1. **File System, inodes, folders**

Файлова система — спосіб організації даних, який використовується операційною системою для збереження інформації у вигляді файлів на носіях інформації. Також цим поняттям позначають сукупність файлів та директорій, які розміщуються на логічному або фізичному пристрої. Створення файлової системи відбувається в процесі форматування. Операційна система Unix використовує деревоподібну структуру для зберігання файлів. Корінь цієї структури (для всієї системи) називається /. До вашої основної (домашньої) директорії можна звернутися як ~ або ~yourloginname. Домашньою директорією іншого користувача з логіном guy буде ~guy. До директорії, з якою ви працюєте зараз, можна звернутися за допомогою команди "." (крапка). До батьківської директорії (вище по дереву) можна звернутися за допомогою ".." (дві крапки).**Тип файлу визначається командою *ls* (параметр -l).**

В багатокористувацьких системах з'являється ще одне завдання: захист файлів одного користувача від несанкціонованого доступу іншого користувача, а також забезпечення спільної роботи з файлами.

Файл може розташовуватися на різних носіях. Файли бувають постійними, тобто записаними на диску, або тимчасовими (у пам'яті). Дані з файлу можуть виводитися на термінал, або файл може приймати дані з терміналу. Якщо файл постійний, то його можна переглянути, а якщо файл тимчасовий, то ви можете навіть не знати про його існування.

У ФС кожен файл визначається конкретним індексом – Inode.
Але при цьому один файл (мова про фізичне розміщення) може мати відразу кілька імен (або шляхів). І якщо в структурі ФС файли будуть відрізнятися, то на жорсткому диску може відповідати один файл.
Кореневий розділ у Linux один - "/" (root, "корінь)". Розділи називають підкаталогами, примонтованими до відповідних каталогів.

Дані про кожен файл містить Inode - специфічний для UNIX-систем індексний дескриптор, що зберігає різну **метаінформацію** (власник файлу, останнім часом звернення, розмір тощо). Коли файл (каталог) переміщається до іншої ФС, його Inode також створюється заново. Також зазначимо, що файл (каталог) існує доти, поки зберігається інформація про його ім'я або шляхи до нього. Після видалення інформації блоки, відведені під файл, стають вільними (для виділення під інший файл).

В Unix існує 6 типів файлів:
Звичайний (regular - набір блоків (можливо порожній) на пристрої зовнішньої пам'яті, на якому підтримується файлова система. Такі файли можуть містити як текстову інформацію (зазвичай у форматі ASCII), так і довільну двійкову інформацію);
Каталог (directory- містить імена файлів, які в ньому знаходяться, та посилання на інформацію, яка дозволяє ОС виконувати операції над цими файлами. Право на запис в каталог має тільки ядро. Каталог є таблицею, кожен запис якої відповідає деякому файлу);
/bin(виконувані модулі), lib-бібл компілятора C, dev-пристрої, etc–конфігурація та вик ф, boot-необх ф для зав, home-дані про юз, proc, svr, sys-обладнання, tmp, usr-ф прогр, var-логи
Файл зовнішнього пристрою; - дозволяє доступ до цього пристрою символьні (байт-орієнтовані) та блочні файли. Символьні -небуферизований обмін, а блочні – обмін даних.
Канал з іменем (**FIFO**); - обмін інформацією між двома процесами на одному комп'ютері. Тимчасовий, існує поки виконується, mknod [назва каналу] p mkfifo [назва каналу]
Посилання (link); - вказівники на дійсні файли, що не дублють вмісту файлів., жорстке посилання (Hard-Link), яке є одним із шляхів файлу (команда ls -li). символьне посилання (Symbolic link) - це файл UNIX з текстовим рядком шляхом до оригінального файлу.
Сокети (socket). - програмний інтерфейс, який використовується для обміну інформацією між двома комп'ютерами.

2. SHELL

Командна оболонка (Shell) є інтерпретатором командного рядка в Unix-подібних операційних системах, що забезпечує традиційний Unix-подібний користувацький інтерфейс командного рядка. Загалом **Shell** – це спеціальна програма, яка використовується як посередник між ядром ОС і користувачем. Користувачі керують роботою комп'ютера, вводячи команди як текст, або створюючи текстові скрипти однієї чи більше таких команд. Взаємодія з оболонкою Unix відбувається за допомогою термінала.

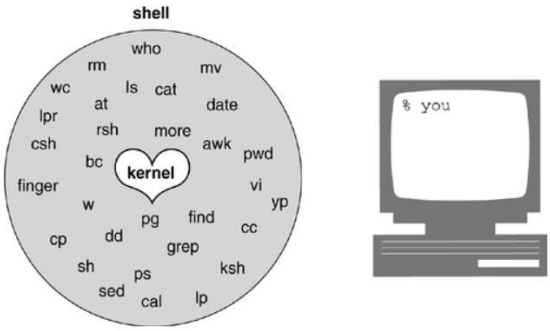
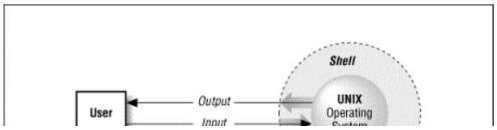


Рис.1.Shell



Ядро – це програма, що реалізовує взаємодію між програмами та обладнанням комп'ютера. Ядро завантажується у пам'ять при запуску і керує системою до завершення роботи.

Всі інші програми, включаючи Shell, знаходяться на диску. Ядро завантажує їх у пам'ять, виконує, і очищає систему, коли вони завершуються. Для запуску тієї чи іншої оболонки в емуляторі терміналу потрібно ввести назву цієї оболонки.

Скрипт, або сценарій — це набір команд, що записані у порядку виконання. Скрипти зберігаються у файлах, що потім викликаються в оболонці. Оболонка = мова програмування. Вони – інтерпритуються, а потім компілюються файлом за раз!

Bourne Shell (sh) є стандартною оболонкою Unix, і використовується переважно для адміністрування системи завдяки своїй чіткості, компактності і швидкості. Ця оболонка була створена Стівеном Борном у AT&T Bell Laboratories. Вперше вона була представлена у Unix V7 у 1979 р. Оболонка базується на мові програмування Algol. Стандартна вказівка у Bourne Shell - це знак $. *MY_MESSAGE="Hello World" # оголошено змінну MY_MESSAGE*

C Shell (csh) була розроблена Білом Джоєм в Берклі і включила ряд новинок, таких як: історія команд, вбудована арифметика, aliasing, job control і т.п. Ця оболонка базується на мові програмування C, в якої вона власне і запозичила синтаксис для написанні скриптів. Стандартна вказівка у C Shell — це знак %. *set MY_MESSAGE = "Hello World" # оголошено змінну MY_MESSAGE*

Korn Shell (ksh) розроблена Девідом Корном. Це інтерактивна командна оболонка і мова програмування, яка забезпечує доступ до системи Unix і багатьох інших систем. Korn Shell належить до родини Bourne-сумісних командних оболонок.

Bourne Again Shell (bash) – це удосконалена Bourne Shell. **TC Shell** – розширення C Shell з додатковими можливостями. Потрібно зазначити, що Bash і TC Shell можна використовувати не лише у Linux, а й у Unix системах.

суперкористувач rood, але для зм шкоди використовуємо sudo

3. Basic UNIX commands (Directory files manipulation)

ls	Переглядає вміст поточного каталогу
touch file_name	Створює файл file_name
mkdir directory_name	Створює директорію directory_name
cat file_name	Показує вміст файлу file_name у терміналі
less file_name	Забезпечує перегляд файлу за допомогою скролінгу
rm file_name	Видаляє файл file_name
rm -r	Видаляє рекурсивно всі файли з директорії
rmdir directory	Видаляє папку directory, яка знаходиться у поточній папці
ln -s /home/user/directory_name/ /home/user/test/	Створює жорсткі та символічні посилання на файли чи папки. Для створення символічного посилання використається опція -s
pwd	Виводить каталог, в якому знаходиться користувач
which program	Виводить каталог, у якому встановлено програму
cd directory_name	Переходить до директорії directory_name
cp file_name directory_name	Копіює file_name до директорії directory_name
nano	Ініціює запуск найпростішого текстового редактора командного рядка Linux
mv file_name directory_name	Переміщує file_name до директорії directory_name
mv old_name new_name	Перейменовує файл/директоріюold_name на new_name
locate file_name	Виконує швидкий пошук файлу
chmod 644 file_name	Змінює права доступу до файлу або каталогу
diff	порівнює два файли і показує, чи вони відрізняються, і, в деяких випадках, у чому полягає різниця між ними.

gzip — команда для стиснення файлу. gunzip — розархівовує файли, стиснені за допомогою gzip. gzcat — команда, яка дозволяє переглядати стиснутий файл без необхідності його розархівування.

```
[-c], [--bytes] – Показує кількість байтів;
[-l], [--lines] – Показує кількість рядків;
[-L], [--max-line-length]
                – Виводить довжину найбільшого рядка;
[-m]            – Виводить кількість символів;
[-w], [--words] – Показує кількість слів у файлі;
[--files0-from=F] – Бере імена файлів з списку у файлі з іменем F
[--help]        – Показує коротку довідку по утиліті;
[--version]     – Виводить версію утиліти;
```

Приклад використання опцій.

