებით
- სანამ წაშლით, შეამოწმეთ: `pwd` და `ls`

შეჯამება

მთავარი დასკვნები:

1. **Linux-ში ყველაფერი ფაილია** - ეს აბსტრაქცია საშუალებას გაძლევთ ერთნაირად იმუშაოთ სხვადასხვა რესურსებზე.

2. **არა ყველა დირექტორია არსებობს ფიზიკურად დისკზე:**

   - 🟢 **დისკზე:** /home, /etc, /bin, /boot, /lib, /usr, /var, /opt, /root
   - 🔵 **RAM-ში:** /proc, /sys, /dev, /run
   - 🟡 **დამოკიდებულია:** /tmp

3. **ვირტუალური დირექტორიები არიან "ფანჯრები"** სისტემის მიმდინარე მდგომარეობაში:

   - `/proc/` → პროცესები და სისტემის ინფო

- `/sys/` → აპარატურა და kernel პარამეტრები
- `/dev/` → მოწყობილობების ფაილები
- `/run/` → რან-თაიმ მონაცემები

4. **Linux Kernel ქმნის ილუზიას** - VFS აბსტრაქციით ყველაფერი ფაილივით გამოიყურება, მიუხედავად იმისა რომ შიგნით სრულიად განსხვავებული პროცესები მიმდინარეობს.

5. **შემოწმების გზები:**

   - `df -h` - ნახეთ რა არის დისკზე, რა tmpfs-ში
   - `mount | grep tmpfs` - ვირტუალური სისტემების სია
   - `ls -ld` - ზომა 0 ან პატარა = ვირტუალური

6. **რატომ არის ეს მნიშვნელოვანი:**

   - თქვენ იცით რა მონაცემები შენარჩუნდება გამორთვის შემდეგ
   - გესმით როგორ მუშაობს სისტემა ქვემოთ
   - შეგიძლიათ უკეთ დიაგნოსტიკა და პრობლემების გადაჭრა

დამახსოვრება:

```
ფიზიკური (დისკი) = მუდმივი, შენარჩუნებული, ნელი
   /home, /etc, /var, /usr, /bin, /boot, /lib

ვირტუალური (RAM) = დროებითი, დინამიური, სწრაფი
   /proc, /sys, /dev, /run

Linux-ის ფილოსოფია = "ყველაფერი ფაილია"
   ისე მოეპყარით აპარატურას თითქოს ფაილს
```

შემდეგი ნაბიჯები:

ახლა რაც გესმით ფაილური სისტემის არქიტექტურა:

1. პრაქტიკაში გამოცადეთ ბრძანებები
2. შექმენით ფაილები სხვადასხვა დირექტორიებში
3. დააკვირდით /proc/ და /sys/-ს დინამიკას
4. ისწავლეთ ფაილებთან მუშაობის ბრძანებები (cp, mv, rm, touch)

დამატებითი რესურსები:

```
# სისტემის დოკუმენტაცია
man hier                # ფაილური სისტემის იერარქია
man proc                # /proc/ დოკუმენტაცია
man tmpfs               # tmpfs დოკუმენტაცია

# ბრძანებების დახმარება
man df                  # df ბრძანება
man mount               # mount ბრძანება
man ls                  # ls ბრძანება
```