Материалы для подготовки к зачёту по СИТу 2025

1 Linux

1.1 Базовые команды терминала UNIX-подобных систем

Основные команды, использующиеся в терминале Linux:

- *touch* <*file*> — создать новый пустой файл;
- *mv* <*source*> <*destination*> — переместить/переименовать файл;
- *cp* <*source*> <*destination*> — скопировать файл;
- *rm* <*file*> — удалить файл;
- *cat* <*file*> — вывести содержимое файла на экран (если передать несколько файлов, cat их сконкатенирует и выведет);
- *less* <*file*> — пагинатор, позволяет вывести часть файла на экран и перелистывать его;
- *echo „text“* — вывод текста;
- *man* <*command*> — информация о команде (мануал);
- *ls* — вывести список файлов в текущей директории;
- *cd* <*directory*> — перейти в другую директорию;
- *mkdir* <*directory*> — создать новую директорию;
- *find* — найти файл;
- *grep* — найти строки, соответствующие определённому шаблону;
- *wc* — посчитать количество строк, слов и байт в файле;
- *history* — показать историю терминала, т.е. последние использованные команды;
- *lsblk* — список физических устройств (дисков);
- *mount* — монтирование файловых систем;
- *reboot* — перезагрузка;
- *shutdown* — выключение;
- *sl* — самая важная команда по моему мнению :)

1.2 Понятие корневого каталога и домашней директории. Переменные среды. Переменная PATH

Корневой каталог (/) в Linux — самый верхний уровень иерархии файловой системы. Как правило он содержит следующие директории:

- /bin — бинарные файлы;
- /boot — файлы, необходимые для загрузки системы;
- /dev — файлы устройств;
- /etc — конфигурационные файлы;
- /lib — системные библиотеки;
- /mnt — примонтированные файловые системы;
- /opt — дополнительные приложения;
- /root — домашний каталог пользователя root;
- /sys — системные файлы;
- /tmp — временные файлы;
- /var — логи и базы данных.

У каждого пользователя в системе есть домашняя директория, которая лежит на пути /home/<username>. «.» обозначает текущую директорию, «..» — родительскую директорию, «~» — домашнюю директорию, эти обозначения можно использовать для более компактного использования команды cd.

Чтобы создать переменную среды, используется команда *export KEY=„value“*. Эта переменная сохранится только для текущего сеанса оболочки, чтобы она сохранилась после перезагрузки, нужно записать её в файл *.bashrc*. Для вывода всех переменных, которые есть в системе, используется команда *env*, а значение одной переменной можно посмотреть с помощью команды *printenv <VARIABLE>* или *echo $VARIABLE*. Названия переменных являются чувствительными к регистру. Также важно, что значения всех переменных являются строками.

Переменная $PATH содержит набор каталогов, где система ищет исполняемые файлы при запуске какой-либо команды. Это избавляет от необходимости каждый раз вводить полный путь до исполняемого файла команды и позволяет использовать короткую запись.

1.3 Флаги и аргументы при командах. Примеры наиболее важных флагов для базовых команд. Получение аргументов в скрипте

Синтаксис команды обычно выглядит как <название команды> -флаги [аргументы]. Флаги несколько изменяют работу самой команды, тогда как в качестве аргументов передаются данные, с которыми работает команда. Несколько примеров базовых флагов:

1. -f (--force) — выполняет команду насильно, игнорируя предупреждения, может необратимо навредить файлам;

2. -v (--verbose) — выводит все ошибки, предупреждения и логи;
3. -h (--help) — выводит краткую справку о команде;
4. -r — *rm -r* рекурсивно удаляет всю директорию;
5. -a — *ls -a* выводит все файлы, в том числе скрытые;
6. -j — *zip -j* не сохраняет иерархическую структуру файлов при архивации;
7. -l — *wc -l* выводит только количество строк в файле.

У большей части флагов есть короткое название из одной буквы, которое записывается через -, и длинное название, которое записывается через -- (исключение составляет команда find, у которой длинные аргументы записываются через один минус, например, *find -name*). Несколько коротких флагов можно объединять, например, -f -a — то же самое, что -fa.

Чтобы получить аргументы внутри скрипта, нужно использовать синтаксис $n, где n — номер аргумента. $1, $2, . . . — первый, второй и т.д. аргументы, $0 — само название скрипта.

