Учебный курс L-201

Расширенное администрирование GNU/Linux(LPI - уровень 2)

версия 1.3

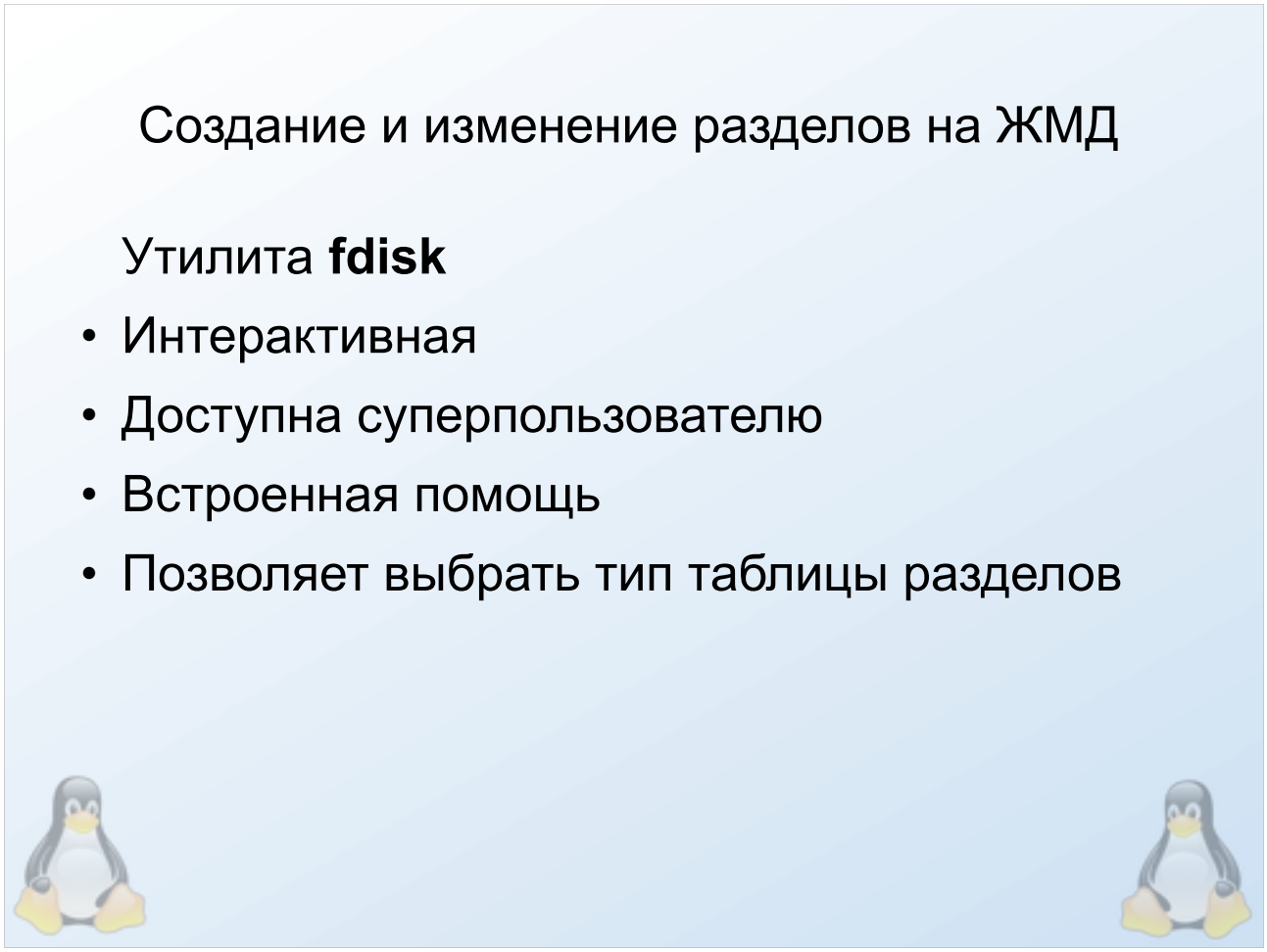
Екатеринбург, 2023

Оглавление

Глава 1. Файловые системы в Linux

1.1. Создание и изменение разделов на ЖМД

1.1.1. Утилита fdisk.



1. Интерактивная утилита `fdisk` позволяет оперировать с дисковыми разделами жестких магнитных дисков, обладая специальным набором собственных команд.
2. Команда `fdisk` доступна только администратору.
3. Для редактирования таблицы разделов на диске эту команду следует запустить в интерактивном режиме, указав файл устройства для требуемого жесткого диска в качестве аргумента команды:

Пример:

```
# fdisk /dev/hda

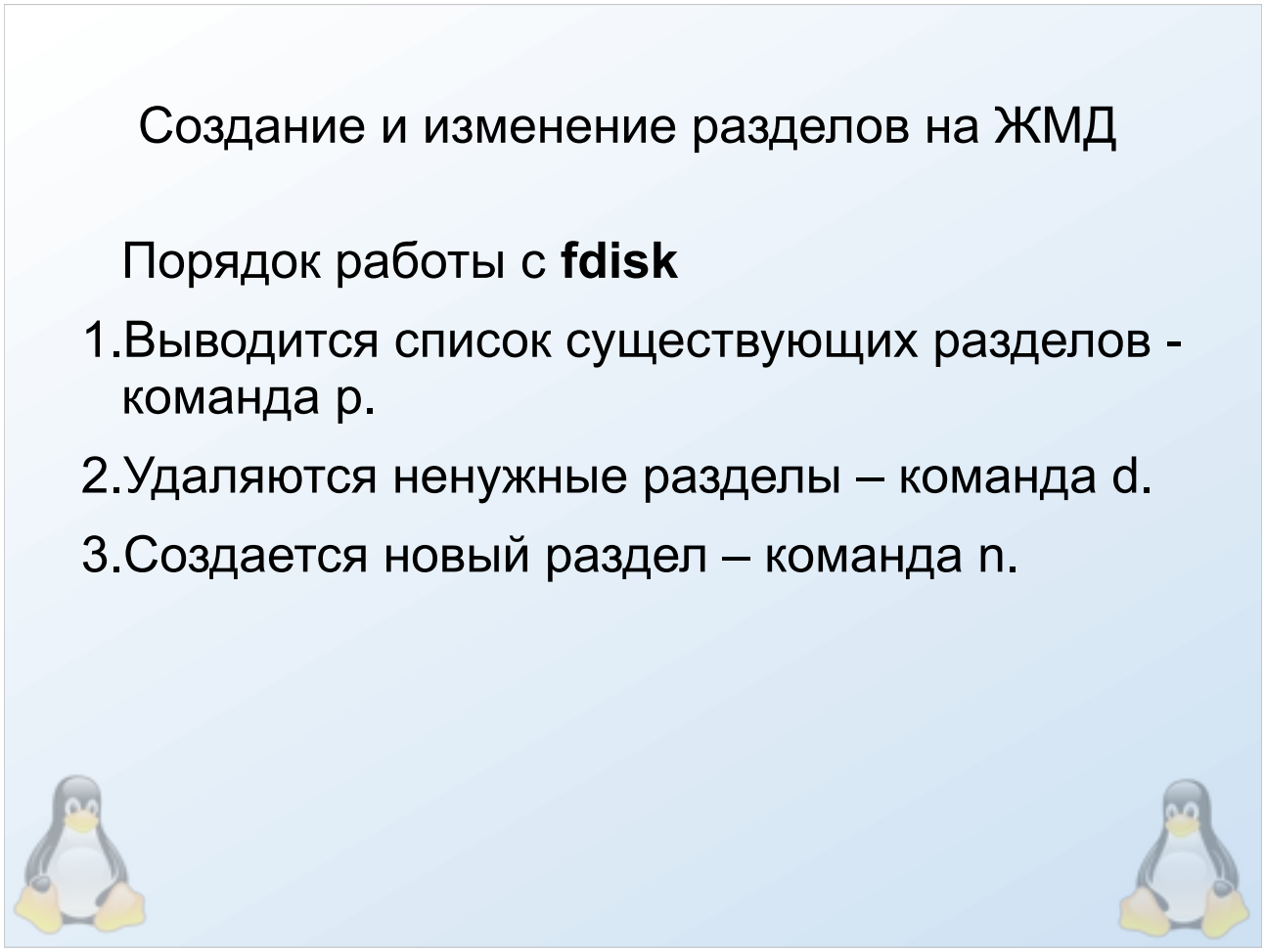
The number of cylinders for this disk is set to 19077.
```

```
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): m
```