1. Підрахунок кількості рядків у файлі з іменем 'foo.txt'.

```
$ wc -l foo.txt
```

Others usrs**: w(з англ. who - хто)** - команда для відображення залогованих користувачів і їх дій. **last - l username** - дає інформацію про користувача, коли і звідки він був востаннє залогований. Якщо не вказати параметру, то команда видасть відповідну інформацію для кожного користувача.

Your acc: **whoami(з англ. who am i - хто я є)** - команда, що повертає Ваш логін. passwd(з англ. **password** - пароль) - команда для зміни паролю. **kill** PID(з англ. kill - вбити) - команда для завершення процесу за його ID. **ps -u yourusername** (з англ. processes - процес) - команда для виведення списку процесів, які зараз виконуються.

4. File permissions

права доступу) — дозвіл або заборона здійснення певних типів дій над об'єктами файлової системи. Права доступу визначають набір дій (читання, запис, виконання), дозволених для виконання користувачам системи над файлами та каталогами. У будь-якого файлу в системі є власник — один з користувачів. Однак кожен файл одночасно належить і деякій групі користувачів системи.. Права доступу можуть бути надані трьом класам користувачів: • власнику файлу • групі, якій належить файл • іншим користувачам

Щоб отримати інформацію про права доступу можна виконати команду **ls** з ключем **-l**. При цьому буде виведена докладна інформація про файли і каталоги, в якій будуть, серед іншого, відображені права доступу. Розглянемо наступний приклад:

```
/home/mary/docs# ls -l report1303
-rw-r--r-- 1 mary users 505 Mar 13 19:05 report1303
```

Перше поле в цьому рядку **(-rw-r--r--)** відображає права доступу до файлу. Третє поле вказує на власника файлу **(mary)**, четверте поле вказує на групу, яка володіє цим файлом **(users)**. Останнє поле - це ім'я файлу **(report1303)**. Інші поля описані в документації до команди **ls**.

Даний файл є власністю користувача **mary** та групи **users**. Послідовність **-rw-r--r--** показує права доступу для користувача-власника файлу, користувачів-членів групи-власника, а також для всіх інших користувачів.

Перший символ з цього рядка **(-)** позначає тип файлу. Символ **(-)** означає, що це - звичайний файл, який не є каталогом (в цьому випадку першим символом було б **d**) або псевдофайлом пристрою (було б **c** або **b**).

Наступні три символи **(rw-)** позначають права доступу, надані **власнику mary**. Символ **r** - скорочення від **read** (читати), а **w** - скорочення від **write** (писати). Отже, **mary** має право на **читання і запис (зміну)** файлу report1303. Після символу **w** могло б стояти декілька можливих символів:

- **x** (**execute** - виконувати), що означає наявність прав на виконання файлу.
- **s** (**setuid** - установка ID користувача), що означає запуск файлу з заміненим UID. Це означає, що при виконанні програма матиме права її власника, а не користувача, який її запустив.
- **t** (**sticky bit** - "липкий біт"), що забороняє перейменовувати або видаляти файл усім користувачам, крім власника файлу чи суперкористувача

оскільки є його власником. Для зміни прав доступу до файлу або каталогу використовується команда **chmod**.

Наступні три символи **(r--)** відображають права доступу **групи** до файлу. Групою-власником файлу в нашому прикладі є група **users**. Оскільки тут присутній тільки символ **r**, всі користувачі з групи **users** можуть читати цей файл, але не можуть змінювати або виконувати його.

Останні три символи **(r--)** показують права доступу до цього файлу всіх **інших користувачів**, крім власника файлу і користувачів із групи users. Так як тут вказано тільки символ **r**, ці користувачі теж можуть тільки читати файл.

Приклади:

1. **-rwxr-x--x**

Користувач-власник файлу може читати файл, змінювати і виконувати його; користувачі-члени групи-власника можуть читати і виконувати файл, але не змінювати його; всі інші користувачі можуть лише запускати файл на виконання.

2. **-rw-------**

Тільки власник файлу може читати і змінювати його.

3. **-rwxrwxrwx**

Всі користувачі можуть читати файл, змінювати його та запускати на виконання.

Права змінюються відразу для трьох типів користувачів: власника-користувача файлу, користувачів групи-власника та для інших користувачів. Аргумент команди **chmod**, що задає дозвіл, може бути записаний у трьох форматах: в **числовому**, **символьному** та **символічному**.

Прикладом **числового** запису може служити **777**, який еквівалентний `-rwxrwxrwx`: кожне право має числовий код і може бути задане вручну:

400 - власник має право на читання;
200 - власник має право на запис;
100 - власник має право на виконання;

040 - група має право на читання;
020 - група має право на запис;
010 - група має право на виконання;

004 - інші мають право на читання;
002 - інші мають право на запис;
001 - інші мають право на виконання.

$4_{10} = 100_2$. Тому 4 відповідає правам доступу r--. Легко бачити, що 0 відповідає правам доступу ---. Тоді 400 буде відповідати правам доступу -r--------.

Додавши ці коди, можна отримати числовий запис. Наприклад, `chmod 444 index.html`: 400 + 40 + 4 = 444 - всі мають право тільки на читання (-r--r--r--).

Ще одним доступним форматом є **символічний**, який дозволяє додавати і видаляти деякі дозволи, при цьому не змінюючи інші. Він має вигляд `chmod [references] [operator] [modes] file`, де:

- `[references]` - користувачі, право яких потрібно встановити:
  - **u** - власник файлу;
  - **g** - група власників файлу;
  - **o** - усі користувачі, крім **u** та **g**;
  - **a** - усі користувачі.
- `[operator]` - визначає, що потрібно зробити з вказаними в **modes** дозволами (користувачам з множини **references**)
  - "+" - додати;
  - "-" - забрати;
  - "=" - задати.
- `[modes]` - дозволи, які потрібно додати, забрати або присвоїти. Їх нотація аналогічна символьному запису.

Розглянемо кілька прикладів використання цього запису:

- `chmod go=w prog.pl` встановлює право на запис для всіх користувачів крім власника.
- `chmod +x prog.pl` надає право на виконання для всіх користувачів.
- `chmod u+r,g+x prog.pl` додає для користувача право на читання, а для групи право на виконання файлу.

Значеннями за замовчуванням є:

- для файлів: 644 (-rw-r-r-) ;
- для директорій: 755 (drwxr-xr-x).

Команда chown

chown (від англ. **ch**ange **own**er) — Unix-утиліта, що змінює власника і/або групу для вказаних файлів. Як ім'я власника/групи береться перший аргумент, що не є опцією. Якщо задано тільки ім'я користувача (або числовий ідентифікатор користувача), то даний користувач стає власником кожного з вказаних файлів, а група цих файлів не змінюється. Якщо за ім'ям користувача через двокрапку слідує ім'я групи (або числовий ідентифікатор групи), без пропусків між ними, то змінюється також і група файлу.

Загальний вигляд команди:

```
chown [-cfhvR] [--dereference] [--reference=rfile] користувач[:група] файл
```

Власник може бути змінений тільки власником файлу або користувачем з правами адміністратора системи.

Приклад:

```
chown vivek demo.txt
```

Команда chgrp

chgrp (з англ. **ch**ange **gr**ou**p**) — UNIX-утиліта. Може використовуватися непривілейованими (не власником файлу) користувачами для зміни групи файлів. На відміну від команди **chown**, **chgrp** дозволяє рядовим користувачам змінювати групи, але тільки ті, членами яких вони є.

chgrp змінює групу кожного заданого файлу на групу, яка може бути представлена як ім'ям групи, так і її числовим ідентифікатором (GID).

Загальний вигляд команди:

```
chgrp [опції] група файл
```

Команда umask

umask (від англ. "***u**ser file creation mode **mask**"* — маска режиму створення користувацьких файлів) — функція середовища POSIX, що змінює права доступу, які присвоюються новим файлам і директоріям за замовчуванням. Права доступу файлів, створених при конкретному значенні **umask**, обчислюються за допомогою наступних побітових операцій (**umask** зазвичай встановлюється в вісімковій системі числення):

- побітове **І** між побітовим запереченням аргументу і режимом повного доступу.

Режим повного доступу для файлів — **666**, для директорій — **777**, бо за замовчуванням створюються невиконувані файли (останній біт в rwx), а для переходу в директорію потрібно, щоб x був встановлений.

Фактично, **umask** вказує, які біти слід скинути у правах, що виставляються на файл — кожний встановлений біт **umask** забороняє виставлення відповідного біта прав. Винятком з цієї заборони є біт, що виконується, який для каталогів виставляється особливо. **umask 0** означає, що слід (можна) виставити всі біти прав (rwxrwxrwx), **umask 777** забороняє виставлення будь-яких прав.