1.4 Перенаправление вывода в файл и из файла. Понятие конвейера, примеры применения конвейеров

Оператор > перенаправляет вывод команды в файл. Например, *echo „hello“ > output.txt*.

Оператор < перенаправляет ввод из файла. Например, *wc -l < input.txt*.

Конвейер обозначается символом | и передаёт данные из стандартного вывода одной команды на стандартный ввод другой. Конвейер удобно использовать в сочетании с инструментом xargs, который разбивает полученный ввод и передаёт его как аргументы команде. Например, *find . . . | xargs cat* объединит и выведет все найденные файлы.

У команды find также есть специальный флаг -exec, который работает аналогично конвейерам. Его синтаксис выглядит следующим образом: *find . . . -exec <название команды> {куда подставить результат выполнения find} \; (+)*. «\;» подставляет файлы по очереди (\ экранирует точку с запятой от bash), «+» подставляет все файлы сразу.

1.5 Небольшие практические задачи

Найти все *.cpp* файлы в домашней директории:

```
find . -name "*.cpp"
```

Посчитать строки в файле:

```
wc -l < "$1"
```

1.6 Стадии компиляции C/C++ кода. Команды компиляции на примере GCC. Системы сборки: что это и зачем они нужны

Компиляция C/C++ кода состоит из следующих стадий:

1. Препроцессинг — подготовка кода к компиляции, работа с препроцессорными директивами: добавление #include в код, замена макросов их значениями, удаление комментариев и т.д.

2. Компиляция — преобразование кода в ассемблерный код (создаётся ассемблерный файл *.s*).

3. Ассемблирование — перевод ассемблерного кода в машинный (создаётся объектный файл *.o*).

4. Линковка — связывание всех объектных файлов в один исполняемый (например, *a.out*), который затем можно запустить с помощью команды *./a.out*.

GCC (GNU C Compiler) — компилятор языка Си, g++ — компилятор языка C++. Для того, чтобы скомпилировать файл, нужно использовать следующий синтаксис: *gcc filename.c (-o <название исполняемого файла>)*.

Системы сборки нужны для того, чтобы автоматизировать процесс компиляции, особенно когда проект состоит из большого количества файлов. Самой распространённой системой сборки является CMake, она генерирует makefile с набором инструкций, к которому затем применяется команда *make*.

2 Git

2.1 Базовые понятия в Git. Состояния файлов и команды, переводящие файлы между этими состояниями

Git — система контроля версий. Чтобы создать git репозиторий в текущей директории, нужно использовать команду *git init*.

Основные состояния файлов:

1. untracked — файл не отслеживается;

2. unmodified — файл не был изменён с момента последнего коммита;

3. modified — файл был отредактирован с момента последнего коммита;

4. staged — изменения в файле были зафиксированы и будут добавлены при выполнении коммита.

Перевод файлов между состояниями:

- *git add file* — untracked/modified → staged;
- *git commit -m "message"* — staged → unmodified;
- *git rm file* — staged/modified/unmodified → untracked.

Для просмотра текущего состояния файлов используется команда *git status*. Команда *git diff* показывает внесённые изменения.

Чтобы избежать индексирования определённых файлов (например, временных или бинарных), их названия или маски можно добавить в файл *.gitignore*.

2.2 Работа с удалённым репозиторием. Связывание локального репозитория с удалённым. Отправка изменений с локального в удалённый репозиторий. Получение изменений с удалённого репозитория. Форки

Локальный репозиторий находится на своём компьютере, тогда как удалённый — на сервере (например, на GitHub).

Чтобы склонировать к себе удалённый репозиторий, можно использовать команду *git clone url*. Чтобы добавить к себе удалённый репозиторий, используется команда *git remote add name url*, флаг -v (*git remote -v*) позволяет посмотреть все добавленные удалённые репозитории. После добавления к себе репозитория можно получать оттуда изменения с помощью команды *git fetch name*. *git pull* автоматически сливает изменения из удалённой ветки с текущей локальной.

Чтобы отправить изменения с локального в удалённый репозиторий, используется команда *git push -u name branch*. Флаг -u в данном случае используется для создания в удалённом репозитории ветки, соответствующей локальной.

