Загальні вимоги до виконання лабораторних робіт

Обладнання

Для виконання лабораторних робіт Вам потрібен комп'ютер, на якому встановлена будь-яка операційна система із сімейства UNIX. Передбачається виконання робіт в комп'ютерній лабораторії ФТІ на системі Solaris 10.

Для самостійної підготовки рекомендуються вільно (безкоштовно) розповсюджувані операційні системи: Solaris 10, FreeBSD, Debian або одна з версій Linux.

Дистрибутив системи Solaris 10 можна вільно завантажити з Інтернету (www. sun com). Типовий дистрибутив розміщується, як правило, на одному DVD, чи на 5 дисках. Зверніть увагу, що для виконання лабораторних робіт Вам не потрібна багатовіконна система X (X Window System), як і інші графічні системи (Gnome, CDE), тому фактично дистрибутив містить 99,9% зайвого для цього курсу лабораторних робіт програмного забезпечення.

Інші дистрибутиви Linux можна також завантажити з Internet.

Матеріали по Solaris 10 в електронній формі запитуйте у викладача, або на сайті docs.sun.com

Що стосується FreeBSD, її добре знають і дуже поважають системні адміністратори. Повний дистрибутив разом з усім програмним забезпеченням займає 4 повних CD-ROM, але знайти її на ринку важко. Зверніть увагу, що для FreeBSD також основний об'єм програмного забезпечення – це графічні багатовіконні системи і все, що працює під їх керуванням. Серйозних вимог до апаратного забезпечення немає. Якщо не використовувати графічних систем, не буде проблем навіть із дуже застарілим комп'ютером (486).

Вимоги до виконання робіт

Методичні вказівки до кожної роботи складаються з завдання для самостійної підготовки й завдання для виконання. Завдання для самостійної підготовки може виконуватись без комп'ютера (якщо є роздруковані матеріали довідкової системи). Воно передбачає ознайомлення з теоретичним матеріалом і завданням до виконання, а також підготовку до виконання практичної частини роботи.

Деякі теоретичні відомості містяться безпосередньо в методичних вказівках. Але не слід обмежуватись цією далеко не повною інформацією. Зверніться до рекомендованої літератури, багато додаткових відомостей можна знайти в Internet, зокрема на сайті docs.sun.com, www . citforum ru та www. google com Докладні довідкові дані по кожній (майже кожній) команді

UNIX можна одержати, набравши на клавіатурі `man <необхідна_команда>`.

Роботи виконуються в комп'ютерній лабораторії в присутності викладача. Допускається виконання лабораторних робіт в ході самостійної роботи в будь-якому іншому місці за умови дотримання правил оформлення звітів і графіку здачі робіт.

Деякі завдання виконуються відповідно до номера варіанта, який визначається за останньою цифрою номера залікової книжки студента (1 = варіант №1, 2 = варіант №2, …, 9 = варіант №9, 0 = варіант №10). При виконанні роботи студент повинен фіксувати все, що відбувається на екрані монітора комп'ютера.

Зверніть увагу на те, що в індивідуальних завданнях часто зустрічаються певні слова, рядки, вирази в кутових дужках. Це означає, що Ви повинні підставити замість цього виразу разом із дужками значення цього виразу. Наприклад, Вас звуть Сергій, тоді замість `студент_<ваше ім'я>` Ви повинні написати (набрати на клавіатурі) `студент_Сергій`.

Файли та каталоги, створені Вами в ході виконання лабораторних робіт, зберігайте до завершення всього циклу лабораторних робіт, бо вони можуть використовуватись в наступних роботах.

Для здачі лабораторної роботи студент повинен подати викладачу оформлений звіт з цієї роботи, що відповідає викладеним нижче вимогам. Варіант завдань, що виконані студентом, повинен відповідати номеру його залікової книжки. Студент повинен дати правильні відповіді на додаткові запитання за матеріалами цієї лабораторної роботи. Зверніть увагу на те, що студент повинен володіти всім теоретичним і практичним матеріалом, а не тільки тією його частиною, яка безпосередньо входила до індивідуального завдання.

Вимоги до оформлення звіту

Звіт повинен містити:

1. назву лабораторної роботи, тему і мету лабораторної роботи;
2. номер варіанту і зміст індивідуального завдання;
3. дату і місце виконання роботи, коротку характеристику системи, на якій було виконано роботу (клас комп'ютера, версію ОС і т.і.);
4. протокол виконання роботи, де чітко позначено, що вводить користувач з клавіатури, і що в результаті з'являється на екрані (необхідно показати всі результати; якщо іде запис даних в файл, вміст файлу також слід роздрукувати);

5. (**обов'язково!!!**) висновки по роботі, які повинні містити саме висновки, які студент може зробити в результаті виконання цієї роботи, а не констатацію факту "в результаті виконання роботи я навчився…"; у висновках може бути зроблена оцінка наявності певних функцій операційної системи, якості або гнучкості їх реалізації, зручності інтерфейсу і т.і.;
6. на кожній сторінці (в верхньому або нижньому колонтитулі) прізвище студента і номер групи або залікової книжки.

Звіт з кожної лабораторної роботи повинен мати титульний лист, оформлений згідно стандартних вимог, як у наведеному прикладі.

Загальні відомості про систему UNIX

Системи, які узагальнюються під назвою UNIX, є багатозадачними багатокористувацькими системами з розділенням часу. Насправді, це досить різні системи, які зберегли подібний інтерфейс користувача і програмний інтерфейс (інтерфейс системних викликів), а також схожі принципи побудови файлових систем.

Всі ці системи є нащадками однієї системи, яку було розроблено на початку 70-х років XX століття. Шляхи розвитку цієї системи спочатку розійшлися, потім було проведено велику роботу по відновленню сумісності систем. Тепер розрізняють дві гілки розвитку систем UNIX: System V (V – римська цифра "5") розробки підрозділів AT&T (її скорочено називають SVRx, де x – номер версії, тепер поширена SVR4) і BSD UNIX компанії Berkeley Software Distribution. Ще одна гілка розвитку – OSF/1 (Open Software Foundation), що розробляється як конкурент System V, фактично наближена до BSD UNIX, хоча містить ряд особливостей, характерних для System V. Також проміжне положення займає популярна система Linux. Правом на торгову марку "UNIX" володіла AT&T, потім ці права (разом з патентами і правами на вихідні коди системи) були продані Novell, а далі – SCO (Santa Cruz Operation). Інші системи мають власну назву, що, як правило, дещо співзвучна UNIX. Коли бажають підкреслити, що мають на увазі різні операційні системи, що мають спільні риси, часто називають їх системи *IX (UNIX, Linux, AIX, Minix, Xenix, Irix і т.д.). Найбільш помітні для користувача відмінності систем:

- традиційні командні інтерпретатори – sh, ksh для SVR4 і csh для BSD;
- файлова система – S5 для SVR4, UFS для BSD, EXT2 для Linux.

Також є численні відмінності в іменах і розміщенні важливих конфігураційних файлів. Найбільш відомі й розповсюджені операційні системи, що їх можна віднести до SVR4: Solaris (Sun Microsystems), IRIX (Silicon Graphics), SCO UNIX. За зовнішніми ознаками до SVR4 також близька система Linux. До BSD UNIX відносяться в першу чергу системи розробки власне BSD, в тому числі відкриті проекти OpenBSD і FreeBSD (остання система є основною, що рекомендована для виконання цих лабораторних робіт). До системи BSD також дуже близька система Digital UNIX розробки DEC (фактично вона є системою OSF/1).

Описи лабораторних робіт базуються на системі Solaris 10.

Для того, щоби користувач міг розпочати роботу в системі, він повинен проідентифікувати себе і підтвердити ідентифікацію паролем. Для цього адміністратор повинен заздалегідь створити в системі обліковий запис (іноді кажуть – бюджет, англ. – *account*) цього користувача. Інформацією, що ідентифікує користувача є *login* або *userid* (*login* – це ім'я-ідентифікатор з

одного слова, який використовується при взаємодії користувача з системою, *userid* – числовий ідентифікатор користувача, яким користується система). Інформацією, що автентифікує (підтверджує тотожність) користувача є пароль (рядок символів, на який у залежності від конфігурації накладаються деякі обмеження).

Після входу в систему користувач опиняється в графічному середовищі Gnome чи CDE. Щоб потрапити до командної оболонки, в якій безпосередньо будуть виконуватися всі лабораторні роботи, слід запустити термінальне вікно - Terminal. В залежності від типу командної оболонки (*shell*), яка дозволяє йому вводити команди й інтерпретує їх. Оболонка може запускати на виконання файли, що містять програмний код. Також вона може виконувати послідовність команд, що міститься в текстових файлах – так званих пакетних файлах, файлах сценаріїв, або, як їх ще називають, скріптах (від англ. – *script*). До складу операційної системи, крім ядра, входить велика кількість окремих програмних файлів – системних утиліт. Переважна більшість команд системи UNIX є так званими зовнішніми командами, тобто фактично вони означають запуск тієї чи іншої утиліти. Утиліти використовують програмний інтерфейс системи UNIX (так званий інтерфейс системних викликів) і звертаються до внутрішніх структур ядра системи. Вони забезпечують гнучкі можливості керування файловими системами, процесами, контролю стану системи.

Докладну довідку щодо кожної команди UNIX ви можете отримати використовуючи довідкову систему `man`, давши команду:

```
man <команда>
```

Довідковий матеріал розбито на розділи. В описах (як у довідковій системі, так і в літературі) прийнято супроводжувати посилання на певну команду, системний виклик, функцію, номером відповідного розділу довідкової системи. Наприклад, `man(1)`, `passwd(1)`, `passwd(4)`. Нумерація розділів трохи різна в SVR4 і BSD:

Розділ	BSD	SVR4
Команди загального призначення (довідник користувача)	1	1
Системні виклики (довідник програміста)	2	2
Бібліотечні функції (довідник програміста)	3	3
Інтерфейси ядра (довідник програміста)	4	7
Формати конфігураційних і системних файлів	5	4
Різнобічна інформація	7	5
Системні утиліти (довідник системного	8	1M

адміністратора)

Лабораторна робота №1. Структура файлової системи UNIX, основні команди, команди роботи з файлами.

Мета

Оволодіння практичними навичками роботи в системі UNIX. Знайомство із структурою файлової системи, основними командами роботи з файлами.

Завдання для самостійної підготовки

1. Вивчити:
   - команди входу в систему, зміни пароля, одержання системної підказки, виводу календаря і зміни дати;
   - організацію і структуру файлової системи UNIX, обмеження на імена файлів;
   - типи файлів, каталоги і посилання;
   - системні каталоги;
   - створення, видалення, копіювання і перегляд умісту файлів.

2. Ознайомитись з такими командами UNIX:
   `man, passwd, date, cat, more, wc, who, ls, cd, cal, cp, mv, mkdir, rm, rmdir`

3. Відповідно до завдання підготувати послідовність команд для його виконання.

Довідковий матеріал

Якщо ОС UNIX встановлена на персональному комп'ютері, то на ньому підтримується певна кількість так званих віртуальних терміналів, між якими можна переключатись комбінаціями клавіш Alt+F#, де F# – одна з функціональних клавіш. Користувач може працювати в системі, використовуючи одночасно кілька терміналів. Для здійснення входу в систему, користувач повинен спочатку ввести свій ідентифікатор (*login*). Як правило, він вводиться у відповідь на запрошення системи такого вигляду:

```
Login:
```

Якщо цього запрошення не екрані немає (і не діє екранна заставка – *screensaver*), то це означає, що даний термінал не очікує входу користувача. Три найтиповіші причини:

1. Термінал апаратно заблоковано клавішею “Scroll Lock” (це легко визначити за відповідним світлодіодом на клавіатурі) – розблокуйте термінал.
2. Термінал зайнятий, тобто з ним пов’язана деяка програма – слід вийти з цієї програми і з командної оболонки, для чого потрібен певний досвід (можуть спрацювати клавіша q, комбінації клавіш Ctrl+C, Ctrl+D, але іноді це не допоможе – див. Лабораторну роботу № 5).
3. Термінал використовується виключно для виведення на екран важливих системних подій, при цьому на ньому можна вводити команди і будь-які символи, але реакції на це не буде – слід перейти на іншу консоль комбінацією клавіш Alt+F#, де F# – одна з функціональних клавіш, крім F1.

Після введення ідентифікатора система запитує в користувача пароль:

```
Password:
```

Під час введення пароля символи на екрані не відображаються. Якщо ідентифікатор і пароль користувача були введені правильно, система здійснює *авторизацію* користувача, тобто, надає йому певні повноваження, необхідні для роботи в системі. Як правило, після цього користувач опиняється в середовищі командної оболонки (англ. – *shell*). При цьому на екрані з’являється так зване запрошення командної оболонки (найчастіше – символ ‘$’ або ‘>’, також можна довільно змінити запрошення). Командна оболонка приймає команди, що вводяться з клавіатури, інтерпретує їх і виконує відповідні дії. Ці дії можуть полягати у запуску певних утиліт із заданими у командному рядку параметрами. Крім того, командна оболонка надає користувачу певний додатковий сервіс, наприклад, дозволяє виконувати редагування команди (курсор можна переміщати вправо чи вліво, додавати або знищувати символи під курсором), в деяких оболонках можна легко відтворити попередні команди (клавіші переміщення курсору вгору та вниз), а також користуватись підказками щодо імен наявних файлів (клавіша Tab). Докладніше про ці та інші сервісні можливості можна дізнатись в довідковій системі man, а також в будь-якій доступній книзі про системи UNIX чи Linux.