5. File links

Посилання на файл – це особливий файл з інформацією про файл. Існують «жорсткі» вказують на файл за назвою. і «м'які» посилання на файли – за створеною вказівкою ln -s для створення символічного посилання існує окрема команда symlink.

6. C Shell or bash'

C Shell (csh) була розроблена Білом Джоєм в Берклі і включила ряд новинок, таких як: історія команд, вбудована арифметика, aliasing, job control і т.п. Ця оболонка базується на мові програмування С, в якої вона власне і запозичила синтаксис для написанні скриптів. Стандартна вказівка у C Shell — це знак %. *set MY_MESSAGE = "Hello World" # оголошено змінну MY_MESSAGE*

Перевіряємо встановлення :$csh Якщо C Shell не встановлено: $ sudo apt install csh

Розбір командного рядка Інтерпретатор, отримавши командний рядок, виконує над ним ряд перетворень: 1. Розкриває псевдоніми (alias). 2. Розкриває метасимволи (*, [,] , ? , { , } ,~). 3. Підставляє змінні shell. 4. Виконує команду, якщо це вбудована команда інтерпретатора, інакше якщо команда зовнішня, то запускає процес.

Є **змінні** оточення та прості зм.

одинарні лапки (текст всередині одинарних лапок не підлягає розкриттю та інтерпретації) **подвійні лапки** (текст всередині подвійних лапок інтерпретатор вважатиме одним словом)

Список **важливих вбудованих змінних**: home, argv, path, mail, history

Список важливих **вбудованих команд** C Shell ● alias (визначає псевдонім). ● chdir path (команда переходу в каталог path). ● echo (виводить всі аргументи). ● exec filename (запускає процес із файлу замість shell). ● exit (закінчує роботу shell). ● set, setenv (встановити значення змінної чи змінної оточення). ● unset, unsetenv, unalias . ● source filename (зчитує і виконує команди з файлу). ● time command (час,затрачений на виконання команди). ● shift var (зсуває елементи масива var вліво). ● @ name=expr (заносить результат арифм.виразу expr в name).

Вирази між операндами і операторами мають бути *пропуски*; виконуються *справа наліво та зліва направо - залежно від системи*. Тому краще використовувати tcsh - там точно зліва направо.

Будь-який C Shell **скрипт** повинен починатись з рядка: #!/bin/csh, В скриптах змінні $0, $1, $2 і т.д. відомі як **позиційні параметри**. Змінна $0 містить ім'я команди, а змінні $1, $2 і решта є параметрами, що передаються в скрипт. Коли викликається csh скрипт, змінна argv отримує список аргументів, що записані в командній лінійці.

В C Shell значення шляху отримане зі змінної може бути зміненим перед використанням в команді або виразі. **Модифікатори** змінних записуються після ":" вкінці змінної. :h повертає директорію шляху (назва від "head") :t повертає ім'я файлу шляху (назва від "tail") :r повертає директорію і ім'я файлу без останнього розширення (назва від "root") :e повертає розширення шляху (назва від "end")

Змінні в оболонці C Shell є списком з нуля або більше слів. Деякі зі змінних є встановлені оболонкою, а деякі передані їй. Значення змінних можуть бути відображені і змінені використовуючи команди set і unset. Використання змінних. $імя, ${імя} Імена змінних оболонки можуть складатись із букв та цифр і починаються з букви. Найбільша довжина імені - 20 символів. Підкреслення вважається буквою.

Надрукований користувачем рядок інтерпретатор сприймає як команду (або декілька команд). Синтаксис командного інтерпретатора дозволяє набирати **декілька команд в одному рядку**, розділяючи їх крапкою з комою (;).

Керування потоками введення/виведення відбувається за допомогою символів >, <, >>, <<. Додатково C Shell дозволяє групувати команди за допомогою круглих дужок. Наприклад % (command1 | command2) < myfile

7. Pipe

В Юнікс-подібних операційних системах, канали(pipes) - набір процесів, зв'язані своїми стандартними потоками вводу-виводу таким чином, що вихідний потік кожного процесу (stdout) безпосередньо зв'язується зі стандартним потоком вводу (stdin) наступного.

8. Redirection

Redirection - це перенаправлення стандартного потоку даних. У Unix-подібних операційних системах кожна команда, отже і кожний процес, автоматично ініціалізується трьома текстовими потоками даних: одним вхідним (standard input) та двома вихідними потоками (standard output, standard error): stdin (<) – вхідний потік даних, stdout (>) – вихідний потік за звичайних умов, stderr(">&" - записує в файл ">>&" – дописує до файлу) – вихідний потік для інформації про помилки. За замовчуванням команди Unix виводять у вікно терміналу, а вхідні дані вводяться з клавіатури.

за допомогою redirection, вхідний потік можна отримати з довільного файлу або програми. У свою чергу, вихідні потоки також можна перенаправити до довільного файлу, програми чи, навіть, принтера.

Кожному відкритому файлу у системі відповідає файловий дескриптор, який дає можливість однозначно ідентифікувати цей файл. Дескрипторами stdin, stdout та stderr є 0, 1 і 2 відповідно. Для інших файлів в системі залишаються дескриптори від 3 до 9.

Комбіноване перенаправлення

Ми одночасно можемо перевизначати декілька потоків. Основна форма команди із перенаправленням стандартного введення та виведення виглядає наступним чином:

```
command < in-file > out-file
```

або

```
command > out-file < in-file
```

Запишемо посортований вміст файлу *letters* в файл *sorted-letters*:

```
% sort < letters > sorted-letters
```

Найпоширеніші команди переадресації

1) команда > файл - Направляє стандартний потік виводу в новий файл

2) команда >> файл - Направляє стандартний потік виводу у вказаний файл (режим приєднання)

3) команда > файл1 >&файл2 - Направляє стандартні потоки виводу і помилок у вказаний файл

4) команда &> файл - Направляє стандартний потік помилок у вказаний файл

5) команда &>> файл - Направляє стандартний потік помилок у вказаний файл (режим приєднання)

6) команда >> файл1 >>&файл2 - Направляє стандартні потоки виводу і помилок у вказаний файл (режим приєднання)

7) команда < файл1 > файл2 - Отримує вхідні дані з першого файлу і направляє вихідні дані у другий файл

8) команда < файл - в якості стандартного вхідного потоку отримує дані з вказаного файлу

9) команда << розділювач - Отримує дані зі стандартного потоку вводу до тих пір, поки не зустрінеться розділювач

10) команда <&m - В якості стандартного вхідного потоку отримує дані з файлу з дескриптором m

11) команда >&m - Направляє стандартний потік виводу в файл з дескриптором m пОператор n>&m дозволяє перенаправляти файл з дескриптором n туди, куди спрямований файл з дескриптором m.

9. Alias

Alias (дослівно "псевдонім", "синонім") — це вбудована команда в різних командних рядках, таких як Unix shells, Windows PowerShell та інших, яка дозволяє здійснити заміну якогось слова (команди) на інший рядок. Псевдоніми працюють тільки в тому середовищі, в якому були створені, і лише під аккаунтом того користувача, який їх створив. Зазвичай зберігаються протягом одної сесії. **% alias name "value"** можна вик і одинарні лапки, аласи часто вик для управління пакетами(встановлення відкриття і тп) *alias — відображає список поточних псевдонімів; alias myAlias - виводить комаду псевдоніму myAlias. %unalias n – del n* для того щоб він був постійним – записуємо у конфіг файл.

10.Control structures in bash or tsch

Структури керування – це вираз або група виразів, що визначають послідовність виконання деяких рядків коду. Ось перелік структур керування у C-Shell: if/then/else/endif; foreach/end;(команда виккористовується почергово до кожного елемента) break; continue; goto(безумовний перехід до мітки); switch/case/endsw(умовне виконання від значення); while/end(вик доти поки умова - тру); onintr;(передає керування, коли перериваємо скрипт оболонки (CTRL-C). Контроль переноситься на стрічку, яка починається з мітки.) repeat(повторення команди н-ну ксть разів, вказану після оператора).

```
if (-e myfile) echo myfile already exists
```

У наступній таблиці наведені інші можливі стани файлу:

d	файл є директорією
e	файл існує
f	файл є звичайним файлом
o	користувач є власником файлу
r	користувач має права доступу на читання файлу
w	користувач має права доступу на запис в файл
x	користувач може виконувати файл
z	файл довжиною 0 байт

11.Variables in bash or tsch

У мові C Shell визначені наступні типи змінних: слово, рядок, масив слів, позиційна змінна.

```
% set name_1                        # визначення змінної без ініціалізації
% set nameword = word               # змінна ініціалізується словом
% set namelist = (value1 value2)    # змінна задається списком слів
% set namelist[1] = value3          # задається значення елементу списку,
                                    # причому індексація списку починається з 1
% set a = 5 b = 7                   # ініціалізація кількох змінних
```

Імена змінних можуть складатися з букв, цифр і знаку підкреслення, та не починаються з цифр.