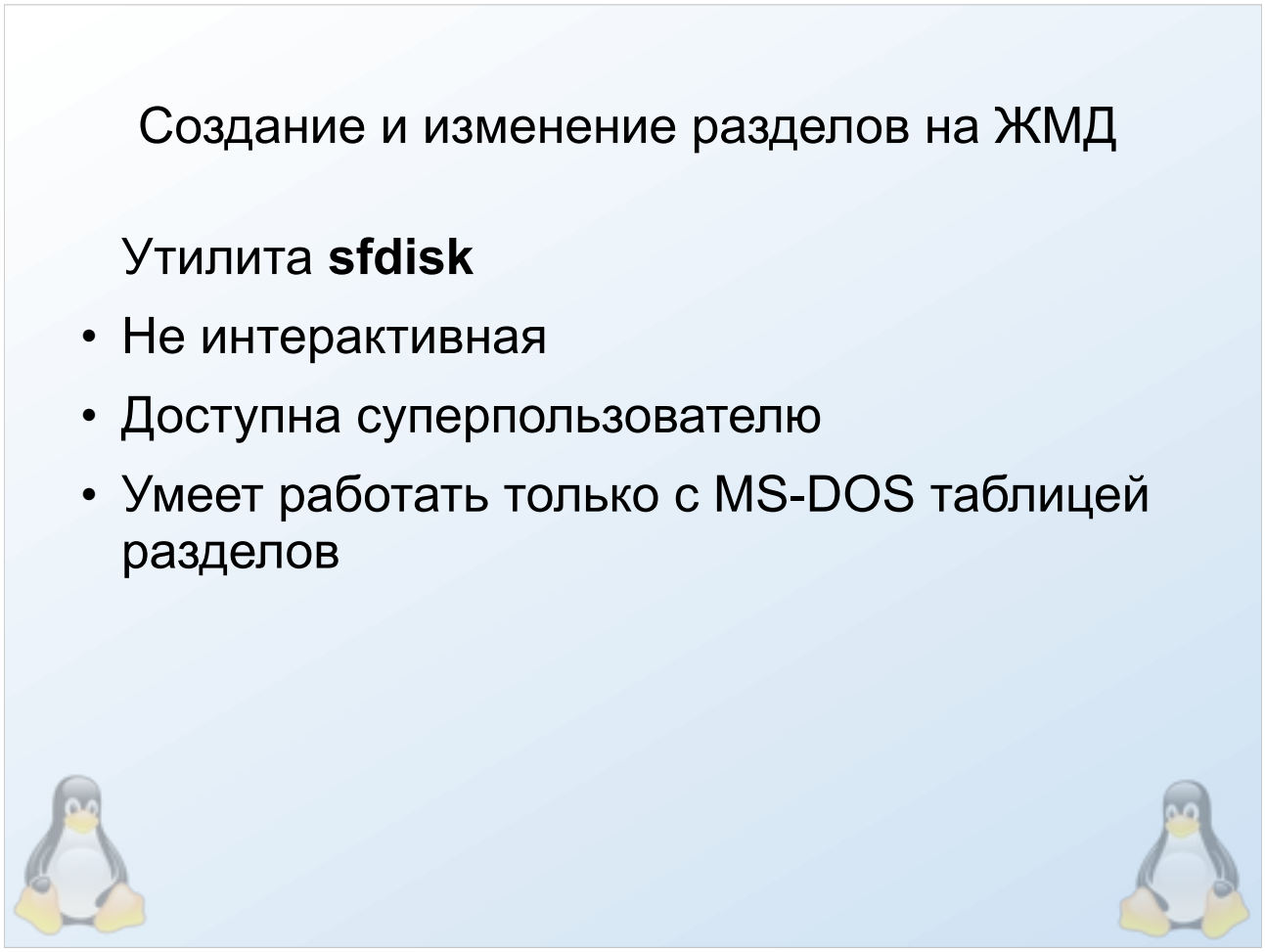
Примечание: В этом примере команда `fdisk` была выполнена с аргументом - файлом устройства первого IDE диска.

4. Список основных опций команды `fdisk`:
    1. `-l` – выводит список разделов и выходит. Если устройство не указано выводит файл `/proc/partitions`
    2. `-u` – выводит список разделов в секторах, а не цилиндрах;
    3. `-s раздел` – вывод размера указанного раздела;
    4. `-b sectorsize` – указывает размер сектора на диске;
    5. `-C` – число цилиндров на диске;
    6. `-H` – число головок на диске;
    7. `-S` – число секторов на трек.



5. Обычная последовательность работы с утилитой `fdisk` для создания нового раздела на жестком диске:
   1. Выводится список существующих разделов на диске - команда `p`.
   2. Удаляются ненужные разделы (при этом данные на них теряются безвозвратно) – команда `d`.
   3. Создается новый раздел – команда `n`. При этом будет запрошены:
      1. тип раздела (`p` – первичный, `e` – расширенный, `l` – логический);
      2. номер раздела;
      3. первый и последний цилиндры раздела; при этом последний цилиндр нового раздела можно указать абсолютно или, после знака `+`, размер раздела в килобайтах (например, `+15000k`) или мегабайтах (`+1200M`).
6. Если новый раздел должен иметь тип, отличный от принятого по умолчанию (`83` – Linux Native), то необходимо указать тип раздела – команда `t` (получение списка возможных типов разделов – команда `l`).
7. Если все требуемые разделы созданы, то необходимо сохранить изменения – команда `w`.

1.1.2. Утилита sfdisk.



1. `sfdisk` не интерактивная утилита для работы с разделами на жестком диске.
2. `sfdisk` имеет четыре основные функции:
   1. перечисление размеров разделов;
   2. перечисление разделов на устройстве;
   3. проверка разделов на устройстве;
   4. изменение разделов на устройстве.
3. Для создания разделов можно использовать специальную таблицу из файла:

   `<start> <size> <id> <bootable> <c,h,s> <c,h,s>`

Где поля разделяются пробелами, запятыми или точкой с запятой. Поля означают следущее:

1. `<start>` - начало раздела. По умолчанию первый незанятый сектор(цилиндр).
2. `<size>` - окончание или размер раздела. Если параметр пропущен, то используется все оставшееся свободное пространство.
3. `<id>` - идентификатор раздела. По умолчанию 83. Можно использовать `[E|S|L|X]`, где `L (LINUX_NATIVE (83)), S LINUX_SWAP (82), E XTENDED_PARTITION (5), X LINUX_EXTENDED (85).`

4. <bootable> - установить флаг загруки.
5. <c,h,s> - геометрия начала или конца раздела.