В кожній системі UNIX є в наявності кілька різних командних оболонок. Не будемо наводити історію їхнього розвитку і причини існування кількох оболонок одночасно (див. відповідну літературу). Просто слід зазначити, що практично в кожній системі можна знайти звичну оболонку, або подібну до неї. Найбільш стандартною, що присутня в усіх системах, є оболонка Bourne (Bourne shell), яка стала основою стандарту POSIX shell. Ця оболонка пропонує мінімальний сервіс для користувача, і для інтерактивної

роботи незручна. Її файл – `/bin/sh`. Існують альтернативні оболонки. Одна з них – Korn shell (`/bin/ksh` або `/usr/bin/ksh`), яку рекомендують як основну для роботи користувача в багатьох системах, здебільшого у версіях System V Release 4 (SVR4). Інша – C shell (`/bin/csh`), вона досить сильно відрізняється синтаксисом багатьох команд, але дуже зручна для користувача і програміста, особливо для тих, хто добре знайомий з мовою програмування C. Як правило, нею користуються адміністратори систем BSD Unix. Також існують, але не у всіх системах є доступними, більш сучасні оболонки: TC shell (`/bin/tcsh`) – розвиток C shell, і Bourne again shell (`/bin/bash` або `/usr/bin/bash`) – розвиток Bourne shell. "Bourne again shell" зберігає програмну сумісність з `sh`, але включає в себе всі можливості `ksh`, ця оболонка використовується як стандартна в Linux.

В подальшому ми будемо використовувати синтаксис оболонки `sh` (запрошення має вигляд ' `$`'), а в деяких випадках порівнювати її з `csh` (запрошення має вигляд '>').

Командна оболонка, в якій розпочинає роботу користувач після входу в систему, визначається з файлу `/etc/passwd`. Це один з найголовніших конфігураційних файлів системи, який містить параметри облікових записів користувачів. Кожний рядок файлу відповідає певному користувачу, точніше, обліковому запису користувача, що розрізняється за ідентифікатором *login* або *userid*. Користувач в процесі роботи може запустити іншу командну оболонку, просто набравши її ім'я.

Усі команди, які можна ввести у рядку запрошення оболонки, належать до одної з таких категорій:

- вбудовані функції,
- функції, що визначені користувачем,
- зовнішні програми й утиліти.

Вбудовані функції реалізуються фрагментами програмного коду оболонки і виконуються найшвидше. Користувач може визначити свої функції (хоча таку можливість використовують нечасто). Якщо команда не є вбудованою функцією і не визначена як функція користувачем, тоді оболонка буде шукати файл з відповідним ім'ям і намагатись запустити його на виконання. Якщо ім'я файлу задано із шляхом до нього, то система намагається знайти його саме у тому каталозі, який явно задано у команді, але якщо ім'я файлу задано без шляху, то пошук файла здійснюється лише у тих каталогах, які задані системною змінною PATH. Для перегляду значення змінних їх слід набрати зі знаком $ попереду. Зокрема, якщо у цій змінній не задано пошук у поточному каталозі, і поточний каталог не є одним із тих, що

задано у цій змінній, то оболонка не побачить виконуваних файлів з поточного каталогу.

Розглянемо деякі основні команди. Слід зазначити, що немає штатного засобу, який надавав би користувачеві перелік доступних йому команд, тому основні команди необхідно пам'ятати.

Команда `man` форматує і відображає на терміналі сторінки довідкової системи. Відповідно до номерів розділів даються посилання на ту чи іншу сторінку довідника. Якщо є необхідність, можна вказати, в якому розділі треба шукати потрібну сторінку (Приклад 2). Приклади використання (системні запрошення не показані):

```
man passwd
man 7 mdoc
```

Команда `passwd`, що викликає однойменну утиліту, дозволяє користувачеві змінювати свій пароль входу в систему. Спочатку необхідно підтвердити свою автентичність, набравши свій пароль, а далі можна ввести новий пароль (це здійснюється двічі для уникнення випадкових помилок).

Команда `who` дозволяє визначити, хто ще працює в поточний момент в системі. Команда `who am i` нагадає вам, який ваш *login*.

Існують зручні команди визначення поточної дати й часу (`date`), а також виводу на екран календаря на будь-який місяць будь-якого року (`cal`).

Для того, щоби переглянути вміст текстового файлу, можна скористатись командою `cat` <*ім'я файлу*>, або `more` <*ім'я файлу*> (остання команда призначена для виводу інформації на екран посторінково, вона надає можливість "перегортати сторінки" вперед і назад). Існує команда `wc` <*ім'я файлу*> (word count – підрахувати слова), яка дозволяє підрахувати кількість рядків (`wc -l`), слів (`wc -w`) і символів (`wc -c`) у файлі. Створити текстовий файл можна командою `touch` <*ім'я файлу*>.

Розглянемо особливості файлової системи UNIX. Вся файлова система поєднується в єдине дерево каталогів, які починаються з кореневого каталогу, що має позначення ' /'. Всі зовнішні файлові системи (змінні носії інформації, мережеві диски і таке інше) монтуються у визначенні місця єдиного дерева файлової системи (див. Лабораторну роботу №7).

Як і в інших ієрархічних файлових системах, у файловій системі UNIX ім'я файлу повинно бути унікальним лише в межах одного каталогу (на відміну від MS-DOS/Windows, UNIX розрізняє великі і малі літери в назвах файлів). Для однозначної ідентифікації файлу в дереві каталогів слід указувати повний путь до файлу. Якщо путь починається з символу ' /' (наприклад, `/usr/local/bin/cal`), то він відраховується від кореневого

каталогу (абсолютний путь), а якщо з іншого символу – то від поточного каталогу, тобто того, в якому користувач знаходиться в поточний момент (відносний путь). Крім того, поточний каталог позначається символом ‘ **.** ’ (крапка), каталог, що знаходиться на один рівень вище, тобто батьківський каталог – символом ‘ **..** ’ (дві крапки). Крім того, існує спеціальне позначення для так званого домашнього каталогу користувача, тобто каталогу, з якого він починає свою роботу – ‘ ~’ (тильда). Домашній каталог для кожного користувача також задається у файлі `/etc/passwd`, за замовчуванням це `/export/home/`<*login*>, або `/usr/home/`<*login*>.

Для переходу з каталогу в каталог існує команда `cd` <*новий каталог*> (change directory – змінити каталог). Якщо використати цю команду без параметрів, відбудеться перехід в домашній каталог користувача. Наприклад, якщо домашній каталог користувача `/export/home/stud1`, в поточний момент він знаходиться в каталозі `/opt/staroffice8/program`, і бажає перейти в каталог `/opt/netbeans6`, він може скористатись однією з наведених нижче команд:

`cd /opt/netbeans6`
`cd ../../netbeans6`

Щоб потрапити до свого домашнього катологу слід набрати

`cd ~`

Слід зазначити, що відносні путі слід використовувати з обережністю. Для того, щоби перевірити, в якому каталозі знаходиться користувач, можна скористатись командою `pwd`.

Перегляд вмісту каталогів здійснюється за допомогою команди `ls`, а розширений варіант цієї команди `ls -l` дає також інформацію з таблиці індексних дескрипторів (див. Лабораторну роботу №2). Щоби скопіювати файл, використовується команда **`cp`** **<***файл-джерело***>** **<***призначення***>**. Для перенесення файлу з каталогу в каталог, а також для перейменування файлу, використовується команда `mv` **<** *файл-джерело***>** **<** *призначення***>**. В обох командах в якості параметра **<***призначення***>** може задаватись каталог призначення або ім’я файлу призначення. Крім того, число параметрів може бути більше двох. В такому випадку всі параметри, крім останнього, розглядаються як список імен файлів-джерел, а останній параметр може бути лише каталогом призначення. Створити каталог можна командою `mkdir`, видалити файл – командою `rm`, видалити каталог – командою `rmdir` або `rm -r`.

Крім звичайних файлів існують різні типи спеціальних файлів. З одним із них ми вже познайомились – це каталоги. Ще одним типом спеціальних файлів є так звані *посилання* (рос. – *ссылка*, англ. – *link*). В системі UNIX розрізняють два принципово різних типи посилань, хоча створюються вони

однією командою – `ln`. Перший тип – це так звані жорсткі посилання. Фактично вони є абсолютно рівноправними новими іменами вже існуючого файлу. Після створення такого посилання система не розрізняє, яке ім'я було первинне, а яке було створене як посилання. Спроба видалити такий файл призводить до того, що одне з його імен (те, за яким ми видаляємо файл), знищується, а інші (як і сам файл) залишаються. Тільки після видалення останнього з імен фактично знищується сам файл. Другий тип посилання – символічне посилання, яке створюють командою `ln -l`. Це спеціальний тип файлу, який містить в собі ім'я того файлу (або каталогу), на який він посилається.

Більшість команд, що застосовуються по відношенню до посилання, діють безпосередньо на файл, на який посилання здійснене. При цьому деякі послідовності команд можуть привести до небажаних наслідків. Наприклад, маємо файл `oldfile` і бажаємо перейменувати його в `newfile`. Це можна зробити як командою

```
mv oldfile newfile
```

так і послідовністю команд

```
cp oldfile newfile
rm oldfile
```

Результати будуть однакові. До речі, в одному командному рядку можна задати декілька команд, розділивши їх знаком '`;`', ці команди будуть виконуватись послідовно:

```
cp oldfile newfile; rm oldfile
```

Тепер уявимо, що маємо файл `targetfile` і посилання на нього `oldfile`. Команда

```
mv oldfile newfile
```

перейменує посилання, тобто тепер `newfile` буде посилатись на `targetfile`. Команда

```
cp oldfile newfile
```

скопіює не посилання, а сам файл `targetfile`, тобто під іменем `newfile` буде створено новий файл – копію `targetfile`. Наступна команда

```
rm oldfile
```

знищить старе посилання, не пошкодивши при цьому файл **`targetfile`**. Тобто замість одного файлу з посиланням на нього у нас утворилися два ідентичних файла, які абсолютно не пов'язані між собою.

Завдання до виконання

1. Завантажтеся в систему під вашим користувацьким ім'ям.
2. Поміняйте ваш пароль. Ваш новий пароль повинен включати в себе як частину номер Вашої залікової книжки.
3. Виведіть системну дату.
4. Підрахуйте кількість рядків у файлі:

Варіант	Файл
1, 2, 4	**/etc/passwd**
3, 10	**/etc/group**
6, 9	**/etc/profile**
5, 7, 8	**/etc/vfstab**

5. Виведіть на екран вміст відповідного файлу.
6. Виведіть календар на <1995+№варіанту> рік.
7. Виведіть календар на 1752 рік. Чи не помічаєте що-небудь цікаве у вересні? Поясніть.
8. Визначте, хто ще завантажений у систему.
9. Наберіть команду ping. Поясніть результат.
10. Скопіюйте *(скопіюйте, а не перемістіть, бо система перестане працювати коректно!)* файли

варіант	**файл 1**[1]	**файл 2**
1	**/bin/cat**	**/bin/at**
2	**/bin/cal**	**/bin/chmod**
3	**/bin/ls**	**/bin/chown**
4	**/bin/tee**	**/bin/file**
5	**/bin/more**	**/bin/gzip**
6	**/bin/date**	**/bin/gunzip**
7	**/bin/cp**	**/bin/ps**
8	**/bin/mv**	**/bin/csh**
9	**/bin/lpr**	**/bin/sh**
10	**/bin/find**	**/bin/ksh**

у ваш домашній каталог різними способами.

11. Створіть каталог `lab_1`.
12. *Скопіюйте* в нього з вашого домашнього каталогу копію файлу 1, яку ви отримали в п.10, під ім'ям `my_<ім'я файлу 1>`. *Перемістіть* в цей каталог з вашого домашнього каталогу копію файлу 2, яку ви отримали в п.10, перейменувавши його при цьому в `my_<ім'я вихідного файлу 2>`. За ім'я вихідного файлу слід брати саме ім'я файлу, без імен каталогів і шляху до файлу (інакше символ "/" буде

[1] Якщо файл 1 або 2 не знайдено в каталозі `/bin`, шукайте його в каталогах `/usr/bin`, `/sbin` або `/usr/sbin`

проінтерпретований операційною системою зовсім не так, як Ви очікуєте).

13. Перейдіть у свій домашній каталог і переконайтеся в тому, що все зроблено правильно.
14. Створіть каталог `lab_1_<№варіанту>` і перейдіть в нього.
15. Скопіюйте в каталог `lab_1_<№варіанту>` файл з п.4 під ім'ям `n<ім'я вихідного файлу>`.
16. За допомогою команд `cat` і `more` перегляньте його вміст.
17. Перейдіть у свій домашній каталог.
18. Видаліть каталог `lab_1_<№варіанту>`.

Лабораторна робота №2. Система розмежування доступу в UNIX та Solaris, права доступу до файлів і керування ними

Мета

Оволодіння практичними навичками керування правами доступу до файлів і їхній аналіз в ОС UNIX та Solaris

Завдання для самостійної підготовки

1. Вивчити:
   - поняття "право доступу" і "метод доступу";
   - атрибути доступу до файлів в UNIX;
   - перегляд інформації про права доступу;
   - зміна прав доступу.
2. Детально ознайомитись з довідкової системи `man` з такими командами UNIX:
   `ls –l, chmod, chown, umask`[2]**`, setfacl, getfacl.`**
3. Відповідно до завдання підготувати послідовність команд для його виконання.

Довідковий матеріал

Складовою кожної захищеної системи є система розмежування доступу. В системі UNIX контролюється доступ до файлів. Кожний файл має свого власника, а також відноситься до визначеної групи. Вся детальна інформація про файл, що включає тип файлу, ідентифікатори власника файлу та його групи, розмір файлу, час останньої модифікації файлу, інформацію щодо прав доступу до файлу, а також про його розміщення (номери блоків), міститься не в каталогах, як це зроблено в багатьох інших файлових системах, в тому числі FAT (MS-DOS, Windows), а в системній таблиці так званих індексних дескрипторів (*i-node*). Безпосереднє звернення до цієї таблиці заборонене. Каталоги містять лише ім'я файлу та індекс – посилання на відповідний запис в таблиці індексних дескрипторів. Саме завдяки такій організації, кожний файл в файловій системі UNIX може мати кілька абсолютно рівноправних імен (жорстких посилань), що в загальному випадку містяться в різних каталогах (це було розглянуто в Лабораторній роботі №1).