Визначити змінну і присвоїти їй значення можна за допомогою команд set, setenv або присвоївши змінній значення виразу. У C Shell змінні є динамічними. Вони оголошуються перший раз коли викликається команда set. Відповідно вони видаляються викликом команди unset: Різниця між простою змінною і *змінною оточення* полягає лише у способі їх визначення. Змінні оточення видимі для дочірніх процесів. Щоб використати значення змінної потрібно перед нею поставити **символ $**. Використання значення змінної називають підстановкою. Крім змінних, визначених користувачем, команда set виведе також наперед визначені змінні.
Змінні ми можемо виводити, видаляти, перевірити чи існують та визначати кількість слів.
Щоб перевірити, чи змінна вже оголошена, використовується символ **'?',** «\» щоб вивести, видалити – unset, визначити кількість слів змінної : «$#»
Підстановка змінних завжди відбувається у подвійних лапках? І не відбув в одинарних.
Рядок між гравісами (`) інтерпретується як командний рядок. Цей командний рядок виконується і замінюється результатом його виконання. (наприклад exo і змінна)
Щоб виокремити змінну використовуємо фігурні дужки.

Арифметичні змінні

Змінні, значення яких є цілими числами використовують дещо іншу команду set. Символ @ використовується замість команди set для позначення присвоєння цілих чисел. Також цей символ означає що всі зазначені далі слова утворюють вираз.

```
% set a = 5 b = 7
% @ c = ($a + $b)
% echo $c
12
% set exp = (1 1)
% @ exp[2] = 5 * -4
```

Модифікатори вик для зміни значень змінних, призначені для маніпуляцій з іменами файлів та каталогів

Використання модифікатора має наступний вигляд:

$var:m або ${var:m}

де m – один з модифікаторів:

- e – (extension) видаляє все, крім розширення;
- h – (head) видаляє задню частину шляху, залишаючи голову;
- t – (tail) видаляє всі передні частини шляху, залишаючи хвіст;
- r – (root) видаляє розширення;
- q – (quote) змінює кожне слово згідно модифікатора що слідує після нього, наприклад gh;
- x – розбиває рядок на список слів у місцях пропуску, символах табуляції та нового рядка.

Нехай маємо скриптовий файл script.csh, після виконання якого отримаємо:

```
% csh script.csh
$path = /home/user/pictures/image.png
$path:e = png
$path:h = /home/user/pictures
$path:t = image.png
$path:r = /home/user/pictures/image
$path = /home/user/pictures/image.png /home/user/documents/f
```

Масиви в C Shell динамічні за розміром, тобто вони можуть зменшуватись/збільшуватись під час виконання. Для доступу до елементів масиву використовується квадратні дужки, нумерація починається з 1. Команда shift використовується для того щоб позбутись першого елемента масиву, і зсунути елементи що залишились на 1 індекс назад. Якщо не передано ніякого аргумента до команди shift, тоді вона зсуває вбудовану змінну argv.

C Shell працює дуже схоже на мову C по відношенню до **булевих виразів**. 0 трактується як false, будь що інше трактується як true.

Далі подано спсок булівських операторів порівняння:

- == – рівність (використовується для стрічок і чисел)
- != – нерівності (використовується для стрічок і чисел)
- =~ – стрічкова рівність
- !~ – стрічкова нерівність
- <= – числове менше або рівне
- >= – числове більше або рівне
- > – числове більше
- < – числове меньше

Зауважте, що деякі з них використовуються тільки для стрічок, деякі тільки для чисел. Якщо змінна не містить значення, то в числовому контексті вона буде інтерпретуватись як 0.

Виведення дуже просте. Для виведення літералів і змінних можна використовувати команду echo. Якщо не подтрібно виводити кінець стрічки, можна використати -n. Для того щоб отримати ввід з консолі, використовується $<. Ця команда зупиняє скрипт доки користувач не введе кінець стрічки (клавіша Enter).

Позиційні змінні ініціалізуються в командному рядку при запуску командного файла на виконання. Наприклад: % csh comfile aaa bbb ccc ddd Кожне слово цього рядка, починаючи з comfile, доступно всередині командного файлу comfile. Щоб отримати значення слова, достатньо вказати його номер у рядку, причому $0 – назва поточного файлу. Також можливо використовувати аргументи через змінну argv як список слів. Для отримання всіх аргументів використовують **$argv, $argv[*], $***

12.Built-in shell variables

У оболонці C-Shell, існують так звані "вбудовані змінні". Ці змінні переважно призначені для встановлення опцій оболонки.

Argv – масив аргументів, переданих разом з назвою файлу/команди що викликається

cdpath (change directory path) Змінна-список, що містить повні шляхи альтернативних директорій для пошуку аргументів для команди cd

child (child process) Змінна містить ідентифікатор останнього запущеного фонового процесу. Коли процес завершений, змінна невизначена.

cwd (current working directory) Змінна містить повний шлях до каталогу в якому ми зараз перебуваємо. Pwd – only dir

echo mode (set / unset) Коли змінна встановлена (set) - кожна команда відображається в консолі перед виконанням.

histchars (history substitution characters) Змінна зберігає два символи, котрі використовуються для роботи з історією.
history (history list size) Змінна зберігає кількість рядків, котрі будуть зберігатися в історії. Ця змінна потрібна для використання фукції history.
home (home directory) Змінна містить повний шлях до домашньої (початкової) директорії користувача. Ініціалізується C shell, використовуючи значення глобальної змінної середовища HOME.
ignoreeoff (ignore end-of-file character (set / unset)) Коли змінна визначена, оболонка не буде закриватись при введені символу кінця файлу (ctrl+d). Для її закриття потрібно буде ввести команду exit.
mail (mail file) Змінна містить список файлів котрі будуть перевірятися на наявність нових листів через певний проміжок часу.
noclobber (do not clobber files (set / unset)) Коли змінна визначена, оболонка не дозволить перезаписувати вже створені файли для команд перенаправлення потоків (буде повертатися повідомлення про помилку).
noglob (do not allow file expansion (set / unset)) Коли змінна визначена, вона дозволяє оболонці ігнорувати розширення файлів
nonomatch (no error on nonmatching file expansion characters (set / unset)) Коли змінна визначена, команда яка містить розширення, що не відповідає жодному типу файлів не викличе помилки. Також буде відсутнім повідомлення про відсутність файлу та буде використано стандартний вивід.
path (command path list) Змінна містить список, що вказує оболонці де саме потрібно шукати команди, введені користувачем. **(за замовчуванням /bin/usr/bin)**
prompt (C shell promt) Змінна містить символ, що показується на початку кожного нового рядка оболонки. **(за замовчуванням %)**
savehist (save commands in history list) Змінна містить кількість рядків історії, що має бути записано в файл .history
shell (default shell file) Змінна містить повний шлях до оболонки, яка використовується в даний момент. **(за замовчуванням /bin/csh)**
status (last command status) Змінна містить статус останньої виконаної команди: 0 - команда була виконана без помилок, 1 - при виконанні команди відбулась помилка.
time (automatic timing control) Змінна задає максимальний час (у секундах), що може поглинути команда, не повертаючи статистики використання
user (user`s name) Змінна містить ім'я залогованого користувача.
verbose (verbose mode (set / unset)) Коли змінна визначена, то оболонка виводить команду на термінал в такому вигляді, в якому вона перебуває після виконання команд для роботи з іст.
printenv Дозволяє вивести всі змінні середовища з їх значеннями на екран.
lang Змінна містить поточну мову.
ssh_connection Визначає клієнтські і серверні з'єднання. Змінна містить чотири значення, розділені пробілами: IP-адреса клієнта, номер порту клієнта, IP-адреса сервера і номер порту сервера.**ssh_tty** Вказує на ім'я tty (шлях до пристрою)

13. Qt Signal-Slot ideas

Сигнали і слоти (**Signals & Slots**) - це підхід в програмуванні, що дозволяє реалізувати патерн **Спостерігач** (Observer), мінімізовуючи написання повторюваного коду. Концепція сигналів та слотів полягає в тому, що компонент може надсилати сигнали про те, що сталася якась подія (змінився його стан). Інші компоненти можуть приймати і обробляти таку інформацію за допомогою спеціальних функцій - слотів. Сигнали і слоти є центральною рисою Qt і використовуються в ній повсюди, зокрема при програмуванні графічного інтерфейсу користувача (GUI).
Спостерігач (**Observer**) - це патерн проектування, в якому при зміні стану об'єкта всі спостерігачі, що за ним слідкують, отримують і обробляють інформацію про цю зміну.
Зв'язок між об'єктами встановлюється наступним чином: **у одного об'єкта повинен бути сигнал, а у другого - слот.** Сигнал оголошується одного разу і на цьому все, йому не потрібна реалізація. Слот є функцію, і тому крім оголошення повинен мати реалізацію, як і звичайна