1. Основные опции команды sfdisk:
   1. -v or --version – вывод информации о версии;
   2. -? or --help – печать краткой справки;
   3. -T or --list-types – вывести известные типы разделов;
   4. -s or --show-size – перечислить размеры разделов;
   5. -g or --show-geometry – вывести информацию о геометрии из ядра для указанного устройства;
   6. -l or --list – перечисление разделов на устройстве;
   7. -d – сохранить информацию о разделах устройства. Затем эту информацию можно использовать для восстановления разделов.

Пример:

```
$ sfdisk -d /dev/hda > hda.out
$ sfdisk /dev/hda < hda.out
```

   8. -V or --verify – проверка корректности таблицы разделов;
   9. -i or --increment – стартовать с 1 цилиндра вместо 0;
   10. -N number – изменить только один указанный раздел;
   11. -Anumber – сделать указанный раздел активным, все остальные неактивными;
   12. -c or --id number [Id] – Если не указан аргумент Id, то вывести идентификатор раздела. Если аргумент Id указан, то изменить у выбранного раздела идентификатор. Также имеются длинные опции для вывода или изменения идентификтора раздела --print-id и –change-id.

Пример:

```
$ sfdisk --print-id /dev/hdb 5
 6
$ sfdisk --change-id /dev/hdb 5 83
 OK
```

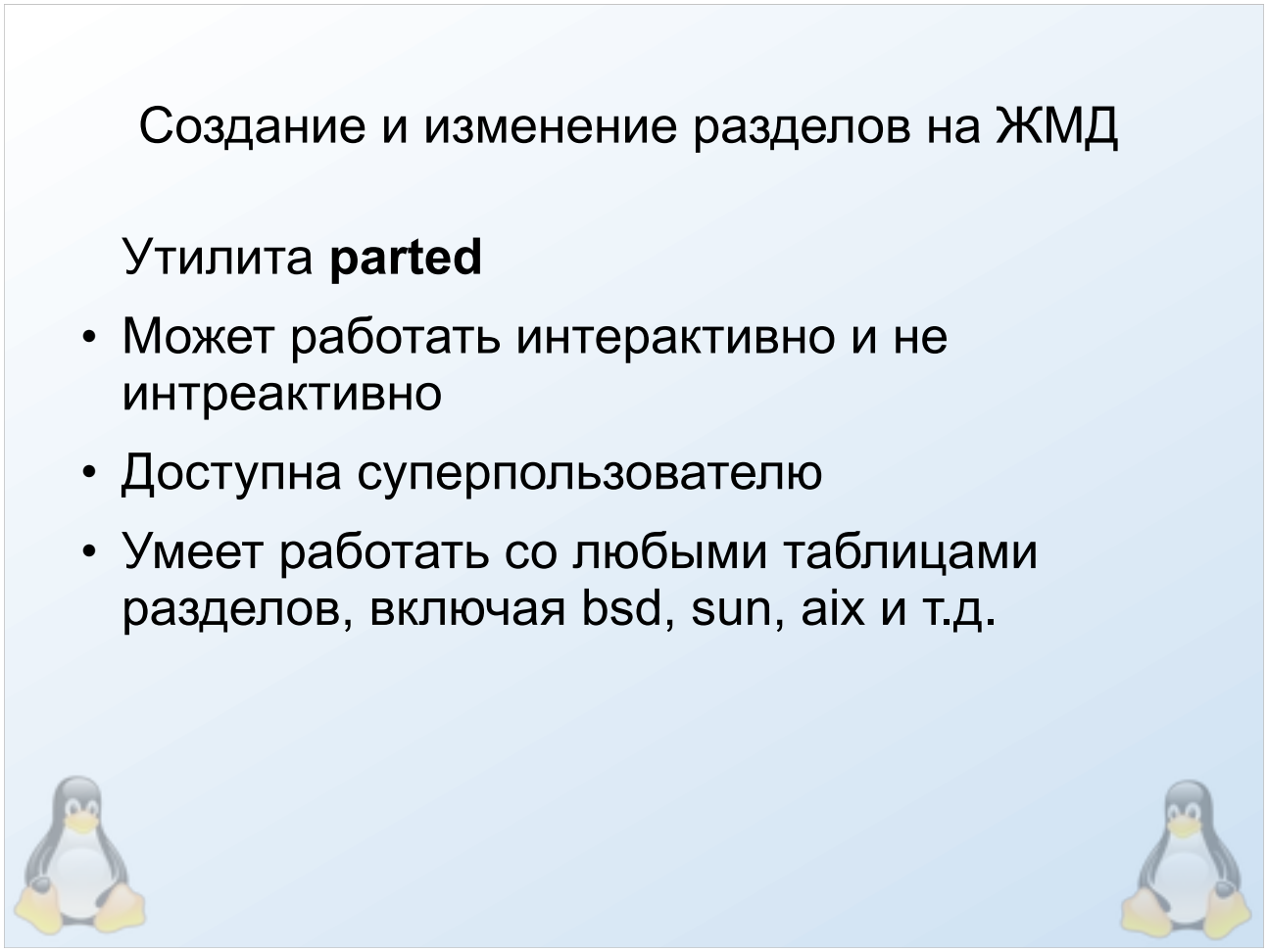
   13. -uS or -uB or -uC or -uM – устанавливает единицы измерения соответственно в секторах, блоках, цилиндрах или мегабайтах (по умолчанию в цилиндрах);
   14. -x or --show-extended – указывает работать с расширенными разделами;
   15. -C cylinders – задает число цилиндров иное чем в ядре;
   16. -H heads – задает число головок иное чем в ядре;
   17. -S sectors – задает число головок иное чем в ядре;
   18. -f or --force – обязательное выполнение команды;