[2] Для команди **`umask`** не існує окремої сторінки **`man`**, оскільки це – вбудована команда **`shell`**, тому її опис (хоча й не повною мірою детальний) міститься в сторінках **`man`** для кожного з **`shell`** (**`sh`**, **`bash`**, **`csh`**, **`ksh`**, тощо).

Інформацію з таблиці індексних дескрипторів можна переглянути командою `ls -l` . При цьому для кожного файлу інформація видається у вигляді одного рядка. Перший символ – тип файлу: **‘ –’** – звичайний файл (текстовий або бінарний), **`d`** –каталог (*directory*), **`l`** – символічне посилання (*link*), **`c`** – символьний пристрій (*character device*), **`b`** – блочний пристрій (*block device*). Наступні дев’ять символів описують права доступу до файлу (див. далі). Далі надається інформація про кількість жорстких посилань на файл, власника файлу, групу, розмір файлу, дату останньої модифікації і останнє поле – ім’я файлу.

UNIX реалізує дискреційну[3] модель розмежування доступу, в якій для кожного файлу визначається, які права мають всі користувачі на доступ до файлу. Для цього з кожним файлом асоціюється спеціальна інформація, що містить ідентифікатор власника файлу, ідентифікатор групи файлу і права доступу до файлу. Права доступу поділяються на 3 частині: права власника, права групи і права всіх інших. У кожному класі користувачів виділено по 4 біти. Перші (молодші) три біти відповідають правам читання, запису й виконання, відповідно. Для каталогів право виконання трактується як право доступу до таблиці індексних дескрипторів на читання і запис, не маючи цього права неможливо зробити поточним цей каталог чи будь-який з його підкаталогів, неможливо ознайомитись і змінити права доступу до об’єктів цього каталогу, можна тільки переглядати його вміст, якщо є право читання. Навіть маючи право запису, без права виконання не можна змінити вміст каталогу. Навпаки, якщо є право на виконання, але не встановлено право на читання для каталогу, то неможливо переглянути вміст каталогу, але можна заходити в його підкаталоги чи звертатись до файлів, що містяться в ньому, якщо знати їхні імена.

Четвертий біт має різне значення в залежності від того, в якій групі прав доступу він установлений. Для групи прав власника цей біт називається SUID (Set-User-ID-on-execution bit), і якщо він установлений для файлу, що виконується, то цей файл виконується для будь-якого користувача із правами власника цього файлу. Якщо четвертий біт встановлений у групі прав доступу членів групи (SGID – Set-Group-ID-on-execution bit), то ця програма виконується для будь-якого користувача із правами членів групи цього файлу. Для каталогів SGID визначає, що для усіх файлів, створюваних у цьому каталозі, ідентифікатор групи буде встановлений такий же, як у каталогу (ці правила залежать від версії UNIX).

Четвертий біт у групі прав доступу всіх інших користувачів називається Sticky, на сьогоднішній день він використовується тільки для

[3] Більш докладну інформацію про моделі розмежування доступу Ви отримаєте з курсів “Теоретичні основи захисту інформації” і “Захист інформації в комп’ютерних системах”

каталогів і визначає, що користувачі, які мають право на запис в каталог, не мають права видаляти чи перейменовувати файли інших користувачів у цьому каталозі. Це необхідно для каталогів загального використання, наприклад `/tmp`.

Права доступу до файлів задаються при створенні файлу і в подальшому можуть бути змінені командою

chmod <*нові права***> <***файл(и)***>**

<*нові права***>** задаються двома способами. Перший – символьний. Використовуються такі позначення: **u** – власник файлу (*user*), **g** – група (*group*), **o** – всі інші (*others*), **a** – всі категорії користувачів одночасно (*all*). Після категорії користувачів задається дія: ‘ **+**’ – додати права до існуючих, ‘**-**’ – відібрати права (якщо існують), ‘ **=**’ – встановити права замість існуючих. Далі позначаються права: **r** – зчитувати (*Read*), **w** – записувати (*Write*), **x** – виконувати (*eXecute*). Можна формувати список зміни прав, розділяючи окремі категорії користувачів комами (без пробілів). Наприклад, команда

```
chmod u+rw,g=rx,o-w my_file
```

встановить для користувача права на зчитування і записування (право на виконання для користувача буде залежати від того, чи було воно встановлено раніше), для групи встановить права на зчитування і виконання (незалежно від того, які права були раніше), а для всіх інших гарантовано заборонить записування (права на зчитування і виконання будуть встановлені в залежності від того, чи були вони встановлені раніше).

Другий – числовий спосіб задавання прав доступу – зручніше використовувати, коли треба встановлювати нові права незалежно від попередньо встановлених. При цьому використовується представлення у восьмеричній системі числення, яке легко зрозуміти з наведеної таблиці:

Восьмеричне представлення	**Двійкове представлення**	**Права доступу**
0	000	**---**
1	001	**--x**
2	010	**-w-**
3	011	**-wx**
4	100	**r--**
5	101	**r-x**
6	110	**rw-**
7	111	**rwx**

Наприклад, команда

chmod 751 my_file

встановлює такі права доступу до файлу `my_file`: власнику – зчитування, записування, виконання, групі – зчитування і виконання, а всім іншим – лише виконання. Команда `ls -l` покаже для цього файлу таку інформацію про права доступу: `-rwxr-x--x` (перший "мінус" означає, що це звичайний файл).

За замовчуванням для всіх нових файлів, що створюються, встановлюються такі права: для файлів 666, тобто `rw-rw-rw-`, для каталогів – 777, тобто `rwxrwxrwx` (ми вже бачили вище, що наявність права на виконання дуже важлива для каталогів). Є особлива команда – `umask`, яка дозволяє зменшити права доступу, що встановлюються для нових файлів. Параметром цієї команди є восьмерична маска, аналогічна тій, що використовується при числовому способі задавання параметрів команди `chmod`, але у випадку команди `umask` права доступу, що задані нею, будуть відніматись від прав доступу, що були задані по замовчуванню для вже утвореного файлу. Команда `umask` не може додавати прав, тому нові файли ніколи автоматично не отримають право на виконання, в разі необхідності його треба буде додати вручну. Дія команди `umask` розповсюджується на всі файли всіх типів, що утворюються протягом поточної сесії користувача після цієї команди. Наприклад, після команди `umask 123` всі файли будуть утворюватись з параметрами доступу `rw-r--r--` (644), а всі каталоги – `rw-r-xr--` (654). За замовчуванням рекомендується використовувати команду `umask 22` (інтерпретується як `umask 022`).

Для зміни власника файлу існує команда `chown`. З міркувань безпеки, ця команда дозволяє встановлювати власником файла будь-кого лише системному адміністратору (*root*'у). Інші користувачі можуть лише привласнити файл собі, якщо вони мають на це права, встановлені правами доступу до файлу і каталогу.

Списки ACL

В современных системах UNIX для большей гибкости прав доступа введены дополнительные свойства объектов, такие как флаги для файлов и каталогов, списки управления доступом (ACL) для файлов и каталогов, аутентификация и авторизация с использованием различных служб аутентификации подобных TACACS и RADIUS, а также модули аутентификации и авторизации (pluggable authentication modules - PAM).

Стандартные права доступа в UNIX не всегда идеальным образом подходят для решения задач администрирования системы. Например, что сделает администратор, если доступ к файлу на чтение и запись надо дать

одной группе пользователей? Конечно, сделает эту группу владелицей файла, а затем делегирует право чтения и записи файла этой группе, вот так:

chgrp нужная_группа файл

chmod g=rw файл

А если надо дать право доступа только одному из членов этой группы? Тогда можно создать новую группу специально для этой цели, сделать ее членом только одного пользователя, и затем указать, что эта новая группа будет владеть файлом:

addgroup новая_группа

chgrp новая_группа файл

chmod g=rw файл

#редактируем /etc/group и добавляем в новую группу существующего пользователя

vi /etc/group

Такое ухищрение нам помогло, но что делать, если надо дать право чтения, записи и запуска двум разным пользователям, а еще двум – право чтения и записи, а еще одному – только чтения. а всем остальным – вообще не дать доступ к этому файлу? На такой вопрос нет ответа, если вы работаете в «классической» версии UNIX. Однако, некоторые современные системы UNIX (например, Solaris и FreeBSD) обладают требуемым механизмом. Это – списки управления доступом к файловой системе, access control lists (ACLs).

Списки управления доступом дают более гибкие возможности назначения прав доступа, чем традиционные права доступа UNIX. С помощью ACL можно назначить как права доступа хозяина файла, группы файла и всех остальных, так и отдельные права для указанных пользователей и групп, а также максимальные права доступа по умолчанию, которые разрешено иметь любому пользователю.

Здесь и далее мы будем называть ACL для файлов и каталогов «расширенными правами доступа». Расширенные права доступа к файлам и каталогам устанавливаются командой setfacl.

Они представляют собой ряд полей, разделенных двоеточиями:

entry_type:[uid|gid]:perms

Здесь entry_type – тип расширенного права доступа, uid и gid – идентификаторы пользователи и группы, соответственно, а perms – собственно назначаемые права доступа.

Расширенные права доступа к файлу могут быть такими, как показано в Таблице:

u[ser]::perms	права доступа хозяина файла.
g[roup]::perms	права доступа группы файла
o[ther]:perms	права всех остальных
m[ask]:perms	маска ACL
u[ser]:uid:perms	права доступа указанного пользователя; в качестве uid можно указать и UID, и имя пользователя
g[roup]:gid:perms	права доступа указанной группы; в качестве gid можно указать и GID, и имя группы

Маска задает максимальные права доступа для всех пользователей, за исключением хозяина и групп. Установка маски представляет собой самый быстрый путь изменить фактические (эффективные) права доступа всех пользователей и групп. Например, маска r — — показывает, что пользователи и группы не могут иметь больших прав, чем просто чтение, даже если им назначены права доступа на чтение и запись.

Кроме расширенных прав доступа к файлу, которые могут быть применены и к каталогу, существуют специфические права доступа к каталогу, а именно права доступа по умолчанию. Файлы и подкаталоги, которые будут создаваться в этом каталоге, будут получать такие же расширенные права доступа по умолчанию, которые заданы в расширенных правах доступа к этому каталогу.

Устанавливая права доступа по умолчанию для конкретных пользователей и групп, необходимо также установить права доступа по умолчанию для хозяина и группы файла, а также всех остальных, а также маску ACL (эти четыре обязательных записи указаны выше по тексту – первыми в таблице расширенных прав доступа к файлу).

Расширенные права доступа к каталогам

d[efault]:u[ser]::perms	права хозяина файла по умолчанию
d[efault]:g[roup]::perms	права группы файла по умолчанию
d[efault]:o[ther]:perms	права остальных пользователей по умолчанию
d[efault]:m[ask]:perms	маска ACL по умолчанию
d[efault]:u[ser]:uid:perms	права доступа по умолчанию для указанного пользователя; в качестве uid можно указать и UID, и имя пользователя

d[efault]:g[roup]:gid:perms	права доступа по умолчанию для указанной группы; в качестве gid можно указать и GID, и имя группы

Присвоение расширенных прав доступа осуществляется программой setfacl с ключом –s (set). Без этого ключа расширенные права доступа модифицируются, с ним – устанавливаются в точности такими, как указано в данной команде setfacl.

Например, для установки права на чтение и запись файла project07 для пользователей lena и petr надлежит выполнить команду

setfacl –m user:lena:rw-,user:petr:rw- project07

Ключ -m служит для добавления прав доступа, а не для их замены. Эффективные (т.е. те, которые в самом деле будут иметь место) права доступа пользователей lena и petr определяют не только их персональные права доступа к этому файлу, но и маска, которая показывает права доступа по умолчанию. Из персональных прав и маски выбираются наиболее строгие ограничения, поэтому, если в персональных правах доступа или в маске для файла project07 отсутствует право на запись, ни lena, ни petr не получат реальной возможности изменить файл project07.

Расширенные права доступа не показываются в выводе программы ls, но команда ls –l позволяет установить наличие расширенных права доступа к файлу или каталогу. Если они назначены, то в первой колонке после стандартных прав доступа владельца, группы файла и остальных будет стоять знак «+» («плюс»). Посмотреть, какие именно расширенные права доступа назначены, можно командой

getfacl имя_файла

Более подробную информацию о ключах setfacl и getfacl следует получить из man.

Помните, что назначение прав доступа с помощью chmod может привести к изменению записей ACL, а изменение маски ACL может повлиять на фактические права доступа к объекту для его владельца, группы и всех остальных пользователей.

Завдання до виконання

1. Створіть каталог `lab_2`.
2. Скопіюйте в каталог `lab_2` файл `/bin/cat` під назвою `my_cat`
3. За допомогою файлу `my_cat`, що знаходиться в каталозі `lab_2`, перегляньте уміст файлу `.profile` (Ви знаходитесь у домашньому каталозі).