функція. Тому, з'єднавши сигнал першого об'єкта і слот другого, ми отримуємо наступне: кожен раз, коли перший об'єкт посилає свій сигнал, другий об'єкт приймає його в свій слот і виконує його функцію.
Для створення зв’язку між сигналом та слотом використовується статичний метод **connect**() з класу QObject.
Формат запису: QObject::connect(QObject *sender,SIGNAL(сигнал(параметри)),QObject *receiver,SLOT(слот(параметри)), type);

Типи з'єднань в Qt:

- Qt::AutoConnection - якщо отримувач і приймач знаходяться в одному потоці, то тип зв'язку Qt::DirectConnection. В протилежному випадку Qt::QueuedConnection. Тип з'єднання визначиться при надсиланні сигналу;
- Qt::DirectConnection - слот виконається зразу після надсилання сигналу. Слот виконається в потоці де надсилається сигнал;
- Qt::QueuedConnection - слот виконається після того, як сигнал попаде в обробник подій потоку(в якому знаходиться отримувач) і "пройде чергу". Слот виконається в потоці отримувача;
- Qt::BlockingQueuedConnection - подібний до Qt::QueuedConnection у випадку різних потоків. Тобто потік, що активує сигнал буде зупинений, допоки сигнал не буде оброблений(в порядку черги) в потоці отримувача;
- Qt::UniqueConnection - прапорець, що може комбінуватись з будь-яким типом з'єднань. Він гарантує, що при повторному створені того самого з'єднання, виконуватись воно буде лиш один раз;

Метаоб'єктний код - це автоматично згенерований код, який дозволяє реалізувати властивості, початково не притаманні C++. Наприклад, в нашому випадку метаоб'єктний код містить назви сигналу і слоту, а також вказівники на них.

14.Qt Events

В **Qt події — це об’єкти**, що наслідуються від абстрактного класу **QEvent**. У свою чергу цей клас представляє дещо, що відбулося всередині застосунку або в результаті зовнішньої активності, про що потрібно знати самому застосунку. Події можуть бути отримані і оброблені будь-яким екземпляром підкласу QObject, але головним чином вони стосуються віджетів.
Коли відбувається подія, то для її представлення **Qt створює об’єкт події, створюючи екземпляр відповідного підкласу QEvent.** Далі Qt доставляє його окремому екземпляру класу QObject (чи одному з його підкласів), викликаючи віртуальну функцію **QObject::event(QEvent* event).** Ця функція не обробляє подію самостійно. Натомість, базуючись на типі доставленої події, вона викликає обробник подій для даного конкретного типу подій і відправляє відповідь на основі того, чи буде подія прийнята чи проігнорована.
Більшість типів подій мають **спеціальні класи**, а саме QResizeEvent для зміни розміру віджета, QPaintEvent для перемалювання, QMouseEvent, QKeyEvent і QCloseEvent. Кожний клас створений як підклас QEvent і додає функції, залежні від подій. Наприклад, QResizeEvent додає size() і oldSize(), щоб дозволити віджетам взнати як змінилися їх розміри. Деякі класи підтримують більше, ніж один реальний тип подій. QMouseEvent підтримує кліки кнопкою мишки, подвійні кліки, переміщення та інші пов’язані операції. *Кожна подія має зв’язаний з нею тип, визначений в QEvent::Type.*
Події обробки клавіш обробляються за допомогою перевизначення функцій keyPressEvent() та keyReleaseEvent().
Іноді об’єкту необхідно ознайомитися і, можливо, перехопити події, які доставлені до іншого об’єкту. Функція QObject::instailEventFilter() дає таку можливість шляхом установки **фільтру події,** заставляючи запропонований об’єкт фільтру отримувати події для об’єкту-приймача в свою функцію **QObject::eventFilter(QObject *obj, QEvent *event).** Фільтр подій отримує можливість обробляти події до того як це зробить об’єкт-приймач, дозволяючи вивчати , і у

випадку необхідності, відкинути подію. Існуючий фільтр подій можна видалити, використовуючи функцію QObject::removeEventFilter(QObject *obj).
Відправлення подій: **sendEvent**() обробляє подію негайно. Коли вона повертається фільтри подій і чи сам об'єкт вже обробили подію, **isAccepted**(), повідомляє про прийняття чи відхилення події останнім обробником, який викликається. **postEvent**() відправляє подію в чергу для наступної відправки.
Events і signal/slots - два механізми, які виконують одну і ту ж річ. Проте існують відмінності у їхньому використанні: **Event** генерується ззовні об'єкта і передається йому через цикл подій у QApplication; сигнали корисні, якщо використовувати віджети, а події - якщо класи наслідувані від віджетів; обробка сигналів виконується відразу після їхнього посилання (синхронно), а подій - через цикл подій (асинхронно) ви обробляєте події, але сповіщаєтесь через сигнали сигнали/слоти використовуються для комунікації між класами сигнал може мати багато слотів, але подія - лише одного отримувача
Signal виробляється коли відбувається певна подія. **Slot** це функція, яка викликається у відповідь на певний сигнал.

Власні події можуть бути оброблені об'єктами. Для створення власних подій потрібно:

1. Створити обробник власних подій для цього класу (customEvent(...)), який буде викликатись для обробки події після того, як подія буде надіслана до екземпляру цього класу.

2. Надіслати подію, використовуючи QApplication::postEvent(...) і вказати екземпляр вище вказаного класу як один з параметрів.

Зразок реалізації **customEvent():**

```
void MyLineEdit::customEvent(QCustomEvent *event)
    {
        if (event->type() == MyEvent) {
            myEvent();
        } else {
            QLineEdit::customEvent(event);
        }
    }
```

Зразок надсилання власної події:

```
const QEvent::Type MyEvent = (QEvent::Type)1234;
    ...
    QApplication::postEvent(obj, new QCustomEvent(MyEvent));
```

Події можуть бути поширені з використанням функцій accept() та ingnore(), які "повідомляють" Qt, що ви "приймаєте" чи "відкидаєте" подію. Якщо обробник події викликає **accept**(), подія не буде поширена далі; якщо ж **ignore**(), то вона буде шукати наступного обробника.

15.Qt Widgets

Віджети є основними елементами для створення інтерфейсу користувача в Qt. Віджети можуть відображати дані та інформацію про стан, отримувати введені користувачем дані, а також забезпечувати контейнер для інших віджетів, які повинні бути разом. Віджет, який не є вбудованим в батьківський віджет, називається вікном. Для того, щоб мати можливість використовувати віджети у вашому проекті, необхідно підключити до проекту бібліотеку QtWidgets: **#include <QtWidgets>** Щоб зв'язати ваш проект з цим модулем, необхідно додати у ваш qmake.pro файл наступний рядок: QT += widgets.
QWidget використовують для створення вікна, всередині якого будуть знаходитися інші віджети. Всі елементи, призначені для створення інтерфейсу користувача є підкласами QWidget, або пов'язані з підкласами QWidget. В свою чергу QWidget є **підкласом QObject.**
Для створення власних віджетів потрібно *створити клас який буде наслідуватися від QWidget* або відповідного його підкласу та перевизначити віртуальні обробники подій.
Бібліотека QtWidgets містить велику кількість вбудованих стандартних загальновідомих віджетів, таких як кнопки, поля вводу даних, датчики прогресу, випадаючі меню та інші.

В наведеній нижче таблиці вказані основні базові віджети і їх короткий опис:

QCheckBox	Чекбокс з текстовою міткою
QComboBox	Поєднання кнопки і випадаючого меню
QCommandLinkButton	Кнопка-лінк
QDateEdit	Віджет для редагування дати на базі QDateTimeEdit віджета
QDateTimeEdit	Віджет для редагування дати і часу
QDial	Округлий регулятор (на зразок спідометра)
QDoubleSpinBox	Лічильник для введення дійсних чисел
QFocusFrame	Фрейм, який може бути поза межам стандартної області промальовування віджету
QFontComboBox	Випадаючий список шрифтів
QLCDNumber	Відображає число LCD-подібними цифрами
QLabel	Текст або малюнок
QLineEdit	Онлайн текстовий редактор
QMenu	Віджет меню, який використовується в панелі меню, контекстному меню та інших спливаючих меню
QProgressBar	Горизонтальна і вертикальна полоса статусу
QPushButton	Командна кнопка
QRadioButton	Вибираюча кнопка з текстовою міткою
QScrollArea	Скролінг на іншому віджеті
QScrollBar	Вертикальний або горизонтальний скролінг
QSizeGrip	Зміна розміру вікна
QSlider	Вертикальний і горизонтальний регулятор
QSpinBox	Лічильник
QTabBar	Панель вкладок
QTabWidget	Стек віджетів з вкладками
QTimeEdit	Віджет для редагування часу на основі QDateTimeEdit віджета
QToolBox	Смужка-меню з віджетами

Додаткові: календар, список,табл, дерово

В наведеній нижче таблиці вказані основні додаткові віджети і їх короткий опис:

QCalendarWidget	Календар з можливістю вибору дати
QColumnView	Model/view реалізація для відображення колонок
QDataWidgetMapper	Зіставлення частини моделі даних з віджетом
QDesktopWidget	Забезпечує доступ до інформації про екран
QListView	Надає список з іконками для відображення моделі
QTableView	Model/view реалізація таблиці за замовчуванням
QTreeView	Model/view реалізація дерева за замовчуванням
QUndoView	Відображення вмісту QUndoStack
Phonon::VideoWidget	Віджет, який використовується для відображення відео
QWebView	Віджет, який використовується для перегляду та редагування веб-документів

Абстрактні віджети

Це абстрактні класи віджетів, які самі по собі не придатні для використання, але забезпечують функціональні можливості, які можна використовувати при наслідуванні цих класів.