19. `-q or --quiet` – работать без предупреждений;
20. `-L or --Linux` – не сообщать о действиях несовместимых с Linux;
21. `-D or --DOS – DOS` совместимый режим;
22. `-E or --DOS-extended` – создает DOS совместимый режим для расширенного раздела;

Примечание: В некоторых версиях DOS номер сектора "внутри" раздела назначается относительно границы стартового цилиндра. В Linux номер сектора назначается относительно стартового сектора.

23. `--IBM or --leave-last` – не использовать последний цилиндр для тестовых программ IBM;
24. `-n` – проделать все действия, но не записывать изменения на диск;
25. `-R` – производит проверку устройства, если тест неудачен, то возможно необходимо произвести размонтирование занятых устройств;
26. `--no-reread` – подавляет тест смонтированных устройств на данном носителе;
27. `-O file` – непосредственно перед записью нового раздела вывести сектора подлежащие перезаписи в файл;
28. `-I file` – если была использована некорректная команда sfdisk, позволяет восстановить испорченные секторы.

1.1.3. Утилита parted



1. `parted` может быть использована для создания, уничтожения, изменения, перемещения и копирования разделов ext2, ext3, linux-swap, FAT и FAT32.

Примечание: Некоторые возможности доступны только для ФС ext2.

2. `parted` имеет следующий синтаксис:

   ```
   parted [options] [device [command [options...]...]]
   ```

3. `parted` работает в интерактивном и не интерактивном режимах.
4. Не интерактивный режим удобно использовать для скриптов. В этом случае удобно использовать опцию `-s`, которая подавляет вывод вопросов пользователю.
5. Опция `-i` используется для вывода запросов по мере необходимости.
6. В `parted` можно использовать следущие команды:
   1. `help [command]` – основная помощь по командам;
   2. `mklabel label-type` – установить метку диска типа label-type. Типы могут быть: "aix", "amiga", "bsd", "dvh", "gpt", "loop", "mac", "msdos", "pc98", или "sun".
   3. `mkpart [part-type name fs-type] start end` – создает раздел. Тип раздела fs-type принимает значения "primary", "logical" или "extended". Начало и конец указываются в мегабайтах.

4. `rescue start end` – поискать потерянные разделы в указанных границах.
5. `name partition name` – задать имя разделу. Только для меток Mac, PC98 или GPT.
6. `print` – вывести таблицу разделов.
7. `quit` - выход
8. `resizepart partition end` — изменить позицию конца раздела.
9. `rm partition` – удалить указанный раздел.
10. `select device` – выбрать устройство.
11. `set partition flag state` -установить флаг состояния раздела. Флаги принимают значения "on" или "off". Поддерживаемые флаги: "boot", "root", "swap", "hidden", "raid", "lvm" и "lba".

1.2. Файловые системы ext2/3/4

1.2.1. Архитектура ext2

1. Расположение файловой системы на диске представлено на рисунке 4.1.
2. Непосредственно за загрузочным блоком файловая система разбита на части фиксированного размера, которые называются блоковые группы (block group).
3. Каждая группа блоков управляет фиксированным набором индексных дескрипторов и блоков данных.