4. Перегляньте список файлів у каталозі `lab_2`. Потім перегляньте список усіх файлів, включаючи приховані, з повною інформацією про файли. Зверніть увагу на права доступу, власника, дату модифікації файлу, що ви тільки-но скопіювали. Потім перегляньте цю інформацію про оригінальний файл (той, який копіювали) і порівняйте два результати.
5. Змініть права доступу до файлу `my_cat` так, щоб власник міг тільки читати цей файл.
6. Переконайтеся в тім, що ви зробили ці зміни і повторіть п.3.
7. Визначте права на файл `my_cat` таким чином, щоб Ви могли робити з файлом усе, що завгодно, а всі інші — нічого не могли робити.
8. Поверніться в домашній каталог. Змініть права доступу до каталогу `lab_2` так, щоб ви могли його тільки читати.
9. Спробуйте переглянути простий список файлів у цьому каталозі. Спробуйте переглянути список файлів з повною інформацією про них. Спробуйте запустити і видалити файл `my_cat` з цього каталогу.
10. Поясніть отримані результати. Результати виконання п.8 можуть бути різними в різних версіях UNIX, зокрема, Linux і FreeBSD. Прокоментуйте отримані результати у висновках.
11. За допомогою команди `su <user name>`, завантажтесь в систему, користуючись обліковим записом іншого користувача. (Вам потрібно знати пароль цього користувача.) Спробуйте отримати доступ до Вашого каталогу `lab_2`. Перевірте, чи правильно зроблено завдання попереднього пункту. Створіть каталог `lab_2_2`.
12. Знову завантажтесь в систему, користуючись своїм обліковим записом[4]. Спробуйте зробити власником каталогу `lab_2` іншого користувача. Спробуйте зробити себе власником каталогу `lab_2_2`. Поясніть результати.
13. Зайдіть у каталог `lab_2`. Зробіть так, щоб нові створені файли і каталоги діставали права доступу згідно Таблиці (див. наступну сторінку). Створіть новий файл і каталог і переконайтеся в правильності ваших установок.
14. Поверніть собі права читати, писати, та переглядати зміст каталогів.
15. Створіть у каталозі `lab_2` каталог `acl_test` та у ньому файли `file1, file2.` Під час створення `file1` командою `echo` додайте до нього довільний текст.
16. Виведіть `ACL` для `file1`

[4] Ви можете одночасно користуватись різними обліковими записами, використовуючи для цього різні віртуальні консолі в текстовому режимі, або різні вікна терміналів в графічній багатовіконній системі. Віртуальні консолі в текстовому режимі переключаються комбінаціями клавіш ALT+F1 – ALT+F8.

17. Змінить права доступу на `file1` так, щоб тільки власник мав право на читання.
18. Увійдіть до системи під іншим обліковим записом та спробуйте прочитати вміст file1. Що отримаємо? Поверніться до свого облікового запису.
19. За допомогою команди `setfacl` додайте право на читання іншому обраному користувачу для `file1`. Перевірте, що створилось нове `ACL` для `file1`.
20. Увійдіть до системи під іншим обліковим записом та спробуйте прочитати вміст file1. Що отримаємо? Поверніться до свого облікового запису.
21. За допомогою команди `setfacl` встановіть значення маски таким чином щоб дозволити читати зміст `file1` іншому користувачу. Виведіть `ACL` для `file1`
22. Увійдіть до системи під іншим обліковим записом, та спробуйте прочитати вміст `file1`. Ви повинні мати таку змогу.

Таблиця до пункту 13.

варіант	**Права для файлів**	**Права для каталогів**
1	**644**	**754**
2	**664**	**774**
3	**6-4**	**7-5**
4	**62-**	**73-**
5	**644**	**745**
6	**664**	**764**
7	**6-4**	**715**
8	**62-**	**63-**
9	**644**	**744**
10	**664**	**765**

Лабораторна робота №3. Редактор vi

Мета

Оволодіння практичними навичками роботи з редактором vi

Завдання для самостійної підготовки

1. Вивчити:

завантаження редактора і вихід з нього;

завантаження і збереження файлів;

редагування тексту;

виконання команд UNIX.

2. Відповідно до завдання підготувати послідовність команд для його виконання

Довідковий матеріал

Текстові редактори присутні в усіх операційних системах. Вони використовуються для створення і редагування текстових файлів. Текстові файли можуть містити документи, програмні коди і конфігураційні параметри.

Документи можуть містити ознаки форматування (стиль і розмір шрифту, поля, відступи, тощо). Іноді такі ознаки можна розмістити безпосередньо в текстовому документі, використовуючи деякий визначений формат розмітки. Прикладами є формати TEX, вже забутий ChiWriter, а також сучасні HTML і XML. Деякі формати документів відходять від використання текстових файлів, як, наприклад, це було з форматами документів Microsoft Office. Але відбувається повернення до використання текстових файлів при збереженні документів. Таким прикладом є формати Open Document які використовуються у OpenOffice та StarOffice, а також у MS Office 2007. В будь-якому випадку, для редагування документів переважно використовують специфічний клас редакторів – Word Processors.

Для власних потреб операційної системи більше значення мають класичні текстові редактори, важливою ознакою яких є те, що під час редагування вони відображають весь вміст файлу, не приховуючи спеціальні символи чи ключові слова, а під час збереження файлу також не додають від себе додаткових керуючих символів. Це необхідно для можливості роботи з файлами, що містять коди програм або перелік конфігураційних параметрів. Система UNIX дуже інтенсивно використовує текстові файли. Будь-який текстовий файл може розглядатись системою як послідовність команд (так званий пакетний файл, або файл сценарію, див. Лабораторну роботу №6).

Крім того, налаштовування практично всіх програм, як тих, що входять до складу операційної системи, так і прикладних програм, здійснюється за допомогою текстових файлів.

Тому ефективні засоби редагування текстових файлів мають виключне значення для UNIX. Одним із таких засобів є редактор `vi`. Він традиційно входить до поставки будь-якої системи UNIX, і майже напевно присутній в кожній системі (звичайно, адміністратор може його видалити, якщо віддає перевагу іншому редактору, але заради зручності інших користувачів він не повинен так діяти).

Редактор `vi` є єдиним текстовим редактором, який ви можете використовувати для редагування системних файлів без необхідності зміни прав доступу до цих файлів.

Редактор `vi` пропонує могутній набір операцій для редагування тексту, заснований на визначеній множині мнемонічних команд. Більшість команд викликаються натисканням одиночних клавіш і виконують прості функції редагування. На відміну від більшості сучасних редакторів, `vi` взагалі не має системи меню. Не має він і вбудованої системи підказок. Для роботи з цим редактором необхідно запам’ятати команди. Однак для тих, хто знає команди, ефективність роботи в цьому редакторі неперевершена.

Редактор `vi` відкриває "вікно" розміром з екран дисплея, у якому ви можете редагувати ваш файл. При редагуванні забезпечується зворотний візуальний зв'язок (ім'я `vi` – скорочення від слова "visual"). Нижній рядок екрану використовується як інформаційний і командний рядок.

Редактор `vi` тісно співпрацює із строковим редактором `ex`. `vi` і `ex` – це той самий редактор: імена `vi` і `ex` ідентифікують скоріше особливий інтерфейс користувача, чим функціональне розходження. Розходження в інтерфейсі користувача, однак, зовсім разюче. `ex` – могутній строчно-орієнтований редактор, схожий з редактором `ed`. Однак, і в `ex`, і в `ed` візуальне коректування екрана термінала обмежено, а команди вводяться з командного рядка. На відміну від них, `vi` – це екранно-орієнтований редактор, влаштований таким чином, що те, що ви бачите на екрані, точно відповідає вмісту файлу, який ви редагуєте.

Запуск редактора

```
vi [option...] [command...] [filename...]
view [option...] [command...] [filename...]
```

Команда `view` аналогічна `vi`, за винятком того, що автоматично встановлюється опція “тільки зчитування” (`-R`). При використанні `view` файл змінюватися не може.

Крім вже згаданої опції `-R`, у командному рядку `vi` припустимі інші опції, з яких слід відзначити `-r` (не плутати великі і маленькі літери!). Ця опція використовується при відновленні, коли мало місце ушкодження редактора чи всієї системи. `vi -r` **<***filename***>** відшукує останню збережену версію зазначеного файлу. Якщо файл не визначений, то ця опція виводить список збережених файлів.

`vi` не виконує ніяких операцій редагування безпосередньо над зазначеним вами файлом. Замість цього він працює з копією вашого файлу, що знаходиться в буфері редагування. Коли ви активізуєте `vi` з одним аргументом – ім'ям файлу, цей файл копіюється в тимчасовий буфер редагування. Редактор запам'ятовує ім'я файлу, визначеного при виклику, таким чином, що пізніше він може скопіювати вміст буфера редагування назад у зазначений файл. Вміст заданого файлу не змінюється доти, поки всі зміни не будуть скопійовані назад у первісний файл.

Приклади команд, що можна застосовувати для входу в редактор `vi`:

`vi`	Редагує порожній буфер редагування
`vi <`*file***`>`**	Відкриває для редагування зазначений файл
`vi +123 <`*file***`>`**	Відкриває для редагування зазначений файл і переходить в ньому на рядок з номером 123
`vi +/<`*word***`> <`***file***`>`**	Відкриває для редагування зазначений файл і шукає перше входження слова "*word*"

Режими роботи

У `vi` існує три окремих режими:

Командний режим: у цьому режимі сигнал із клавіатури інтерпретується як команда редагування.

Режим вставки: перейти в цей режим можна набором будь-яких команд вставки, приєднання, відкриття, підстановки, зміни чи заміщення, що існують в `vi`. У цьому режимі символи, набрані на клавіатурі, вставляються в буфер редагування.

Режим переключення в '`ex`': У `vi` команди – це одиночні клавіші. У `ex` командами є рядки тексту, завершені натисканням клавіші **`RETURN`**. `vi` має спеціальну команду "`escape`", що дозволяє перейти до більшості строчно-орієнтованих команд редактора `ex`. Для використання режиму переключення в `ex` наберіть символ '**`:`**'. Цей символ відобразиться в командному рядку як покажчик на наступну команду редактора `ex`. Більшість команд обробки файлу виконується в режимі переключення в `ex`

(наприклад, команди читання з файлу і запису з буфера редагування назад у файл).

Команди переключення між режимами

ESC Переводить з режиму вставки в командний режим. Якщо не впевнені, в якому режимі знаходитесь – краще зайвий раз натиснути **ESC**

Команди вставки: переводять з командного режиму в режим вставки. При застосуванні команд вставки текст не знищується.

I
i Команди '**i**' та '**I**' – "insert". Команда '**i**': починає вставляти текст перед символом, що розташований під курсором. Для вставки нового рядка натисніть RETURN. Команда '**I**' починає вставляти текст із початку поточного рядка.

A
a Команди '**a**' та '**A**' – "append". Команда '**a**' працює точно так, як команда '**i**', тільки вставка тексту починається після курсору, а не перед ним. Це є однією з можливостей додавання тексту в кінець рядка. Команда '**A**' починає вставку тексту наприкінці поточного рядка.

O
o Команди '**o**' та '**O**' – "open". Відкривають новий рядок і вставляють текст. Команда '**o**' відкриває новий рядок під поточним рядком, команда '**O**' відкриває новий рядок над поточним рядком. Після того, як новий рядок відкритий, обидві команди працюють аналогічно команді '**I**'.

Команди переходу в режим редактора 'ex'.

Q Переводить з командного режиму `vi` в режим редактора `ex`. Після виконання цієї команди буде продовжуватись редагування того самого файлу. Ви можете повернутися в режим редактора `vi`, набравши '`vi`' у редакторі `ex`.

: Переводить в режим тимчасового переключення в '`ex`' для виконання однієї команди редактора `ex`. В інформаційному рядку з'являється двокрапка '**:**' як підказка-вказівка для введення ex-команди. Ви можете ввести ex-команду, закінчити її символами **RETURN** чи **ESC**, після чого цю команду буде виконано. Потім вам запропонують натиснути клавішу **RETURN** для повернення в командний режим `vi`.

Команди виходу

Існує кілька шляхів виходу з редактора `vi`. Якщо ви знаходитесь в режимі вставки, то перший крок – перейти в командний режим (клавіша **ESC**). Далі можна використовувати такі команди:

ZZ Здійснює вихід з командного режиму `vi`. Вміст буфера редагування записується у файл, що редагувався, тільки за умови, що були зроблені які-небудь зміни. При цьому ви повертаєтесь в ту командну оболонку, з якої був запущений редактор `vi`.

:x Діє так само, як і ZZ. Фактично означає перехід в режим '`ex`' з наступним виконанням команди **x**.

:q Намагається здійснити вихід з редактора `vi` без запису з буфера редагування у файл. Однак, якщо з моменту останньої команди '**w**' до буферу редагування вносились зміни, то `vi` видає попереджуюче повідомлення, і вихід з редактора не здійснюється. `vi` також видає діагностику, якщо в списку аргументів залишилися імена ще невідредагованих файлів. Зазвичай ви хочете зберегти усі ваші зміни; для цього вам належить набрати команду '**:w**'.

:q! На відміну від попередньої, ця команда скасовує сеанс редагування. Знак оклику вказує редакторові `vi` на необхідність безумовного виходу. Вміст буфера редагування не зберігається.

:wq Переходить в режим `'ex'`, зберігає зміни буфера редагування в поточний файл (команда **w**), після чого здійснює вихід з редактора (команда **q**).

:wq! Скасовує перевірку, що звичайно виконується перед командою '**w**'. Наприклад, якщо ви володієте файлом, але не маєте дозволу на запис у нього, команда '**:wq!**' дозволить вам змінити вміст.

Увага! Для виходу з редактора `vi` (як і з інших програм UNIX) не можна використовувати комбінацію клавіш **CTRL+Z**, оскільки ця комбінація не завершує програму, а лише призупиняє її (див. Лабораторну роботу 5).

У режимі вставки можуть використовуватися такі символи:

BKSP У поточному рядку повертає курсор назад на один символ. Останній символ, набраний перед **BKSP**, видаляється з вхідного буфера, але залишається на екрані дисплея.

Ctrl-V Відміняє спеціальне значення наступного набраного символу. Використовуйте **Ctrl-V** для вставки керуючих символів.