QAbstractButton	Абстрактний базовий клас для віджет-кнопок, що забезпечує функціональність кнопок
QAbstractSlider	Елемент для вибору цілого числа в діапазоні
QAbstractSpinBox	Лічильник з рядком редагування для відображення значень
QDialog	Базовий клас для діалогових вікон
QFrame	Базовий клас для віджетів, які можуть містити інші віджети

QWidget має багато **функцій**, але деякі з них мають обмежену функціональність. Наприклад, QWidget має властивість керувати шрифтом, але ніколи його не використовують безпосередньо. Є багато підкласів, які забезпечують реальну функціональність, такі як QPushButton, QListWidget, QTabWidget.

Потужний механізм для кастомізації завнішнього вигляду віджетів надає **таблиця стилів вбудованого класу QStyle.** В ній міститься велика кількість заготованих стилів, які можна редагувати під власні потреби. Також можна створити клас, який буде наслідуватися від QStyle і міститиме налаштування вашого власного стилю

Контекст	
Вікно	show(), hide(), raise(), lower(), close().
Вікно верхнього рівня	isWindowModified(), setWindowModified(), windowTitle(), setWindowTitle(), windowIcc
Вмістиме вікна	update(), repaint(), scroll().
Геометрія	pos(), size(), rect(), x(), y(), width(), height(), sizePolicy(), setSizePolicy(), sizeHint(), upda
Режим	isVisible(), isVisibleTo(), isMinimized(), isEnabled(), isEnabledTo(), isModal(), isWindow
Внутрішні стилі	style(), setStyle(), cursor(), setCursor() font(), setFont(), palette(), setPalette(), backgroundI
Функції фокусу клавіатури	setFocusPolicy(), focusPolicy(), hasFocus(), setFocus(), clearFocus(), setTabOrder(), setFo
Мишка і клавіатура	grabMouse(), releaseMouse(), grabKeyboard(), releaseKeyboard(), mouseGrabber(), keybc
Оброблення повідомлень	event(), mousePressEvent(), mouseReleaseEvent(), mouseDoubleClickEvent(), mouseMov
Системні функції	parentWidget(), window(), setParent(), winId(), find(), metric().
Підказка What's this	setWhatsThis()
Функції управління фокусом	focusNextChild(), focusPreviousChild()

QWidget надає кілька функцій, для роботи з **геометрією віджета**. Деякі з цих функцій працюють чисто з клієнтською частиною (тобто вікно без рамки), інші включають рамку. Включаючи рамку вікна: x(), y(), frameGeometry(), pos(), and move().
Виключаючи рамку вікна: geometry(), width(), height(), rect(), and size().
Слід зазначити, що різниця має значення тільки для віджетів верхнього рівня. *Для всіх дочірніх віджетів геометрія рамки еквівалентна геометрії клієнтської області віджету.*
Клас **QDialog** є ***базовим класом діалогових вікон.*** Діалогове вікно є вікном верхнього рівня, яке в основному використовується для короткочасних завдань і короткої комунікації з користувачем. QDialogs може бути модальним або немодальним. QDialogs може повертати значення, також вони можуть мати кнопки за замовчуванням.
QDialog наслідується від QWidget.
Реалізовують QDialog наступні віджети: QAbstractPrintDialog, QAxSelect, QColorDialog, QErrorMessage, QFileDialog, QFontDialog, QInputDialog, QMessageBox, QPageSetupDialog, QPrintPreviewDialog, QProgressDialog, QWizard
Qt MultimediaWidgets надають додаткові мультимедійні віджети і елементи управління. Класи розширюють можливості модулів Qt Multimedia і Qt Widgets. Щоб використовувати Qt Multimedia віджети в проекті, додайте цю директиву в файли C ++: **#include<QMultimediaWidgets>**

QCameraViewfinder	Віджет для видошукача камери
QGraphicsVideoItem	Графічний елемент, який показує відео створене за допомогою QMediaObject
QVideoWidget	Віджет, який показує відео створене медіа об'єктом
QVideoWidgetControl	Медіа контрол, який реалізовує відео віджет.

Model / View це технологія, яка використовується для **розділення даних від відображення віджетів, які обробляють ці дані.** Стандартні віджети не призначені для розділення даних від представлення і тому Qt має два різних типи віджетів. Обидва типи віджетів виглядають однаково, але вони взаємодіють з даними по-різному. Стандартні віджети використовують дані, які є частиною віджета. Model/View віджети працюють на зовнішніх даних (модель).
QListWidget-> QListView QTableWidget->QTableView QTreeWidget->QTreeView

16.Qt Database

База даних(БД) — впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно, та призначені для задоволення інформаційних потреб користувачів. Створення бази даних в Linux Розглянемо приклад з базою даних MySQL. Для цього потрібно виконати такі дії:

1. Відкрити термінал (Atl+Ctrl+T). Запустити MySQL-сервер від імені суперкористувача: user@user:~$ mysql -u root –p

2. Створити базу даних. Назвемо її “students”, тоді команда матиме такий вигляд: mysql> create database students;

3. Оскільки дана база даних створена користувачем root, то, відповідно, доступ до неї має лише root. Для того, щоб працювати з даною БД міг інший користувач, наприклад user, необхідно виконати таку команду: mysql> GRANT ALL ON students.* TO user@localhost IDENTIFIED BY "password"; Ця команда надає користувачу user повний доступ до усіх можливих дій із БД students, а також задає пароль доступу даного користувача до даної БД: password.

4. Вийти з mysql консолі ввівши команду quit

1.Запустити MySQL сервер для конкретного користувача, ввівши команду у терміналі:

```
user@user~$ mysql -u <username> -p
```

де username – ім’я користувача.

У нашому випадку:

```
user@user~$ mysql -u user -p
```

2. Вибрати БД командою:

```
mysql> use <dbname>;
```

де dbname – назва бази даних.

Для нашого прикладу:

```
mysql> use students;
```

Примітка. Для перегляду списку доступних даному користувачу баз даних використовують команду:

```
mysql> show databases;
```

3. Для створення таблиці(наприклад student_data) у вибраній БД потрібно виконати команду create table, описавши у цій команді стовбці таблиці. Приклад так

QtSql - один із модулів бібліотеки Qt. Цей модуль містить набір класів, що призначені для роботи з базами даних. Основні класи модуля QtSql: **QSqlDatabase** — використовується для з'єднання з базою; **QSqlQuery** — забезпечує виконання SQL-команд і опрацювання їх результатів; **QSqlTableModel** та **QSqlRelationalTableModel** — забезпечують високорівневий інтерфейс доступу до бази даних. Тобто використання цих класів не вимагає знання синтаксису SQL. Для підключення цього модуля використовують директиву: #include <QtSql>
Для створення об’єкту QSqlDatabase викликається функція QSqlDatabase::addDatabase(). Перший аргумент функції addDatabase() задає драйвер бази даних, до якої здійснюється підключення. У даному випадку це MySQL. Далі у функції **createConnection**() вказується назва бази даних, а також ім’я і пароль користувача, який здійснює підключення. Після цього відбувається відкриття з’єднання. Якщо функція open() завершується невдачею, то з'єднання з базою даних не вдалося встановити і, відповідно, результат createConnection() буде false.
Для виконання SQL-запитів у QT використовується клас **QSqlQuery**.
QSqlQuery query("SELECT name, surname, faculty, course, email FROM student_data WHERE course = 2"); Результатом виконання запиту SELECT буде набір записів.
Для додавання записів у базу даних використовується запит INSERT
Qt дозволяє виконання **транзакцій** у тих базах даних, де вони передбачені. Для запуску транзакції застосовується метод **transaction**() об’єкту **QSqlDatabase**, який є з’єднанням з базою даних. Для завершення транзакції потрібно викликати або функцію commit(), або функцію rollback().
Клас **QSqlTableModel** Цей клас дозволяє здійснювати SQL-запити(SELECT, INSERT, UPDATE, DELETE) без знання синтаксису SQL.
Для створення декількох з’єднань із базою потрібно для кожного з них вказати назву. Саме за нею відбуватиметься звертання до відповідного з'єднання. Це можна зробити передавши назву з’єднання другим аргументом функції addDatabase().
Закриття зєднань:

1. **void QSqlDatabase::close ()** Закриває з'єднання з базою даних, звільняє зайняті ресурси і робить недійсними всі існуючі об'єкти QSqlQuery, які використовувалися разом з базою даних. Це спрацює для всіх копій об'єкта QSqlDatabase.
2. **void QSqlDatabase::removeDatabase (const QString & connectionName) [static]** Прибирає з'єднання connectionName зі списку з'єднань з базою даних. Увага: Під час виклику цієї функції не повинно бути відкритих запитів до даної базі даних, в іншому випадку відбудеться витік ресурсу (resource leak).