В публичные проекты как правило нельзя вносить изменения, поэтому можно с использованием веб-интерфейса сделать форк репозитория, с которым уже можно будет работать в своём аккаунте, как с обычным удалённым репозиторием. Если Вы хотите, чтобы изменения были добавлены в оригинальный репозиторий, можно сделать pull request, который должен быть одобрен владельцем оригинального репозитория.

2.3 Отмена внесённых ранее изменений. Откат репозитория (--soft, --hard варианты)

Для отмены ранее внесённых изменений используются следующие две команды:

- *git reset HEAD∼n*, где n — число коммитов, на которое сбрасывается состояние указателя HEAD. У этой команды есть три режима:
  - --soft — мягкий режим: изменения остаются в индексе и в рабочей директории, но удаляются из репозитория;
  - --mixed — смешанный режим: изменения остаются в рабочей директории, но удаляются из репозитория и индекса;
  - --hard — жёсткий режим: изменения полностью удаляются из репозитория, индекса и рабочей директории;
- *git revert commit-id* — откатывает изменения, создавая для этого новый коммит и не удаляя ошибочные коммиты из истории.

Чтобы изменить предыдущий коммит, например, добавить в него дополнительные файлы или изменить сообщение коммита, используется команда *git commit --amend*.

2.4 Работа с ветками в Git. Просмотр всех веток, создание, переключение локального репозитория на конкретную ветку. Объединение веток, возможные проблемы при этом

В репозитории Git может быть несколько веток помимо основной, которая обычно называется main или master. Основные команды для работы с ветками и их флаги:

- *git branch* — список локальных веток;
- *git branch -r* — список удалённых веток;
- *git branch -a* — список всех веток (локальных и удалённых);
- *git branch branch-name* — создание новой ветки с названием branch-name;
- *git checkout branch-name* — переключиться на ветку с названием branch-name;
- *git checkout -b branch-name* — создать новую ветку и сразу переключиться на неё.

Чтобы объединить некоторую ветку с данной, используется команда *git merge*. При объединении веток могут возникнуть так называемые merge-конфликты, когда изменения в ветках противоречат друг другу. Такие конфликты решаются вручную, если же их не возникает, слияние происходит автоматически.

Для объединения изменений двух веток с сохранением линейности истории можно также использовать команду *git rebase*.

2.5 Сохранение изменений в буфер («копилку»). Основные команды, возможное применение

Для сохранения изменений в буфер используется команда *git stash*. Она сохраняет («прячет») неподтверждённые изменения (индексированные и неиндексированные) в отдельном хранилище, чтобы к ним можно было вернуться позже. Эта команда может быть полезна, например, для внесения срочных правок в другой ветке.

- *git stash* — «прячет» внесённые изменения;
- *git stash pop* — убирает изменения из хранилища и применяет их к рабочей копии;
- *git stash apply* — применяет изменения к рабочей копии, но оставляет их и в хранилище.

3 LaTeX

3.1 Понятие класса документа. Примеры классов документов

На тип документа указывает команда `\documentclass{}`. Основные типы документов:

- article — для оформления статей;
- book — для оформления книг;
- letter — для оформления писем;
- beamer — для оформления презентаций;
- CSWorks — для оформления научных работ на факультете КНиИТ.

У команды `\documentclass` есть параметры, которые указываются в квадратных скобках до фигурных скобок. Основные параметры:

- размер шрифта (10 pt, 12pt, 14pt) — по умолчанию 10pt;
- тип бумаги (a4paper, letterpaper) — по умолчанию letterpaper;
- нужна ли пустая страница после титульного листа (titlepage, notitlepage) — для article по умолчанию notitlepage;
- тип печати (oneside, twoside);
- количество колонок (onecolumn, twocolumn).