Клавіші курсору	Можуть діяти нормально (переміщати курсор) або ні (вводити якісь незрозумілі символи) в залежності від версії редактора і налаштовувань терміналу

Переважна більшість команд, що діють в командному режимі `vi`, в режимі вставки діяти не можуть, оскільки викликають вставку в буфер редагування набраних літер.

Команди редактора `vi`, що діють в командному режимі

Далі описані лише основні, найнеобхідніші команди. Більшість команд можуть використовувати лічильник, що стоїть перед ними і показує число повторів команди. Цей параметр у наступних описах команд не заданий, але він мається на увазі, якщо не скасований яким-небудь аргументом, що стоїть перед ним. Коли редактор `vi` одержує команду неправильного формату, він сигналізує про це.

Переміщення курсору

Команди керування курсором дозволяють вам переміщати курсор по файлу. Вони дуже важливі тому, що деякі команди з інших груп (див. далі) використовують команди переміщення курсору як аргумент, що дозволяє визначити блок тексту. Поточна позиція курсору розглядається як початок блоку, а позиція курсору, яку він займе в разі виконання команди переміщення, – як кінець блоку (або навпаки, якщо переміщення здійснюється не вперед, а назад). Виконання таких команд відбувається лише після введення команди переміщення курсору.

l **SPACEBAR** →	Переміщає курсор на один символ вперед. Якщо заданий лічильник, то переміщає вперед на зазначене число символів. Ви не можете перемістити за межу кінця рядка.
h **BKSP** ←	Переміщає курсор на один символ назад. Якщо заданий лічильник, то переміщає назад на зазначене число символів. Ви не можете перемістити за межу початку рядка.
+ **RETURN**	Переміщає курсор вниз на початок наступного рядка.
J **Ctrl-N** ⇓	Переміщає курсор вниз на один рядок, залишаючи його в тому ж стовпчику.
–	Переміщає курсор на початок попереднього рядка. Якщо заданий лічильник, то курсор переміщається вгору на зазначене число рядків.

K **Ctrl-P** ⇑	Переміщає курсор на один рядок вгору, залишаючи його в тому ж стовпчику. Якщо заданий лічильник, то курсор переміщається на зазначене число рядків.
O	Переміщає курсор на перший символ поточного рядка.
^	Переміщає курсор на перший символ рядка, такий, що не є міткою табуляції чи пробілом. Це дуже зручно при редагуванні тексту з форматованими відступами (наприклад, тексту програми).
$	Переміщає курсор у кінець поточного рядка. Зверніть увагу, що курсор знаходиться над останнім символом у рядку.
w **b** **e**	Переміщає курсор вперед на початок наступного слова (**w**), назад на початок поточного слова (**b**), або на останній символ поточного слова (**e**), де слово визначене як рядок буквено-числових символів, розділених пунктуацією, мітками табуляції, символами нового рядка чи пробілами.
%	Переміщає курсор на узгоджуючий роздільник, яким можуть бути кругла, операторна чи фігурна дужки. Це дуже зручно при узгодженні пар вкладених круглих, операторних і фігурних дужок (наприклад, при редагуванні тексту програми).

Команди екрана

Команди екрана не є командами керування переміщенням курсору, і не можуть використовуватися як роздільники об'єктів тексту. Однак команди екрана здійснюють переміщення тексту і дуже зручні для сторінкової організації чи "прокручування" інформації з файлу на екрані дисплея.

Ctrl-U **Ctrl-D**	"Прокручує" екран на половину вікна нагору (**Ctrl-U**) чи вниз (**Ctrl-D**).
Ctrl-F **Ctrl-B**	Посторінково перегортає екран уперед та назад. Якщо можливо, то між сторінками зберігаються два нерозривні рядки. Заданий попереду лічильник указує число сторінок, на яке треба пересунути вперед чи назад.
Ctrl-G	Показує стан редактора **vi** в інформаційному рядку: ім'я файлу, що редагується, чи був він змінений, номер поточного рядка, число рядків у файлі і відсоток файлу (у рядках), що передує місцеві перебування курсору.

Ctrl-R
Ctrl-L Перемальовує вміст екрана. Використовуйте цю команду для витирання будь-яких системних повідомлень, що можуть накладатися на інформацію, що міститься на вашому екрані. Зверніть увагу, що системні повідомлення не впливають на файл, що редагується вами.

Видалення тексту

Найбільш універсальна команда видалення тексту використовує як оператор виконання клавішу '**d**'. Цей оператор видаляє текстові об'єкти, обмежені поточною позицією курсору і командою переміщення курсору. Видалення фактично відбувається після завершення вводу команди переміщення курсору. Вилучений текст завжди продовжує зберігатися в буфері.

x Видаляє символ, що знаходиться під курсором.

<*n*>x Видаляється <*n*> символів вправо від символу, що стоїть під курсором.

X Видаляє символ, розташований перед курсором.

<*n*>X Видаляється <*n*> символів у зворотному напрямку, починаючи із символу, що стоїть перед курсором.

d <*mov*> Видаляє текстовий об'єкт. Команда '**d**' бере як аргумент команду переміщення курсору **<*mov*>**. Якщо **<*mov*>** задає переміщення в межах рядка, то здійснюється видалення від курсору до кінця текстового об'єкта, обмеженого аргументом. Видалення в прямому напрямку (вперед) видаляє символ, розташований під курсором; видалення в зворотному напрямку (назад) не виконується. Якщо **<*mov*>** задає переміщення на інший рядок, то видалення здійснюється з поточного рядка, включаючи його самого, до текстового об'єкта, обмеженого аргументом.

Наприклад:

dl знищує символ, що стоїть під курсором;

d5l знищує 5 символів, починаючи з того, що стоїть під курсором;

d3J знищує 3 рядки, починаючи від поточного рядка вниз

dd Видаляє цілі рядки.

D Видаляє усі символи від позиції курсору, включаючи її, до кінця поточного рядка.

Переміщення тексту

Переміщення тексту здійснюється командами видалення, копіювання і вставки текстових блоків. Для цього зарезервовано 9 буферів видалення (позначених цифрами від 1 до 9), 26 буферів для копіювання (позначених літерами від a до z) і один “безіменний” буфер, який використовується за замовчуванням. Якщо необхідно вказати певний буфер, перед будь-якою командою копіювання або вставки вводяться подвійні лапки (**"**), а за ними – позначення буфера. Для команд вставки (*put*) позначення буфера може бути цифрою від 1 до 9 або літерою від a до z. Для команд копіювання (*yank*) позначення буфера може бути літерою від `a` до `z` або від `A` до `Z`. Якщо ви копіюєте текст у буфер з ім'ям '**A**' замість '**a**', то текст додається до вмісту буфера '**a**' (так само для будь-якого іншого буфера). При видаленні вилучений текст автоматично заноситься в стек буферів, пронумерованих від 1 до 9, тобто щойно видалений блок поміщається в буфер 1, блок з буферу 1 переміщається у буфер 2 і так далі. Щойно вилучений текст також розміщається в “безіменному” буфері.

Таким чином, в буферах 1-9 знаходяться блоки тексту, які було вилучено, в буферах a-z – блоки тексту, які було скопійовано, а в безіменному буфері – блок тексту, який було вилучено під час останньої операції або скопійовано без зазначення буфера.

Наприклад, команда **"4p** поміщає вміст буфера видалення з номером 4 у ваш буфер редагування під поточним рядком. Якщо ім'я буфера-джерела не зазначено, то текст вставляється з "безіменного" буфера. Таким чином, дуже просто видалити текст, потім пересунути курсор у те місце, куди ви хочете вставити вилучений текст, і після цього вставити текст у нове місце за допомогою команди '**p**' або '**P**'.

Пойменовані буфери найбільш зручні для збереження сукупності декількох частин тексту, які ви хочете мати постійно напоготові для пізнішого доступу до цих текстів, або їхнього переміщення і перерозміщення. Наприклад, для копіювання рядка з файлу в буфер '**a**' наберіть **"ayy**. Для того, щоб помістити цей текст назад у файл, наберіть **"ap**.

Слід зазначити, що вміст пойменованих буферів не руйнується при переключенні файлів. Тому ви можете видалити чи скопіювати текст у буфер, відкрити новий файл і потім виконати команду '**p**'. Вміст буферів губиться при виході з редактора, тому будьте обережні.

["<*buf***>]p** Вставляє текст із буфера **<***buf***>** в буфер редагування під поточним рядком чи після курсору, в залежності від того, містить буфер повний чи рядок ні.

["<*buf*>]p	Вставляє текст із буфера **<*buf*>** в буфер редагування або над поточним рядком, або перед курсором, у залежності від того, містить буфер повний рядок чи ні.
["<*buf*>]y <*mov*>	Копіює текст із буфера редагування в буфер **<*buf*>**. Аргумент **<*mov*>** обмежує об'єкт, що копіюється (аналогічно команді видалення '**d**' – див. вище.
["<*buf*>]yy **["<*buf*>]Y**	Копіюють один рядок, чи, якщо заданий лічильник, кілька рядків.

Відміна/повтор операцій

u	Скасовує результат останньої команди вставки чи видалення. Після виконання команди вставки команда '**u**' видаляє вставлений текст, а після команди видалення – вставляє текст назад.
U	Відновлює поточний рядок у його первісному стані, перш ніж він був відредагований, незалежно від того, скільки операцій редагування ви здійснили з того моменту, як ви перейшли на нього.
.	Повторює останню команду вставки чи видалення.

Пошук

Команди пошуку здійснюють пошук тексту, що відповідає заданому регулярному виразу і знаходиться в буфері редагування, у прямому чи зворотному напрямку.

/[<*pat*>] ↲ **?[<*pat*>]** ↲	Здійснює пошук тексту, що відповідає шаблону **<*pat*>**, у прямому (**/**) чи зворотному (**?**) напрямку. Рядок є дійсним регулярним виразом. Якщо шаблон **<*pat*>** не заданий, то для пошуку використовується останнє значення **<*pat*>**, що використовувалось раніше. (↲ – натискання клавіші "**RETURN**")
n **N**	Повторює останню команду пошуку. Команда '**n**' повторює пошук у тому ж напрямку, що й остання команда пошуку, команда '**N**' – у протилежному напрямку.
f <*ch*> **F <*ch*>**	Знаходить символ **<*ch*>** у поточному рядку. Команда '**f**' виконує пошук у прямому напрямку, команда '**F**' – у зворотному.
t <*ch*> **T <*ch*>**	Встановлює курсор над символом **<*ch*>**.
; **,**	Знак "**;**" повторює пошук останнього символу. Знак "**,**" змінює напрямок пошуку на протилежний.

***Команди редактора* ex**

Введення двокрапки (**:**) при перебуванні в командному режимі видає підказку-запрошення (**:**) у рядку стану. Це підказка для введення команди, доступної в строковому редакторі **ex**.

В основному, **ex**-команди дозволяють вам записувати до і читати з файлів, переключатись в shell чи змінювати файли, що редагуються. Більшість цих команд виконує дії, що впливають на "поточний" за замовчуванням файл. Поточним звичайно вважається файл, що ви вибрали при запуску редактора **vi**, хоча поточний файл може бути змінений командою **file** (**f**) чи командою **next** (**n**).

Як імена більшості ex-команд використовуються англійські слова, а доступною для використання абревіатурою є початкові букви цих слів. В подальших описах згадуються тільки абревіатури як найчастіше використовувана форма запису команди.

Загальна форма команди **ex** досить складна. Для всіх ex-команд використовується наступний формат:

[address][command][!][parameters][count][flags]

Усі частини необов'язкові, в залежності від команди та її опцій.

Більшість команд використовує адреси, що стоять попереду і визначають рядки, над якими необхідно виконувати дії. Ряд команд також може містити наступний за ними лічильник **count**, що показує число рядків, що захоплюються при виклику команди. Числа **count** при необхідності округляються вбік меншого значення. Таким чином, команда '**10p**' показує десятий рядок у буфері (тобто, 10 – це адреса), а команда '**move 5**' переміщає поточний рядок за рядок з номером 5 (тобто, 5 – це лічильник).

Деякі команди також використовують параметри, розміщені після імені команди. Ряд команд має варіанти. Інша форма команди викликається встановленням знаку оклику (**!**) безпосередньо за ім'ям команди.

Далі команди **ex** розглянуті в мінімальному обсязі: лише ті команди, які часто викликаються з **vi** і суттєво доповнюють можливості останнього. Крім того, розгляд обмежено командами, що стосуються всього файлу.

Команди запису

Команди запису дозволяють вам переписувати весь ваш буфер редагування або його частину в поточний або будь-який інший файл.

w [<*file*>] Записує зроблені зміни у файл **<*file*>**, показуючи кількість записаних рядків і символів. Якщо параметр **<*file*>** не заданий, буфер записується в поточний файл. Якщо ім'я **<*file*>** визначене, то буфер редагування записується в цей файл. Редактор здійснює запис у файл тільки якщо це поточний файл і він редагується, або якщо файл не існує (в останньому випадку файл створюється). В інших випадках для запису ви повинні задати команду у зміненій формі – **'w!'**.

w! <*file*> Скасовує перевірку звичайної команди **write** і здійснює запис у будь-який файл, що дозволений з боку системи.

w >> <*file*> Додає вміст буфера до кінця існуючого файлу. Попередній вміст файлу не руйнується.

Команди зміни поточного файлу редагування

Для редагування файлу, іншого від того, що редагується в поточний момент, ви можете використовувати один з варіантів команди '**e**'.

e <*file*> Використовується для початку редагування нового файлу. Редактор спочатку перевіряє, чи був модифікований буфер з моменту використання останньої команди '**w**'. Якщо це було зроблено, то видається попередження, і команда переривається. Якщо ні – команда видаляє вміст буфера редагування, робить файл **<*file*>** поточним і висвітлює нове ім'я файлу. Після перевірки, що цей файл дійсний (тобто не є бінарним файлом, каталогом або пристроєм), редактор читає файл у свій буфер. Якщо читання файлу виконане без помилок, у рядку стану з'являється число прочитаних рядків і символів. Поточним рядком спочатку вважається перший рядок файлу.

e! <*file*> Такий виклик скасовує повідомлення про модифікації, що були зроблені і не записані з буфера редагування, викликаючи, тим самим, скасування всіх змін, що були виконані перед редагуванням нового файлу.

e +n <*file*> Змушує редактор почати редагування не з першого, а з n-го рядка. Аргумент **n** може бути також командою редактора, що не містить пробілів, наприклад, **+/pattern**.