Ось правильний варіант реалізації:

```
{
    QSqlDatabase db = QSqlDatabase :: database ( "students");
    QSqlQuery query ( "SELECT name, email FROM student_data", db);
}
// Обидва, "db" і "query", знищуються при виході з їхнього блоку
QSqlDatabase:: removeDatabase ( "sales"); // правильно
```

17.Qt Layout Management

Способи компонування Існує 3 основних способи компонування дочірніх віджетів на формі: абсолютне позиціонування; ручне компонування; використання менеджерів компонування.
Абсолютне позиціонування не є гнучким способом компонування віджетів. Використання цього способу компонування передбачає фіксований розмір форми та задання розмірів і позицій дочірніх віджетів форми у коді програми.
При використанні **ручного компонування** позиції і розміри віджетів знову ж таки встановлюються у коді програми, але при збільшенні чи зменшені розмірів основної форми позиції і розміри дочірніх віджетів форми змінюються відповідно до нових розмірів. Така поведінка досягається перевизначенням функції форми resizeEvent() для встановлення геометричних розмірів і позицій своїх дочірніх віджетів.

Менеджери компонування

На відміну від попередніх цей спосіб пропонує абсолютно простий підхід до компонування віджетів і тому він вважається найзручнішим. Менеджери компонування самі обирають оптимальний розмір і позицію для віджета, а також автоматично оновлюють компонування, коли змінилися шрифт, вмістиме чи розмір форми.

У *Qt* передбачено 4 базові менеджери компонування:

- *QHBoxLayout* - елементи розміщуються горизонтально в один рядок;
- *QVBoxLayout* - елементи розміщуються вертикально в один стовпець;
- *QGridLayout* - елементи розміщуються на двовимірній сітці, при цьому один елемент може займати декілька сусідніх клітинок;
- *QFormLayout* - елементи розміщуються по 2 на рядок у стилі форми: назва - поле.

Батьком елементів, які знаходяться в компонуванні є елемент, на якому встановлюється це компонування. ***Батьком віджету може бути тільки віджет, а не компонування.***
QHBoxLayout, QVBoxLayout та QGridLayout пропонують методи addLayout, які аналогічні до відповідних методів addWidget для цих класів. **QFormLayout** пропонує перевантажені варіанти методу addRow: void addRow (QWidget * label, QLayout * field) void addRow (QLayout * layout) Перший метод другим параметром приймає компонування, і розміщує його в рядку поруч з переданим віджетом, а другий метод - приймає лише компонування, яке охоплюватиме весь рядок.

18.Qt Input/Output and Networking

Qt може завантажувати і зберігати дані у вигляді простого тексту, XML або в бінарних форматах. Qt обробляє локальні файли використовуючи свої власні класи, віддалені файли - за допомогою FTP і HTTP протоколів. Основою структури обробки файлів в Qt є клас QIODevice – основний базовий клас для файлів, сокетів та інших.
QIODevice - це абстрактний клас, узагальнюючий пристрій введення / виведення, який містить віртуальні методи для відкриття і закриття пристроїв введення / виведення, а також для

читання і запису блоків даних або окремих символів. Реалізація конкретного пристрою відбувається у наслідуваних класах.
У Qt є чотири класи, які наслідують клас QIODevice: **QFile** - клас для проведення операцій з файлами; **QBuffer** - дозволяє записувати і зчитувати дані в масив QByteArray, начебто це пристрій або файл; **QAbstractSocket** - базовий клас для мережевої комунікації за допомогою сокетів. **QProcess** - цей клас надає можливість запуску процесів з додатковими аргументами і дозволяє обмінюватися інформацією з цими процесами за допомогою методів, визначених у QIODevice.
Клас **QFile** підтримує файли великого об'єму та з довгою назвою. Класи **QDir** та **QDirIterator** використовуються для зчитування та перегляду вмістимого директорії.
QFileInfo надає детальну інформацію про файл(напр. його розмір, право на доступ, час останньої модифікації.)
Qt обробляє файли як об'єкти. Файл представляється у вигляді об'єкту класу QFile. Для перевірки існування класу використовується метод exists(). Якщо файл існує, його можна видалити методом remove() . Для читання / запису файл потрібно відкрити через метод open(). Метод повертає true / false. Файл можна відкрити в таких режимах: IO_ReadOnly IO_WriteOnly IO_ReadWrite IO_Append
Використання stream класів полегшує читання / запис файлів. Для обробки тексту використовують **QTextStream**. Клас QTextStream призначений для читання текстових даних. В якості текстових даних можуть виступати не тільки об'єкти, вироблені класами, наслідувані від QIODevice, а й змінні типів char, QChar, char *, QString, QByteArray, short, int, long, float і double.
Щоб зчитати текстовий файл, необхідно створити об'єкт типу QFile і зчитати дані методом QTextStream :: readLine ()
Методом **QTextStream :: readAll** () можна зчитати відразу весь текстовий файл в рядок. Наприклад: QFile file ("myfile.txt"); QTextStream stream (& file); QString str = stream.readAll ();
Для обробки бінарної інформації використовують клас **QDataStream**. Клас QDateStream є гарантом того, що формат, в якому будуть записані дані, залишиться платформонезалежним

платформонезалежного представлення шляху **Qt** використовує клас **QDir**.

Основні методи:

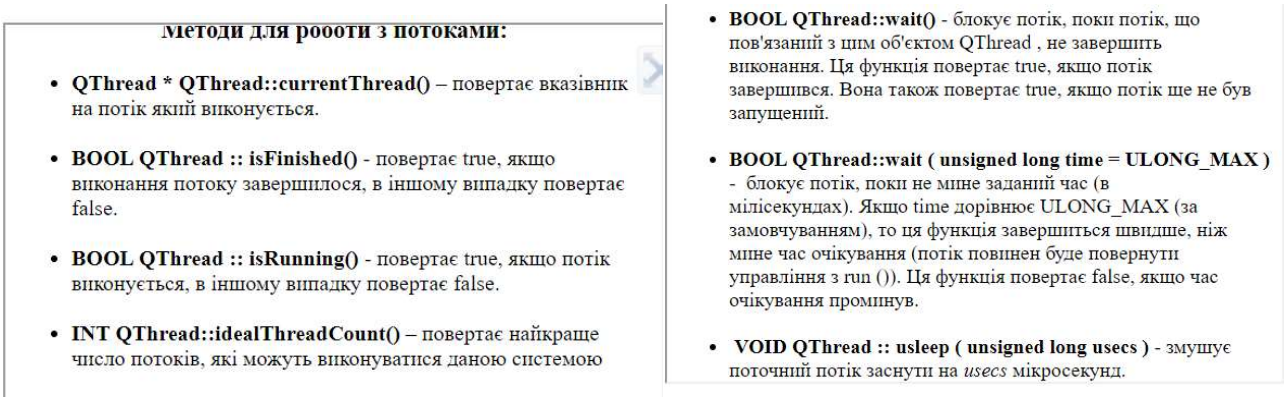
- **current()** - повертає **QDir**, що відповідає поточній для програми папці.
- **root()** - повертає кореневий каталог
- **home()** - повертає поточну home папку користувача
- **exists()** – перевіряє чи існує заданий шлях
- **cd(dirName)** – замінює **QDir** директорію на **dirName**. Повертає булівське значення(trueякщо нова директорія існує і доступна для зчитування)
- **cdUp()** – переходить на директорію «вище». Повертає булівське значення(**true** якщо нова директорія існує і доступна для зчитування)

- **isRoot()** – перевіряє чи директорія є кореневим каталогом.
- **mkdir(const QString & dirName)** – створює піддиректорію **dirName**. Повертає **true**,якщо успішно, інакше - **false**.
- **rmdir(const QString & dirName)** – видаляє директорію **dirName**. Повертає **true**, якщо успішно, інакше - **false**. Для успішного видалення директорія повинна бути пустою.
- **rename()** – перейменовує файл або директорію. Повертає **true**, якщо успішно, інакше - **false**.
- **setSorting(flags)** - встановлює порядок сортування вмістимого директорії (для **entryList()** та **entryInfoList()**) відповідно до **flags**.
- **flags** можуть бути такими: **QDir::Name**, **QDir::Time**, **QDir::Size** та **Unsorted**.
- **setFilter(filters)** - встановлює фільтр для **entryList()** та **entryInfoList()**

Qt забезпечує кросплатформенний інтерфейс для написання TCP/IP клієнтів та серверів, підтримуючи IPv4 та IPv6. Клас QTcpSocket забезпечує асинхронне буферизоване TCP з'єднання. Користувацькі TCP сервери можуть бути реалізовані через підкласи QtcpServer. Підтримка проксі сервера здійснюється через клас QNetworkProxy. Клас QAbstractSocket є платформонезалежною оболонкою для оригінальних сокетів API. Менеджмент - QNetworkAccessManager, Інформацію про мереживний інтерфейс - QNetworkInterface.