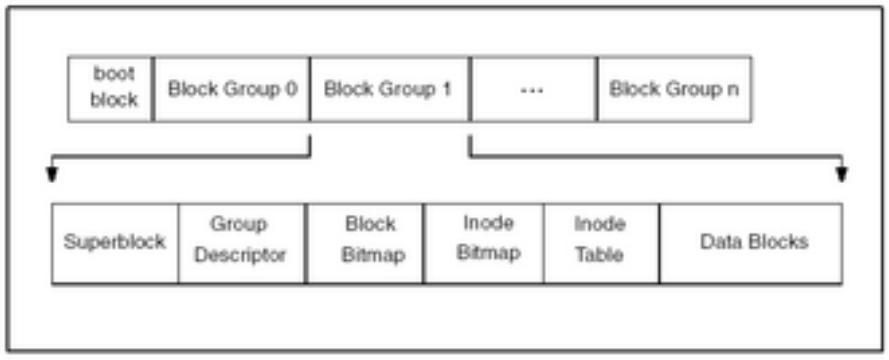
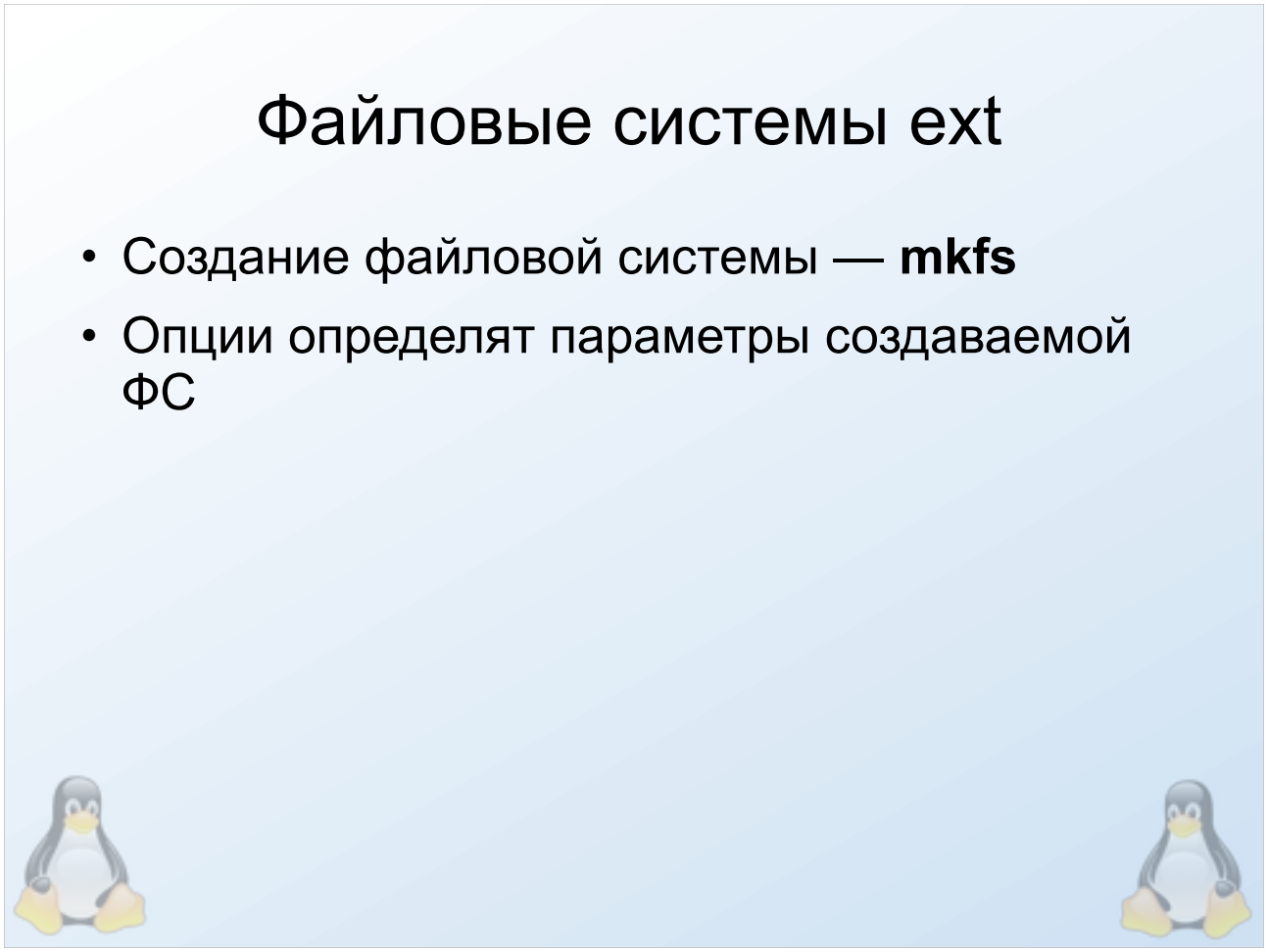


Рис 4.1. Расположение ext2 на диске.

4. Каждая группа блоков содержит:
    1. копию суперблока;
    2. дескриптор группы;
    3. карту блоков;
    4. карту inode;
    5. таблицу inode;
    6. блоки данных.
5. Суперблок содержит основную информацию, необходимую для монтирования и работы файловой системы:
    1. тип файловой системы;
    2. размер, расположение и количество блоков в файловой системе;
    3. количество индексных дескрипторов;
    4. время последнего монтирования;
    5. информация о том была ли размонтирована файловая система;
    6. счетчик числа монтирований;
    7. число свободных блоков и индексных дескрипторов;
    8. время последнего монтирования и записи;
    9. время следующей проверки;
    10. интервал проверок;
    11. UID и GID пользователя, для которого резервируется место на разделе.
6. При обслуживании файловых систем ext применяются следующие утилиты:
    1. `mke2fs` - для инициализации (создания) файловой системы ext на заданном устройстве;
    2. `badblocks` - для проверки наличия плохих блоков на носителе данных;
    3. `e2fsck` - для проверки целостности файловой системы;
    4. `tune2fs` - для настройки параметров файловых систем ext;
    5. `dumpe2fs` - для печати суперблока и блоков данных;

6. `debugfs` - интерактивная утилита, применяющаяся для восстановления и исправления файловых систем ext.