Команди читання

Команди читання дозволяють вам читати текст у ваш буфер редагування з будь-якого місця (тобто, дозволяє вставити певний текст у визначене місце). Текст, що ви читаєте, повинен складатися, принаймні, з одного рядка, чи бути або файлом, або вхідною інформацією команди.

r [<*file*>] Поміщає копію тексту з заданого файлу **<*file*>** в буфер редагування після поточного рядка. Якщо файл не заданий, то використовується поточне ім'я файлу. Якщо буфер редагування порожній, то це трактується як команда '**e**'. Коли команда '**r**' завершується успішно, то видається статистика, подібна до тієї, яка супроводжує виконання команди '**e**'. Після команди '**r**' поточним вважається останній прочитаний рядок.

r! <*cmd*> Читає вихідну інформацію команди **<*cmd*>** у буфер після визначеного рядка.

Команди закінчення роботи

Існує кілька шляхів виходу з редактора **vi**. Деякі переривають сеанс редагування, інші записують вміст буфера редагування перед виходом, треті попереджають вас, якщо ви вирішили вийти без записування буфера редагування. Ці шляхи були розглянуті вище. Зазначимо додатково лише те, що **ex**-команди **wq**, **wq!** і **x** дозволяють зробити запис у файл, відмінний від поточного (наприклад, **wq <*file*>**).

Команди переключення в shell

Часто виникає необхідність тимчасово вийти з редактора і виконати звичайні команди системи UNIX. Також може виникнути необхідність змінити робочий каталог, щоби редагування не впливало на цілісність іншого робочого каталогу. Нижче описані необхідні для цього операції:

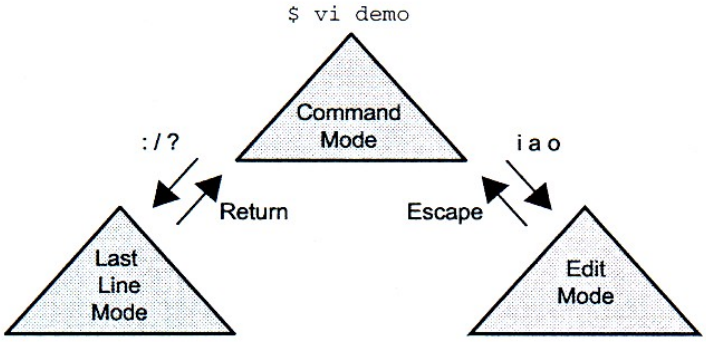
cd <*dir*> Зазначений каталог **<*dir*>** стає вашим поточним каталогом. Якщо каталог не визначений, то в якості імені цільового каталогу використовується домашній каталог користувача.

Sh Запускається новий shell, у якому ви можете виконати будь-яку кількість команд. Для повернення в **vi** натисніть **Ctrl-D**.

!<*command*> Залишок рядка після знака “ **!**” передається в shell як команда для виконання. В текст команди **<*command*>** замість символів '**%**' і '**#**' підставляються імена поточного файлу й останнього файлу, що редагувався, а символ '**!**' заміняється на текст попередньої команди. Ці підстановки повторюються на екрані, але запам’ятовується рядок цієї команди без підстановок.

Якщо з моменту останньої зміни в буфері редагування запис файлу не здійснювався, то перед виконанням команди видається попередження. Коли команда виконається, висвітлюється знак "**!**".

Якщо ви використовуєте оболонку C-shell і встановлюєте змінну **`prompt`** для виводу підказки-вказівки **`prompt>`**, необхідної для роботи з інтерактивними shell, то при використанні вами вищенаведених команд **`prompt`** розглядається як ім'я файлу. Це може привести до виникнення несподіваних ситуацій. Щоб уникнути їх, використовуйте значення **`prompt`**, призначуване за замовчуванням, яке визначено у файлі **`/usr/lib/mkuser/mkuser.cshrc`**.



Search Functions

`/exp`	Go forward to exp
`?exp`	Go backward to exp

Move and Insert Text

`:3,8d`	Delete line 3-8
`:4,9m 12`	Move lines 4-9 to 12
`:2,5t 13`	Copy lines 2-5 to 13
`:5,9w file`	Write lines 5-9 to file

Save Files and Exit

`:w`	Write to disk
`:w newfile`	Write to newfile
`:w! file`	Write absolutely
`:wq`	Write and quit
`:q`	Quit editor
`:q!`	Quit and discard
`:e!`	Re-edit current file, Discard buffer

Control Edit Session

`:set nu`	Display line number
`:set nonu`	Turn off line number
`:set all`	Show all settings
`:set list`	Display invisible Characters
`:set wm=5`	Wrap lines 5 spaces From right margin

Screen/Line Movement

`j`	Move cursor down
`k`	Move cursor up
`h`	Move cursor left
`l`	Move cursor right
`0`	Go to line start (zero)
`$`	Go to line end
`G`	Go to last file line

Word Movement

`w`	Go forward 1 word
`b`	Go backward 1 word

Search Functions

`n`	Repeat previous search
`N`	Reverse previous search

Delete Text

`x`	Delete 1 character
`dw`	Delete 1 word
`dd`	Delete 1 line
`D`	Delete to end of line
`d0`	Delete to start of line
`dG`	Delete to end of file

Cancel Edit Function

`u`	Undo last change
`.`	Do last change again

Copy and Insert Text

`Y`	Yank a copy
`5Y`	Yank a copy of 5 lines
`p`	Put below cursor
`P`	Put above cursor

Add/Append Text

`a`	Append after cursor
`A`	Append at line end
`i`	Insert before cursor
`5i`	Insert text 5 times
`I`	Insert at beginning of line

Add New Lines

`o`	Open a line below cursor
`O`	Open a line above cursor

Search Functions

`n`	Repeat previous search
`N`	Reverse previous search

Change Text

`cw`	Change a word
`3cw`	Change 3 words
`C`	Change line
`r`	Replace one character
`R`	Replace/type over a line

Quick Reference Chart for the **vi** editor

Завдання до виконання

23. Завантажтеся в систему під Вашим користувацьким ім'ям.
24. Створіть новий текстовий файл `text` за допомогою редактора `vi`. Наберіть два абзаци тексту. Текст повинен містити Ваше прізвище (наприклад, у вигляді підпису). Запишіть файли під іменами `text` і `text1`, вийдіть із редактора.
25. Установіть на файл `text1` права доступу так, щоб Ви могли тільки читати цей файл, але не модифікувати його.
26. Завантажте файл `text` у редактор, скопіюйте перші 2 рядка тексту в буфер і вставте їх у кінець тексту.
27. Запишіть файл під тим же ім'ям.
28. Не виходячи з редактора, завантажте файл `text1` і, попередньо відкривши новий `shell` і змінивши права доступу на файл, запишіть файл, не виходячи з редактора.
29. Користуючись пойменованими буферами, перенесіть 3 рядки тексту з першого файлу в другий. Збережіть зміни. Вийдіть з редактора.
30. Відкрийте у редакторі файл **`text`** і в кінець його додайте уміст файлу **`text1`**.
31. Запишіть отриманий файл як **`text2`** і, не виходячи з редактора, видаліть файли **`text`** і **`text1`**.

Лабораторна робота №4. Командна оболонка shell, стандартні потоки вводу/виводу, фільтри і конвеєри

Мета

Оволодіння практичними навичками перенаправлення стандартних потоків, роботи з фільтрами і організації конвеєрів

Завдання для самостійної підготовки

1. Вивчити:
   - командні оболонки, їх запуск, конфігураційні файли;
   - стандартні потоки і їх перенаправлення;
   - організацію конвеєрів;
   - організацію фільтрів і команди, використовувані як фільтри.

2. Ознайомитись з такими командами UNIX:
   `tee, find, cut, date, grep, sort`
   Звернути увагу на метасимволи **`*`**, **`?`**, **`/`**, **`[…]`**, **`$`** і на правила інтерпретації їх при використанні одинарних та подвійних лапок **`'...'`** та **`"..."`**. Розібратись з використанням в командах операторів перенаправлення потоків і організації конвеєрів **">"**, **"<"**, **"|"** і використанням псевдопристрою **`/dev/null`**.

3. Відповідно до завдання підготувати послідовність команд для його виконання

Довідковий матеріал

У цій роботі ми розглянемо деякі можливості командних оболонок, які дозволяють користувачам гнучко формувати завдання для виконання операційною системою.

Кожна командна оболонка в процесі свого запуску робить налаштовування системного оточення (виконує ініціалізацію системних змінних та змінних командної оболонки), для чого використовує визначені файли. Їх щонайменше два – глобальний і локальний (деякі оболонки можуть використовувати більшу кількість конфігураційних файлів). Глобальні конфігураційні файли для всіх оболонок знаходяться в каталозі `/etc`. Локальні конфігураційні файли для всіх оболонок знаходяться в домашньому каталозі користувача. Імена конфігураційних файлів, як правило, закінчуються на '`rc`'. Локальні файли роблять "прихованими", щоби вони не заважали користувачу в його повсякденній роботі (приховані файли мають

ім'я, що починається з символу '`.`', команда `ls` без параметрів їх не показує). Приклади конфігураційних файлів: для `sh` – `/etc/profile` і `~/.profile`, для `csh` – `/etc/cshrc` і `~/.cshrc`, для `tcsh` – `/etc/cshrc` і `~/.cshrc`, а також `/etc/tcshrc` і `~/.tcshrc` (двох останніх може і не бути).

Конфігураційні файли є звичайними командними файлами (докладніше про командні файли див. Лабораторну роботу №6).

Часто в якості параметру деякої команди нам треба вказати не один файл, а кілька файлів, назви яких мають певні спільні риси. В таких випадках використовують так звані маски пошуку. Спеціальний символ '**?**' в масці означає один будь-який символ, а спеціальний символ '*****' – будь-яку послідовність символів. Наприклад, команда `ls /bin/????` виведе на екран всі файли з каталогу `/bin`, імена яких складаються з чотирьох символів, а команда `ls /etc/d*` виведе на екран всі файли з каталогу `/etc`, імена яких починаються з літери **d**. Також можна задати список символів, наприклад, маска `[abc]???` задає ім'я з чотирьох літер, перша з яких – **a**, **b** чи **c**.

Для пошуку файлів за певними ознаками можна використовувати команду `find`. Перший параметр цієї команди (обов'язковий) – це каталог, з якого починається пошук (наприклад, **/** – кореневий каталог), далі – параметр, що задає ознаку пошуку (наприклад, `-name` – пошук файлів, імена яких відповідають заданій масці, `-atime` – пошук файлів, дата модифікації яких відповідає заданій умові), далі іде власне маска пошуку, а далі – дія. Найтиповіша дія – `-print`, вивід результатів пошуку на екран. Якщо цей параметр не вказати, пошук відбуватись буде, а от результатів його видно не буде.

Наприклад:

```
find / -name "*.c" -print
```

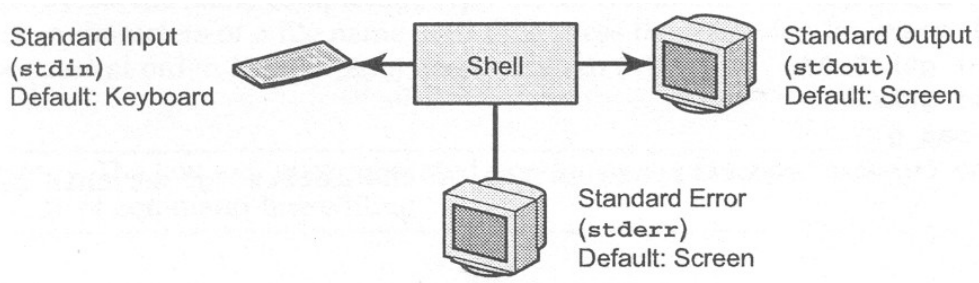
виведе на екран список всіх файлів, імена яких мають розширення .c, тобто вихідних кодів на мові програмування C[5].

Ще одна можливість оболонок – перенаправлення потоків вводу-виводу. Як правило, більшість команд (утиліт) приймає інформацію з клавіатури, або з файлу, якщо його вказано як параметр, і виводить результати на екран. Однак, фактично вони працюють із так званими стандартними потоками вводу і виводу, які пов'язані з певними файлами. Файл в системі UNIX розглядається як потік байт. Оскільки пристрої в UNIX розглядаються як файли, а операції вводу і виводу – як читання і запис у

[5] Якщо ваш UNIX поставляється з вихідними кодами (Linux, FreeBSD), то таких файлів буде безліч!

відповідні файли, це дозволяє легко переводити вхідний і вихідний потоки з файлів на пристрої чи навпаки.

Стандартний ввід за замовчанням зчитується з клавіатури. Якщо в якості параметру вказано ім'я файлу, то замість стандартного вводу відповідна утиліта буде організовувати вхідний потік з указаного файлу (але так діють не всі команди!) Вихідних потоків є два – стандартний потік виводу (за замовчанням в сучасних системах – на екран монітору), і потік повідомлень про помилки (за замовчанням – туди ж).



Оболонка дає змогу перенаправити потоки у заданий файл. Символ '**<**' перенаправляє вхідний потік. Після цього символу очікується ім'я файлу або пристрою, з якого буде братись вхідний потік.