3.2 Понятие преамбулы документа, подключение пакетов. Основные пакеты для базовой компиляции документа на русском языке

Сам документ заключён между командами `\begin{document}` и `\end{document}`, которые называются окружением. Перед началом окружения располагается преамбула, которая задаёт правила форматирования и другие настройки. Пакеты подключаются с помощью команды `\usepackage`. Можно вынести преамбулу в отдельный файл *preamble.sty* и подключить его из основного файла командой `\usepackage{preamble}`. Для отображения русского языка необходимо подключить следующие три пакета:

```
\usepackage[T2A] {fontenc}
\usepackage[utf8] {inputenc}
\usepackage[russian] {babel}
```

3.3 Секционирование документа (основные команды). Взаимодействие секционирования с результатом выполнения команды tableofcontents. Структурирование документа (разделение на поддокументы с последующей компиляцией воедино). Хорошие практики при структурировании документа

Чтобы соблюдать логические границы между различными частями документа, существует секционирование, то есть разделение документа на секции. Имеет место вложенность вида: глава (только для книги) $\rightarrow$ секция $\rightarrow$ подсекция $\rightarrow$ подподсекция. Используются следующие обозначения:

- `\chapter{}` — глава, где в фигурных скобках при необходимости пишется название;
- `\chapter*{}` — глава, которой не присваивается номер (без названия не имеет смысла);
- `\section(*){}` — секция, которой присваивается (соотв. не присваивается) номер;
- `\subsection(*){}` — подсекция, которой присваивается (соотв. не присваивается) номер;
- `\subsubsection(*){}` — подподсекция, которой присваивается (соотв. не присваивается) номер.

Создание новой секции в статье или главы в книге автоматически заканчивает текущую и начинает новую страницу.

В документе можно создать содержание, куда автоматически включаются нумерованные главы, секции и подсекции, с помощью команды `\tableofcontents`.

Хорошей практикой является разделение документа на поддокументы. Для этого части документа сохраняются в отдельные файлы, а затем подключаются с помощью команды `\input{}`, где в фигурных скобках указывается название файла и путь к нему, если он лежит не в текущей директории. Если подключаемый файл имеет формат *.tex*, то содержимое этого файла будет предварительно скомпилировано.

3.4 Форматирование текста. Базовые команды для оформления знаков препинания: тире, дефиса, кавычек. Команды для оформления жирного и курсивного шрифтов

Абзацы отделяются друг от друга пустой строкой, перенос на новую строку осуществляют команды `\newline` и `\\`, перенос на новую страницу — `\newpage`.

После разделительных знаков препинания, но не перед ними, ставится пробел; скобки с обеих сторон выделяются пробелами, вокруг дефиса пробелы не ставятся. Некоторые знаки препинания записываются особым образом:

- тире: `"---` или `~---`;
- числовые интервалы: `--`;
- дефис: `"=`;
- кавычки-ёлочки: `<<text>>`.

Существуют команды, изменяющие размер шрифта:

- `\tiny` (5pt);
- `\scriptsize` (7pt);
- `\footnotesize` (8pt);
- `\small` (9pt);
- `\normalsize` (10pt);
- `\large` (12pt);
- `\Large` (14pt);
- `\LARGE` (17pt);
- `\huge` (20 pt);
- `\Huge` (25pt).

Жирный шрифт оформляется с помощью команды `\textbf`, а *курсив* — с помощью команды `\textit`. Команда `\verb|command|` позволяет вставить в документ команду LaTeX без её выполнения. Окружения flushleft, center и flushright обеспечивают выравнивание по левому краю, по центру и по правому краю соответственно.

3.5 Понятие списка. Виды списков, оформление вложенных списков

Список — перечисление элементов. Если элементы списка представляют собой целые предложения, то в конце каждого элемента обычно ставится точка, если же это слова или словосочетания, они разделяются точками с запятой. Элементы списка разделяются с помощью команды `\item`.

Существует три вида окружения для списков:

- `\begin{itemize} \end{itemize}` — создаёт ненумерованный список;
- `\begin{enumerate} \end{enumerate}` — создаёт нумерованный список;
- `\begin{description} \end{description}` — создаёт список-описание (в таком случае каждый элемент начинается как `\item[заголовок элемента]`.

3.6 Понятие математического окружения. Синтаксис. Встроенные формулы и выносные формулы. Особенности математического окружения: работа с отступами, кириллицей

Для того, чтобы оформлять математические формулы, в LaTeX есть математическое окружение. Существует два типа формул: встроенные и выносные.

Встроенные формулы располагаются внутри текста, они выглядят как команды, заключённые в два знака $ (`$equation$`).