19. Qt Standard Dialogs
20. Qt Template Library
21. **Qt Threading Classes**

Щоб створити багатопоточне виконання в Qt потрібно створити підклас класу QThread і перевизначити його функцію run().



У кожного потоку є **пріоритет,** який вказує процесору, як має протікати виконання потоку по відношенню до інших потоків. Пріоритети розділяються по групах:
в першу входять чотири найбільш часто вживаних пріоритети. Процесорний час для них розподіляється за зростанням: **IdlePriority, LowestPriority, LowPriority, NormalPriority.** Вони підходять для вирішення *завдань, для яких процесор потрібний тільки час від часу, наприклад, для фонового друк*у або для яких-небудь нетермінових дій;
в другу групу входять два пріоритети: **HighPriority , HighestPriority .** Користуйтеся такими пріоритетами з великою обережністю. Зазвичай ці потоки більшу частину часу очікують якихось подій; в третю входять два пріоритети: **TimeCriticalPriority, InheritPriority** . Потоки з цими пріоритетами потрібно створювати у випадках крайньої необхідності. Ці пріоритети потрібні для програм, які безпосередньо зв'язані з *апаратурою*
якщо два потоки одночасно намагаються отримати доступ до однієї глобальної змінної, то результат, звичайно, буде невизначений. Для таких випадків у Qt є наступні класи: **Qmutex(захист, блокується для інших потоків) QmutexLocker(заб обробку винятків) QSemaphore(**Semaphore є узагальненням mutex. Як і mutex, вони служать для захисту критичних секцій, щоб доступ до них одночасно могло мати певне число потоків. Всі інші потоки зобов'язані чекати. *починають діяти з встановленого значення лічильника***) QWaitCondition(**забезпечує можливість координації потоків. Якщо потік має намір дочекатися розблокування ресурсу, то він викликає метод wait() і, тим самим, входить в режим очікування) **– синхронізація потоків**
QFuture представляє результат асинхронної компіляції

22.Qt XML Classes

QtXml – це модуль який забезпечує роботу з XML. Підтримує різні підходи (API) трактування XML, а саме: SAX, DOM.
DOM (Document Object Model) - це стандарт API, для розбору XML документів, розроблений у World Wide Web Consortium (W3C). У Qt реалізовано DOM Level 2 для читання, зміни і запису XML-документів. DOM представляє XML файл в пам'яті у вигляді деревовидної структури. Є можливість переміщатися по цій структурі у будь-якому напрямку. Програма може змінити вміст дерева і зберегти його назад у файл.
В Qt, імена класів вузлів починаються із префікса QDom. Таким чином, клас QDomElement представляє вузол Element, а QDomText - вузол Text.
SAX (Simple API for XML) - це стандарт програмного інтерфейсу з відкритим вихідним кодом, який забезпечує читання документів XML.
Qt забезпечує побудований на основі інтерфейсу SAX парсер документів XML - **QXmlSimpleReader**. Він не передбачає перевірку правильності документів. Цей парсер розпізнає добре сформовані документи XML і підтримує простір імен XML.
QXmlStreamReader є більш швидкою та зручною заміною SAX парсеру (клас QXmlSimpleReader). У деяких випадках він також може бути швидшою та простішою альтернативою у застосунках що використовують DOM модель (QDOMDocument). **QXmlStreamReader** може зчитувати дані з масиву байтів (QByteArray), текстового рядку чи об'єктів що реалізують інтерфейс QIODevice.
XmlPatterns — це модуль що надає підтримку XPath, XQuery, XSLT та XML Schema Validation.

23.Python
24.Perl
25.Java
26..net Core under linux, native builds

27.Регулярні вирази

Регулярні вирази (англ. regular expression) -- це рядки, що містять символи та метасимволи, і описують шаблон деякого рядка. RegEx використовують для пошуку в тексті, для перевірки структури тексту (напр. валідація введених даних) тощо. Синтаксис регулярних виразів залежить від інтерпретатора, що використовується для їхньої обробки. Звичайні символи у RegEx позначають самі себе, їх ще називають літералами.
Метасимволи - це символи, що позначають такі поняття як кількість, розсташування чи типи символів у рядку. Крапка (.) -- метасимвол, що позначає якийсь один довільний символ Зворотний слеш \ перед метасимволом перетворює його на звичайний символ. Коли треба в тексті знайти крапку, то використовують \.

Класи -- це послідовності символів у квадратних дужках, які задають область допустимих значень деякого одного символа у регулярному виразі. Розглянемо приклади:

- **a[0123456789]b** -- рядок починається з a, закінчується літералом b, а між ними може бути одна цифра;
- **a[0-9]b** -- компактніший запис попереднього прикладу;
- **[A-Za-z]** -- літера довільного регістру. До речі, клас **[A-z]** це не те ж саме: ані в таблицях UNICODE, ані в ASCII послідовність малих латинських літер не йде безпосередньо за послідовністю великих, між ними є ще інші символи;
- **[\[\] \-]** -- один символ: або відкриваюча квадратна дужка, або закриваюча, або дефіс;
- **[.]** -- позначає крапку;
- **[^0-9]** -- клас-заперечення, позначає довільний символ окрім цифри;
- **[^a-zA-Z]** -- клас-заперечення, позначає довільний символ окрім букви;
- **[\^a-zA-Z]** -- символ ^ або латинська буква (\^ всередині класу позначає символ ^);
- **[^\^]** -- будь-який символ, окрім ^

Метасимволи, що відповідають класам

- **\d** -- (digit) те саме, що **[0-9]**, тобто одна цифра.
- **\D** -- те саме, що **[^0-9]**, тобто один довільний символ, але не цифра.
- **\w** -- (word character) те саме, що **[A-Za-z0-9_]**, буква, або цифра, або нижнє підкреслення;
- **\W** -- те саме, що **[^A-Za-z0-9_]**, довільний символ, але не цифра, не літера і не підкреслення;
- **\s** -- (space) пропуск, або символ табуляції, або символ повернення каретки(carriage return), або символ зміни рядка (line feed);
- **\S** -- довільний символ окрім тих, які описує **\s**.

Квантифікатори

Квантифікатори -- це метасимволи або послідовності метасимволів, які вказують, скільки разів має повторюватися шаблон.

- **[A-Za-z]{2}** -- описує довільні комбінації з двох літер довільного регістру: **aA, mr, Zh** тощо;
- **[A-Za-z]{2,5}** -- рядки довжиною від 2 до 5 букв, що складаються з латинських літер довільного регістру;
- **[A-Za-z]{0,5}** -- відповідно рядки довжиною від 0 до 5 букв;
- **[A-Za-z]{5,}** -- рядки довжиною 5 і більше букв;
- **[A-Za-z]?** -- те саме, що **[A-Za-z]{0,1}**, тобто порожній рядок або одна буква;
- **[A-Za-z]*** -- те саме, що **[A-Za-z]{0,}**, тобто порожній рядок або рядок довільної довжини;
- **[A-Za-z]+** -- те саме, що **[A-Za-z]{1,}**, тобто одна буква або рядок довільної довжини;
- **[{}] {2,4}** -- рядок, що може містити від двох до чотирьох фігурних дужок;
- **\{ \}ab** -- відповідає рядку **{}ab**;
- **.*** -- рядок довільної довжини, що складається з довільних символів.

Квантифікатор є "жадібним", тобто перш за все задає рядки що відповідають шаблону і мають найбільшу довжину, а далі по спаданню довжини. Першим результатом пошуку рядка за шаблоном **.*** буде весь текст, в якому шукають. Щоб шукати від найменших до найбільших рядків використовують "лінивий" квантифікатор **.*?**

Альтернатива

Використовують, коли потрібно задати умову "або". Розглянемо приклади

- **(cat|dog)** -- рядок **cat** або рядок **dog** ;
- **([cat]|[dog])** -- рядок з двох символів, перший -- **c** або **a** або **t**, другий -- **d** або **o** або **g**, тобто **cd**, **cg**, **ao** тощо.

Круглі дужки

Можна групувати частини шаблону всередині за допомогою круглих дужок і до них застосовувати квантифікатори. Наприклад, **\w+(\s+\w+)*** -- одне або більше слів (слово **\w+**), розділені одним або більше роздільними символами класу **\s**.

Метасимволи, що описують межі

- **^**, **\A** -- початок рядка;
- **$**, **\Z**, **\z** -- кінець рядка;
- **^$** -- порожній рядок;
- **^.*$** -- весь текст;
- **\b, \B, \<, \>** -- межі слова;
- **\G** -- позиція, де попереднє співпадіння закінчилось.

Email **^[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z_+])*@([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,9}$** Regular Expression Pocket Reference (автор Tony Stubblebine, Second Edition, видавництво O'Reilly).