Наприклад, команда `cat` без параметрів очікує ввід з клавіатури, і кожний рядок передає на екран монітора. Команда `cat my_file` замість вводу з клавіатури виведе на екран вміст файлу `my_file`, якщо такий існує, і повідомлення про помилку, якщо такого не існує.

Команда `cat < my_file` на перший погляд буде робити те ж саме, але з точки зору операційної системи і самої утиліти `cat` все буде відбуватись по-іншому. В попередньому випадку командна оболонка запускала утиліту `cat` і передавала їй параметр командного рядка `my_file`, а сама утиліта `cat` вже інтерпретувала цей параметр як ім'я вхідного файлу. В останньому випадку командна оболонка запускала утиліту `cat` без параметрів, але передавала їй вміст файлу `my_file` в якості вхідного потоку.

Команда `cat > my_file` перенаправляє вихідний потік. Оскільки вхідний потік не перенаправляється, ввід буде очікуватись з клавіатури. Тому ця команда створить файл `my_file`, якщо він не існує, знищить вміст файлу `my_file`, якщо він існує, і буде записувати в цей файл все, що буде введено з клавіатури, аж поки не поступить символ кінця файлу EOF (Ctrl+D). Існує також можливість дописати інформацію в кінець файлу, не знищуючи його вмісту. Така команда буде мати вигляд `cat >> my_file`

Команда

```
cat < my_file1 > my_file2
```

перенаправляє як вхідний, так і вихідний потік. Якщо файл `my_file1` існує, то його вміст буде записано в файл `my_file2`. Якщо не існує, то повідомлення про помилку буде виведено на екран. Потік повідомлень про помилки в оболонці `sh` перенаправляється окремо, він позначається **`2>`**. Наприклад, команда

```
cat < my_file1 > my_file2 2> my_file3
```

у разі, якщо файл `my_file1` існує, запише його вміст в файл `my_file2`, а якщо не існує, то запише повідомлення про помилку в файл `my_file3`. Важливо наголосити, що оболонки `csh` і `tcsh` не мають засобів безпосередньо перенаправляти потік повідомлень про помилки. Тим не менше, засобами цих оболонок також можна організувати окреме перенаправлення потоків завдяки хитрим прийомам програмування. Інформацію про це можна знайти в літературі.

Дуже часто потік помилок намагаються взагалі "загасити". Для того, щоби знищити якийсь потік, існує спеціальний пристрій `/dev/null`. Все, що в нього направляється, зникає безслідно.

Перенаправлення вводу-виводу широко використовується у двох випадках. Перший – це запуск утиліт у фоновому режимі. Щоби їхня робота не заважала роботі користувача з терміналом, слід так перенаправити потоки вводу-виводу, щоби вони працювали лише з файлами. Другий – це використання спеціальних команд-утиліт, які призначені саме для того, щоби прийняти певну інформацію з одного файлу, обробити її, а результат записати у другий файл. Такі утиліти називаються фільтрами. Утиліта `cat`, варіанти використання якої з перенаправленням потоків було розглянуто вище – це простіший фільтр. Він практично не обробляє інформацію, лише може зчіплювати кілька файлів в один. Інші корисні фільтри: `cut, grep, sort`.

Утиліта `cut` переглядає вхідний файл, і виділяє з кожного його рядка інформацію за ознаками розміщення в певних колонках або полях. Наприклад, рядки файлу `/etc/passwd` розділяються на поля за допомогою символу '**`:`**'. Перше поле – *login*, п'яте поле – інформація про користувача. Якщо ми хочемо надрукувати лише цю інформацію, ми можемо дати команду:

```
cut -d: -f1,5 < /etc/passwd
```

Ключ **`-d`** задає символ-роздільник полів (у цьому випадку – '**`:`**'), ключ **`-f`** – список полів, що треба роздрукувати (у цьому випадку – 1 і 5).

Утиліта `grep` виводить лише ті рядки, в яких зустрічається заданий рядок пошуку. Утиліта `sort` виконує сортування вхідного потоку,

наприклад, за абеткою. Докладніше про ці та інші фільтри вам слід дізнатися з довідкової системи `man`.

Існує можливість перенаправити вихідний потік однієї утиліти безпосередньо у вхідний потік іншої, без використання тимчасових файлів. Це так звані конвеєри (pipe). В системі UNIX всі утиліти, що поєднані в конвеєр, запускаються паралельно і обробляють інформацію по мірі її надходження. Конвеєр утворюється за допомогою символу '**|**' таким чином:

```
util1 | util2 | util3
```

При утворені конвеєра окремо перенаправляти вхідні й вихідні потоки на проміжних стадіях не можна – це буде або сприйнято як синтаксична помилка, або результат може бути непередбачуваним.

Приклад конвеєру:

```
ps –al | grep root | more
```

Команда `ps` з ключами `–al` направить у вихідний потік список всіх процесів у системі, `grep root` вибере з них лише ті, які виконуються від імені *root*'а, `more` забезпечить вивід їх на екран посторінково. Інший приклад:

```
cat /etc/passwd | cut –d: –f1,5 | more
```

Ця команда зробить те ж саме, що й приклад з командою **`cut`**, що розглядався раніше, але вивід на екран буде посторінковим.

Якщо проміжні результати на якійсь із стадій конвеєра бажано зберегти, можна скористатись командою **`tee my_file`**. Ця команда візьме вхідний потік, передасть його без змін у вихідний потік і одночасно продублює у файл **`my_file`**[6]. Наприклад, так можна модифікувати один із розглянутих вище прикладів:

```
ps –ef | grep root | tee my_file | more
```

Тепер ми не лише побачимо на екрані посторінково виведений список всіх процесів ***root***'а, але й збережемо його у файлі `my_file`.

[6] Походження назви цієї команди добре пояснює її дію. Взагалі "tee" – це назва літери "T" в англійській абетці. Літера "T" якраз і використовується, щоби проілюструвати відгалуження, розщеплення потоку на основний і бічний (коли натякають на розщеплення потоку на два еквівалентні, використовують літеру "Y").

Завдання до виконання

1. Перейдіть у каталог **/bin**. Перегляньте список усіх файлів, що починаються із символу, який визначено в таблиці індивідуальних завдань.
2. Перегляньте список файлів, імена яких складаються з визначеної у таблиці індивідуальних завдань кількості символів.
3. Перегляньте список файлів, імена яких починаються із символів, які визначено в таблиці індивідуальних завдань. Зробіть це декількома способами.
4. Створіть в Вашому домашньому каталозі підкаталог `lab_4` і перейдіть в нього.
5. За допомогою команди `cat` створіть файл `my_text` і запишіть у нього кілька рядків. Потім за допомогою команди `cat` допишіть у нього ще кілька рядків.
6. Підрахуйте кількість файлів у каталозі, визначеному з таблиці індивідуальних завдань, використовуючи і не використовуючи конвеєри. Порівняйте результат.

Таблиця індивідуальних завдань

варіант	п.1	п.2	п.3	п.6, 7
1	**a**	**2**	**a, b, c, d**	**/bin**
2	**b**	**3**	**e, f, g, h**	**/usr**
3	**c**	**4**	**i, j, k, l**	**/usr/bin**
4	**d**	**5**	**m, n, o, p**	**/home**
5	**f**	**2**	**q, r, s, t**	**/var**
6	**g**	**3**	**u, v, w**	**/**
7	**h**	**4**	**x, y, z**	**Ваш домашній каталог**
8	**k**	**5**	**a, d, k, l**	**/tmp (або /var/tmp)**
9	**l**	**3**	**m, g, y**	**/sbin**
10	**n**	**2**	**x, z, r, q**	**/usr/sbin**

7. Підрахуйте кількість файлів у каталозі, визначеному з таблиці індивідуальних завдань, при цьому зберігши список файлів у файлі `filelist`, використовуючи команду `tee`.
8. Починаючи з Вашого домашнього каталогу, виведіть на екран у повному форматі назви усіх файлів і каталогів, що починаються на '**m**'. При цьому перед виводом кожної назви на екран повинен виводитися запит на його підтвердження.

9. Починаючи з кореневого каталогу, виведіть на екран імена всіх каталогів, що останній раз змінювалися більш 15 днів назад.
10. Виведіть на екран тільки час, що повертається командою `date`.
11. Виведіть на екран список усіх користувачів системи, тобто перші поля кожного рядка файлу `/etc/passwd` (роздільник полів — символ '**:**').
12. Виведіть на екран імена усіх файлів у каталозі `/bin`, що містять слова `Software` чи `software`. Потік помилок при цьому не повинний виводитися на екран.
13. Відсортуйте конфігураційний файл Вашого shell (`.profile`, `.cshrc`) відповідно до кодової таблиці ASCII так, щоб при цьому ігнорувалися пробіли на початку рядків.

Лабораторна робота №5. Процеси в ОС UNIX і керування ними

Мета

Оволодіння практичними навичками роботи з процесами — створення і знищення, керування процесами і їхній аналіз

Завдання для самостійної підготовки

1. Вивчити:

   поняття процесу і його характеристики;
   вивід на екран списку процесів і його аналіз;
   фонові й активні процеси;
   пріоритет процесів і його зміна;
   відправлення сигналів процесам, організація перехоплення сигналів;
   виконання завдань у системі в заданий час і з заданою періодичністю.

2. Ознайомитись з такими командами UNIX:

   `ps, ptree, pgrep, kill, pkill, fg, bg, jobs, crontab, at`

   Зверніть увагу на використання параметру командного рядка **`"&"`**

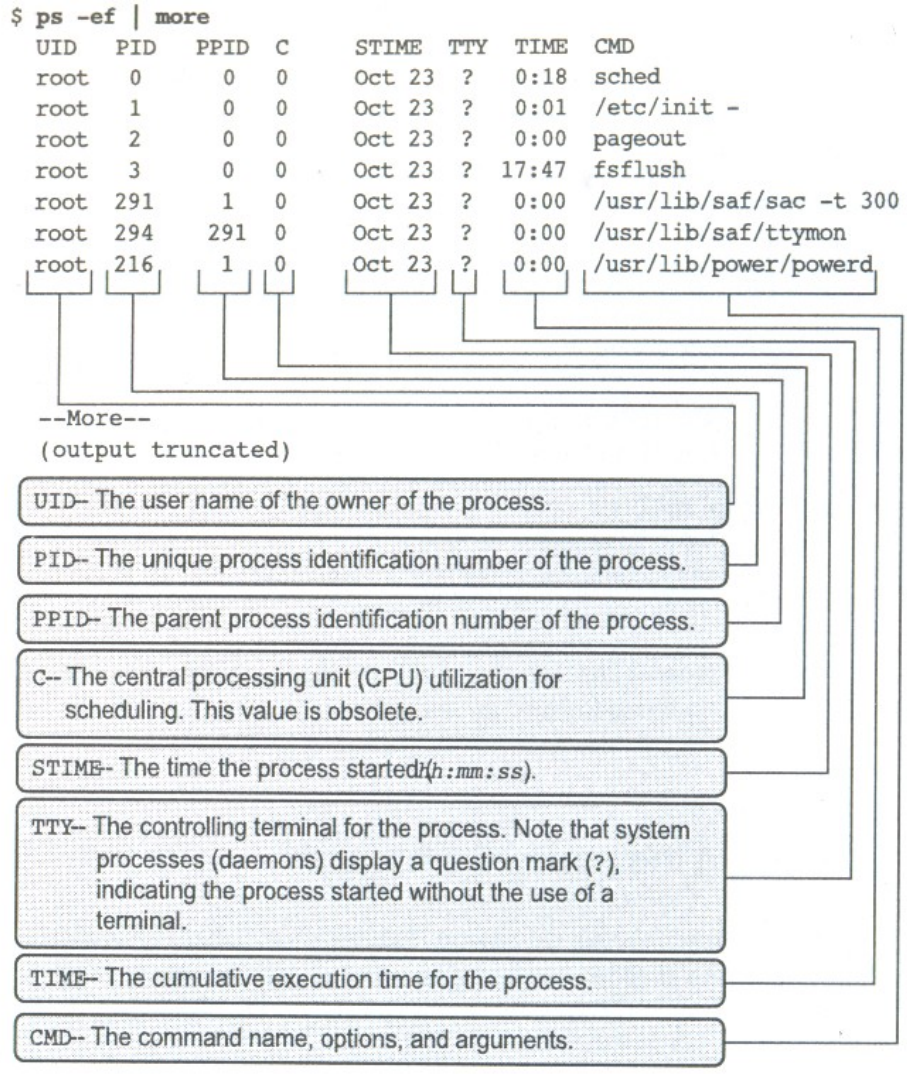
3. Скласти послідовність команд для виконання необхідного варіанта завдання

Довідковий матеріал

UNIX – багатозадачна система з розділенням часу. Це означає, що в системі одночасно виконується багато процесів. Кожний процес асоційований з певним користувачем, від імені якого цей процес діє. Для того, щоби переглянути список процесів, існує команда **`ps`**. Ця команда має багато ключів-модифікаторів, які визначають, яку саме інформацію про процеси повинна виводити команда. Слід зазначити, що в різних системах UNIX значення цих ключів може суттєво відрізнятись. Типові ключі: **`-a`** виводить інформацію про всі процеси, а не лише про процеси даного користувача, **`-l`** та **`-x`** виводять розширену інформацію про процес.

Кожний процес в системі має свій унікальний ідентифікатор – **`PID`**. За цим ідентифікатором можна звертатись до процесу. Крім того, кожний процес виникає не сам по собі – він має так званий батьківський процес, що характеризується ідентифікатором **`PPID`** (*parent process ID*). Таким чином

утворюється ієрархія процесів, що бере початок від початкового процесу **`init`**.