Для выносных формул можно использовать окружение `\begin{equation} \end{equation}` (для нумерованных формул) или `\begin{equation*} \end{equation*}` (для ненумерованных формул). Окружение `\begin{align} \end{align}` позволяет выравнивать длинные формулы.

Верхний индекс обозначается с помощью символа ^, а нижний — с помощью символа _. Индексы из нескольких символов заключаются в фигурные скобки.

Знак умножения ставится с помощью команды `\cdot`, дробь записывается как `\frac{}{}`. Знаки сравнения записываются как `\le` ($\le$), `\leqslant` ($\leqslant$), `\ge` ($\ge$), `\geqslant` ($\geqslant$), `\ne` ($\ne$).

Некоторые примеры команд математического окружения:

- Греческие буквы: `\alpha` (α), `\epsilon` (ϵ), `\varepsilon` (ε);
- Сумма, произведение, интеграл: `\sum` ($\sum$), `\prod` ($\prod$), `\int` ($\int$);
- Корень n-ной степени: `\sqrt[n]{}` ($\sqrt[3]{x}$).

Чтобы в математическом режиме отображался текст, его нужно оборачивать в команду `$\text{}$`, для отображения пробелов применяется тильда $\sim$.

3.7 Работа со сложными объектами. Базовые команды для создания матриц, таблиц, систем уравнений. Синтаксис

Для создания матриц используются следующие окружения:

- `\begin{matrix} \end{matrix}` — матрица без скобок;
- `\begin{bmatrix} \end{bmatrix}` — матрица в квадратных скобках [ ];

- `\begin{pmatrix} \end{pmatrix}` — матрица в круглых скобках ( );
- `\begin{vmatrix} \end{vmatrix}` — матрица в прямых скобках | |;
- `\begin{Bmatrix} \end{Bmatrix}` — матрица в фигурных скобках { };
- `\begin{Vmatrix} \end{Vmatrix}` — матрица в двойных прямых скобках ‖ ‖.

Элементы матрицы в рамках одной строки разделяются символом &, а строки матрицы — символом \\.

Простая таблица создаётся с помощью окружения `\begin{tabular} \end{tabular}`, если нужна подпись и ссылка, используется окружение `\begin{table} \end{table}`. Разделение элементов выполняется так же, как в матрице.

Для создания систем уравнений применяется окружение `\begin{cases} \end{cases}`.

4 Работа с программным кодом. Аргументы команд пакета minted или Verbatim (fancyvrb). Основные опциональные параметры

Для вставки кода можно использовать пакет minted, который подключается с помощью команды `\usepackage{minted}`. Когда код пишется в теле документа, он выделяется с помощью команды `\begin{minted}[options]{language} \end{minted}`, а для подключения кода из отдельного файла используется синтаксис `\inputminted[options]{language}{filename}`. Основные опциональные параметры:

- frame=lines/leftline/topline/bottomline — рисует линии вокруг кода, чтобы выделить его;
- framesep=2mm/pt — дистанция между текстом и окружением кода;
- baselinestretch — дистанция между строками;
- bgcolor — цвет фона;
- fontsize — размер шрифта;
- style=bw — установка чёрно-белого стиля кода;
- breaklines — перенос длинных строк;
- linenos — включение отображения номеров строк.

5 Вставка изображений. Окружение figure, позиционирование плавающих объектов. Масштабирование изображений

Плавающие объекты вставляются с помощью окружения `\begin{figure} \end{figure}`. Внутри этого окружения существует команда для центрирования объекта `\centering`, а также параметры, отвечающие за то, как объект будет вставлен в документ:

- h — вставка на месте, т.е. ближе всего к реальному расположению объекта в документе;
- H — вставка **ровно** на том месте, где объект указан в документе;
- t (b) — вставка вверху (внизу) страницы;
- p — вставка на специальной странице для плавающих объектов;
- ! — попытаться перегрузить внутренние параметры LaTeX, чтобы определить «хорошую» позицию.

Рисунки вставляются командой `\includegraphics[options]{file}`. Для изменения относительного размера изображения используется параметр scale. Если он равен 1.0, изображение не изменит масштаб, если он равен 2.0 — увеличится в 2 раза, 0.5 — уменьшится в 2 раза. Команда `\caption` вставляет подпись к рисунку.