1.2.2. Создание файловой системы mke2fs

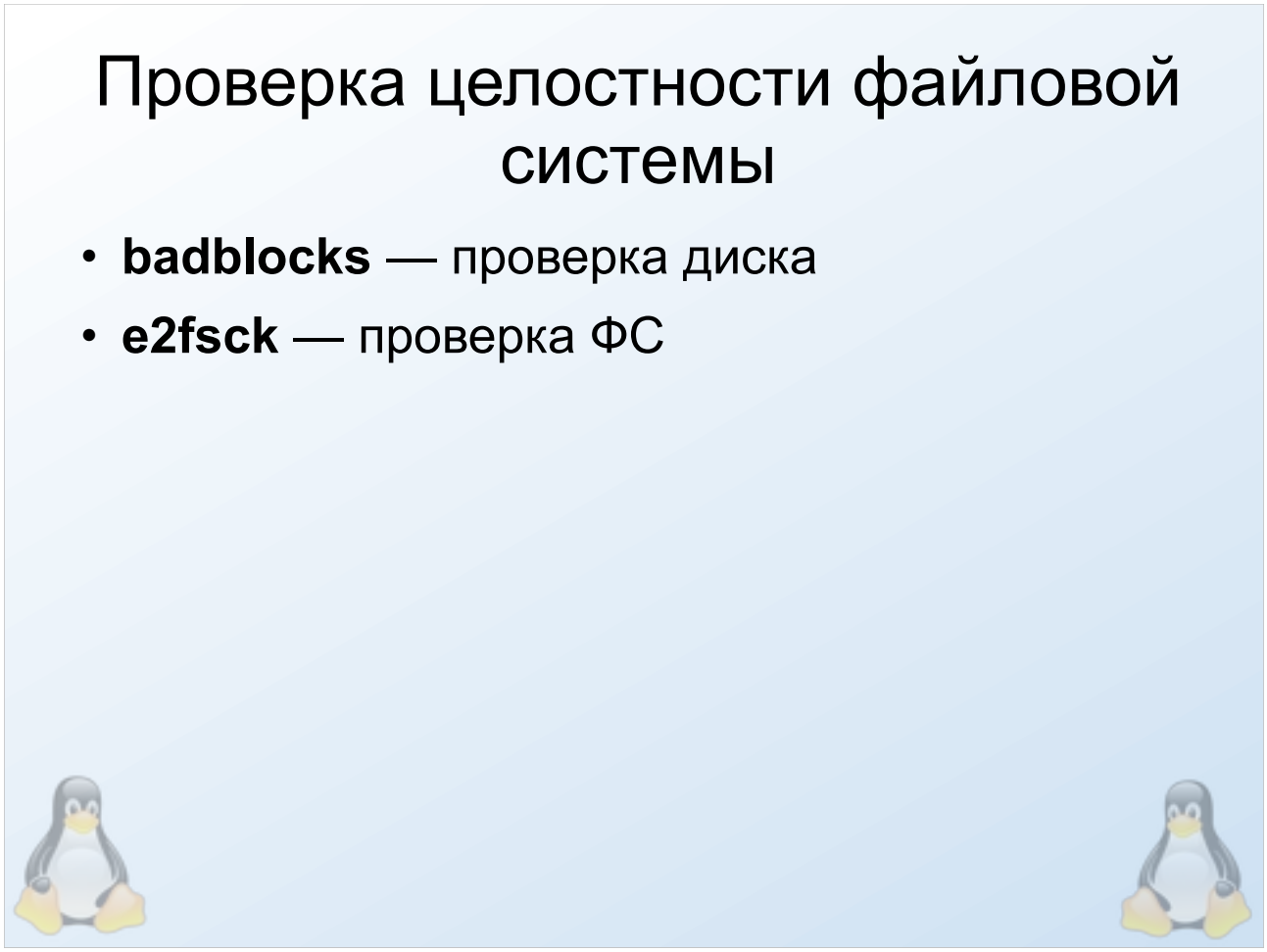


1. Для создания файловых систем ext можно использовать утилиту `mkfs`, являющуюся оболочкой для различных утилит построения файловых систем.
2. Файловые системы ext2, ext3 и ext4 создаются с помощью утилит `mke2fs, mkfs.ext2`, `mkfs.ext3` или `mkfs.ext4`.
3. Три последние - это обычно ссылки на `mke2fs`.
4. В качестве аргумента `mke2fs` принимает имя блочного файла устройства.
5. Основные опции команды `mke2fs`:
   1. `-b` - размер блока (1024, 2048 или 4096 байт);
   2. `-c` - использование программы `badblocks` для проверки наличия плохих блоков;
   3. `-i` - позволяет задать количество байт на один inode;
   4. `-N`- позволяет задать количество inode;
   5. `-j` - создает журналируемую файловую систему ext3;
   6. `-J size` - размер журнала (от 1024 до 102400 блоков);
   7. `-J device` - имя устройства внешнего хранения журнала, на котором должна быть

предварительно создана специальная файловая система с помощью команды `mke2fs -O journal_dev /dev/...`;