The ps -ef Command Output Description

Створення нового процесу у системі UNIX виконується у два етапи. Спочатку системний виклик `fork()` викликає “розщеплення” поточного процесу на два тотожні (різниця буде лише у тому, що процес-“нащадок” має інші **`PID`** і **`PPID`**). Далі виконується системний виклик `exec()`, який “підміняє” контекст поточного процесу іншим контекстом. Розглянемо типовий приклад, коли один процес має запустити на виконання інший – командна оболонка `sh` виконує команду `ls`, для чого утворює новий процес, в якому виконується програма `/usr/bin/ls`. Для цього програміст, який пише код `sh`, повинен передбачити такі кроки:

1. Якщо під час синтаксичного аналізу командного рядка виявлено, що необхідно запустити на виконання деяку команду, то виконується виклик `fork()`.

2. Наступна за `fork()` інструкція перевіряє значення, що повернув виклик. Якщо виклик `fork()` був успішний, то в результаті його виконання утворюється новий процес, тотожній батьківському, і операційна система буде виконувати їх обидва (в режимі розділення часу). Для процесу-"нащадка" значення, що повертає `fork()`, дорівнює 0, для батьківського процесу повертається значення PID нащадка.
3. Якщо це процес-нащадок, то виконується виклик `exec()`, якому в якості параметра передається ім'я файлу програми, яку необхідно запустити на виконання (в нашому прикладі – `/usr/bin/ls`). Якщо виклик був успішний, то на цьому виконання коду `sh` припиняється, і на його місце завантажується код `ls`.

Якщо це батьківський процес, то його дії залежать від того, які параметри були у командному рядку. Зокрема, якщо `ls` було запущено у пріоритетному режимі [7], то виконання `sh` призупиняється до завершення процесу-нащадка, якщо ж `ls` було запущено у фоновому режимі, то `sh` видає необхідні повідомлення про запуск фонового процесу і продовжує свою роботу, тобто знову приймає і редагує командний рядок.

Процеси можуть взаємодіяти між собою за допомогою так званих сигналів. Існує обмежена кількість сигналів, які мають свої числові ідентифікатори і мнемонічні позначення. Сигнали діють як переривання, тобто вони призупиняють процес, до якого вони направлені, і викликають відповідний обробник сигналу. Деякі із сигналів мають чітко визначене значення і обробляються системним обробником. Інші можуть перехоплюватись процесом, тобто процес може встановлювати для цих сигналів свій обробник. Звичайно, існують певні узгодження щодо призначення певних сигналів, і програмістам слід дотримуватись їх при розробці своїх обробників.

Користувач також може відправити процесу сигнал, для цього існує команда **`kill`**. Формат команди:

`kill [-<`*сигнал***`>] <`***PID***`>`**

<*сигнал***>** – це мнемонічне або числове позначення сигналу (наприклад, **`STOP`**, **`TERM`**, **`CONT`**, **`9`**), а **<***PID***>** –ідентифікатор процесу, якому посилають сигнал. Якщо не вказати параметр **<***сигнал***>**, то буде відправлено сигнал завершення процесу **`TERM`** (**`15`**). Цей сигнал може перехоплюватись процесом, але існує сигнал **`KILL`** (**`9`**), який не перехоплюється і безумовно знищує процес (якщо у користувача є достатньо для цього прав). Таким

[7] Про пріоритетний і фоновий режими див. далі

чином можна зупинити будь-який свій процес, якщо над ним втрачене керування (або процес "завис", що у системі UNIX трапляється дуже нечасто, або користувач не знає, яку команду процес може сприйняти). Для цього слід зайти з іншої консолі (віртуального або фізичного терміналу) і дати команду **kill -9 <** *PID***>**, де **<***PID***>** можна дізнатись за допомогою попередньої команди **ps**.

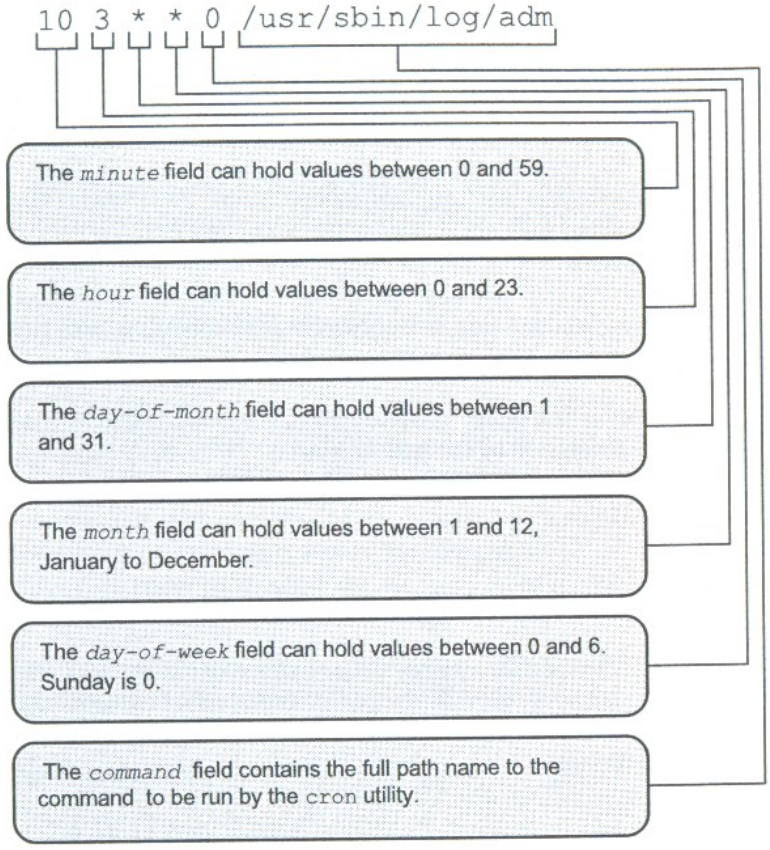
Команда **ps** без аргументів дає список лише процесів даного користувача і (в деяких системах) лише пов'язаних з конкретним терміналом. При роботі в графічній оболонці, а також при роботі на персональному комп'ютері, де підтримується певна кількість так званих віртуальних терміналів, користувач може працювати в системі, використовуючи одночасно кілька терміналів (наприклад, на персональних комп'ютерах між ними можна переключатись комбінаціями клавіш Alt+F#, де F# – одна з функціональних клавіш). Однак в часи створення системи UNIX такі можливості не передбачались. Передбачалось, що користувач має доступ лише до одного терміналу. Тому було розроблено систему завдань і фонового виконання процесів, щоби користувач міг одночасно виконувати різні задачі.

Коли користувач дає команду з консолі, в системі запускається процес, або кілька процесів. Якщо в командному рядку утворюється конвеєр (наприклад, **ls –l | wc –l**), то всі процеси (у нашому прикладі **ls** і **wc**) запускаються одночасно. Разом вони складають так зване завдання. Завдання пов'язано з терміналом. Поки воно не завершиться, користувач не має можливості вводити наступну команду. Це так званий пріоритетний (*foreground*) режим виконання завдання. Щоби під час виконання завдання мати можливість запускати інше завдання, перше з них слід запустити в так званому фоновому (*background*) режимі. Для того, щоби запустити завдання в фоновому режимі, слід завершити рядок команди символом **"&"** (після пробілу). При цьому стандартний ввід за замовчанням назначається порожньому файлу **/dev/null**. Слід врахувати, що завдання в фоновому режимі може намагатись здійснювати вивід на екран, заважаючи при цьому виводу пріоритетного процесу (спробуйте працювати, запустивши у фоновому режимі команду **ping**). Тому слід подбати, щоби фонові завдання здійснювали вивід у файли (див. Лабораторну роботу №4). Завдання, що було запущено у пріоритетному режимі, можна перевести у фоновий. Для цього необхідно спочатку призупинити виконання завдання (комбінація клавіш **CTRL-Z**). Далі можна поновити виконання завдання у пріоритетному режимі (команда **fg**) або у фоновому режимі (команда **bg**). Завдання мають свої номери. Переглянути їх можна за допомогою команди **jobs**.

Важливою можливістю є запуск певних завдань за розкладом. Це реалізується за допомогою програми-демона **crond**, який переглядає свої файли **crontab** (окремі для кожного користувача) і запускає завдання згідно

розкладу від імені того користувача, в файлі якого це завдання задано. Для зміни розкладу завдань треба відредагувати файл **crontab** і після цього перезапустити **crond**, що може зробити лише ***root*** та ті користувачі, яким це право надано. Останнім рекомендується для цього користуватись командою **crontab**, яка після редагування автоматично перезапускає **crond**.

Користувачу дозволено виконувати команду **crontab** тільки за умови, що його ім'я зустрічається в файлі **cron.allow** і не зустрічається в файлі **cron.deny** (в різних системах Unix ці файли можуть знаходитись в різних каталогах, наприклад **/usr/lib/cron/** або **/etc/cron.d/**). Якщо обидва файли відсутні, то тільки ***root*** може користуватись командою **crontab**. Якщо **cron.allow** не існує, **cron.deny** існує, але не містить імен, то використовувати команду **crontab** дозволено усім. Файли **cron.allow** і **cron.deny** повинні містити по одному імені в рядку.



First Five Fields in a crontab File

Існує спеціальна команда **at**, яка призначена для одноразового виконання заданої команди в будь-який заданий час. Правила дозволу і заборони користування командою **at** аналогічні **crontab** (файли **at.allow** і **at.deny**). Завдання, що задані командою **at**, виконуються тим же демоном **crond**.

Завдання до виконання

1. Перегляньте список процесів користувача (Вас).
2. Перегляньте повний список процесів, запущених у системі. При цьому гарантуйте збереження інформації від "утікання" з екрана (якщо процесів багато). Зверніть увагу на ієрархію процесів. Простежте через поля **`PID`** і **`PPID`** всю ієрархію процесів тільки-но запущеної Вами команди, починаючи з початкового процесу **`init`**. Зверніть увагу на формування інших полів виводу.
3. Запустіть ще один **`shell`**. Перегляньте повний список процесів, запущених вами, при цьому зверніть увагу на ієрархію процесів і на їхній зв'язок з терміналом. Використовуючи команду **`kill`**, завершіть роботу в цьому **`shell`**'і.
4. Перегляньте список задач у системі і проаналізуйте їхній стан.
5. Запустіть фоновий процес командою
   `find / -name "*.c" -print > file 2> /dev/null &` [8]
6. Визначте його номер. Відправте сигнал призупинення процесу. Перегляньте список задач у системі і проаналізуйте їхній стан. Продовжить виконання процесу. Знову перегляньте список задач у системі і проаналізуйте його зміну. Переведіть процес в активний режим, а потім знову у фоновий. Запустіть цей процес із пріоритетом 5.
7. Виведіть на екран список усіх процесів, запущених не користувачем ***root***.
8. Організуйте вивід на екран календаря <1996+№варіанту> року через 1 хвилину після поточного моменту часу.
9. Організуйте періодичне (щоденне) видалення в домашньому каталозі усіх файлів з розширенням **`*.bak`** і **`*.tmp`**.

[8] Увага! Ця команда коректно працює в оболонках `sh` і `bash`, і не працює коректно у оболонках `csh` і `tcsh` (це пов'язано з правилами перенаправлення потоку помилок, див. Лабораторну роботу №3).

Лабораторна робота №6. Професійна робота з командними оболонками

Мета

Оволодіння практичними навичками професійної роботи з командною оболонкою shell – використання змінних і створення командних файлів.

Завдання для самостійної підготовки

1. Вивчити:
   - організацію умовного виконання командного рядка, угруповання команд у командному рядку;
   - використання змінних shell;
   - організація командних файлів: передача параметрів, уведення значень, умовні розгалуження і цикли;
   - арифметичні обчислення в shell.
2. Розробити алгоритм рішення відповідно до завдання
3. Скласти програми рішення завдань
4. Підготувати тест для перевірки програм

Довідковий матеріал

У попередніх роботах ми вже познайомились з командними оболонками (*shell*). У цій роботі розглянемо прийоми професійної роботи з командними оболонками, а саме використання змінних оточення і створення командних файлів.

Змінні оточення

Усі змінні вашого оточення виводяться за допомогою команди *set*, ознайомтесь з ними.

Командні файли

Командний файл, або сценарій (також дуже часто кажуть "*скрипт*" від англійського *script* – сценарій) є текстовим файлом, який оформлено з дотриманням певних правил, і який містить команди, у найпростішому випадку повністю аналогічні тим командам, що вводяться з клавіатури. Командна оболонка здатна запускати такий файл на виконання і послідовно виконувати команди, що містяться в ньому. Для користувача, що запустив

цей сценарій, його виконання буде виглядати як виконання звичайної програми.

Зверніть увагу на розбіжності у різних програмних оболонках shell, які суттєві для програмування. Під час виконання роботи впевніться, в якій із програмних оболонок Ви працюєте (зазвичай, для FreeBSD це `csh` чи `tcsh`, а для Linux – `bash`, який є розвитком `sh`), і яка буде запускатись для виконання Вашого командного файлу (це визначається першим рядком Вашого командного файлу). Уважно прочитайте правила використання операторів `if` і формування перевірки відповідної умови. Зверніть увагу на команду `test`.

Важливою можливістю командних оболонок (усіх) є обробка так званих пакетних файлів.

Завдання до виконання

Написати скрипт для виконання наступного завдання:

Частина 1

1. Визначити, хто є користувачем системи та виведіть на екран.
2. За допомогою змінних оточення визначити домашній каталог користувача
3. Знайти всі файли, які належать вам у вашому домашньому каталозі
4. За архівувати ці файли з іменем back_up_<your_name>_<data>.
5. Якщо архів з даним іменем вже існує, то вивести запит на його перезапис.
6. Встановити права на отриманий архів тільки для читання.

Частина 2

1. Розархівувати файли з архіву зі збереженням структури каталогів
2. Якщо файли відрізняються у порівнянні з оригіналом, то потрібно виводитися запит на його перезапис.