8. `-l` - чтение списка плохих блоков из файла;
9. `-L` - установка LABEL (LABEL удобно использовать при монтировании и в файле /etc/fstab);
10. `-m` - определяет процент резервирования пространства в файловой системе для администратора (по умолчанию 5%);
11. `-M` - устанавливает каталог "последнего монтирования" (используется некоторыми утилитами для монтирования файловой системы);
12. `-n` - симуляция создания файловой системы вместо ее настоящего создания;
13. `-S` - реинициализация суперблока вместо создания файловой системы. Может быть использована в крайнем случае. После выполнения этой операции должна быть вызвана утилита `e2fsck`;
14. `-T` - "тип" создаваемой файловой системы:
    1. `news` - один индексный дескриптор inode на 4К,
    2. `largefile` - один inode на 1М,
    3. `largefile4` - один inode на 4М.

1.2.3. Проверка целостности файловой системы



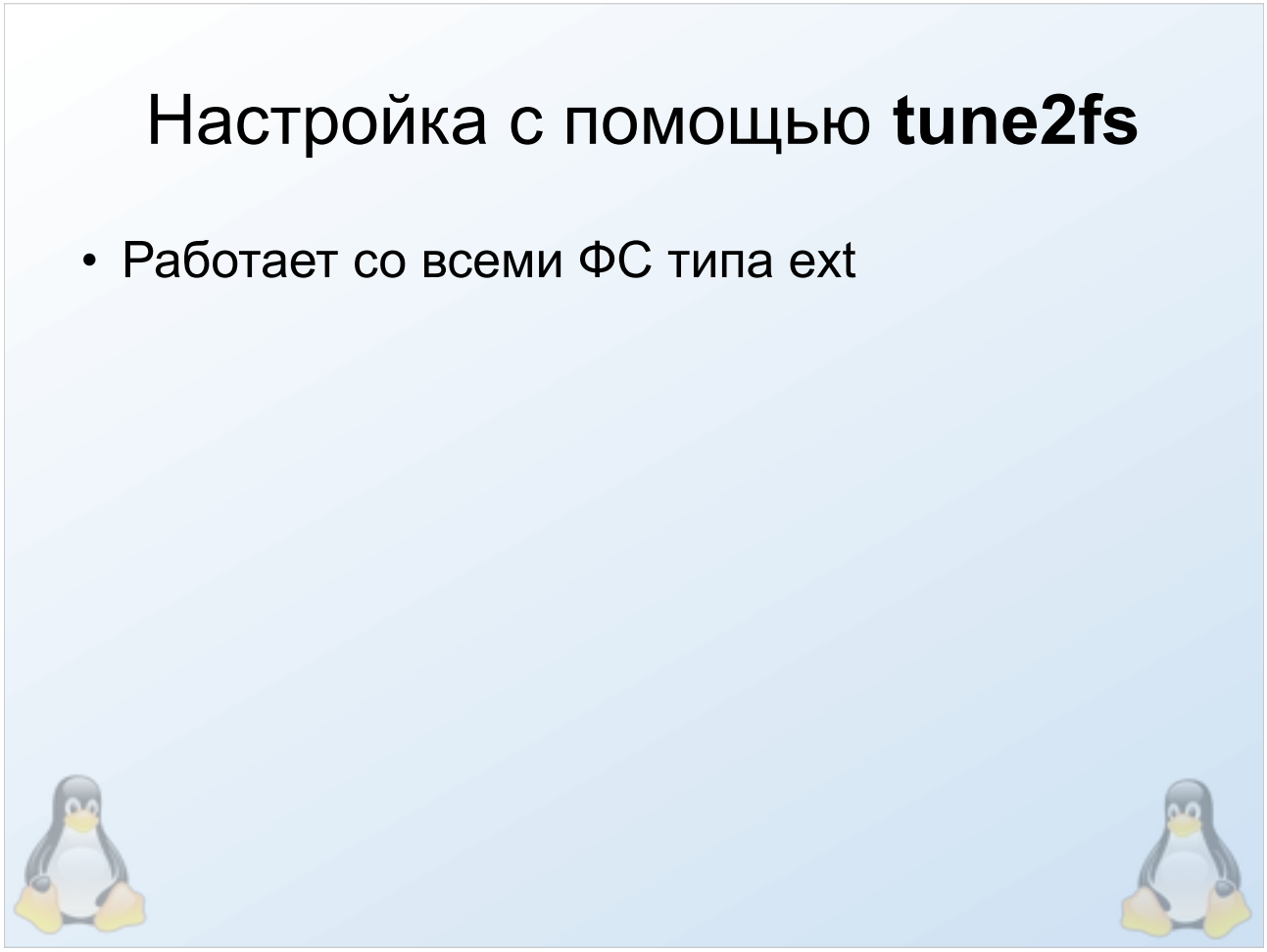
1. Утилита `badblocks` предназначена для поиска плохих блоков на заданном в качестве аргумента устройстве, обычно являющимся разделом жесткого диска.
2. При работе с этой утилитой можно указать начальный и конечный блоки для тестирования.
3. Обычно эту утилиту не используют непосредственно в связи с возможностью неверного указания количества блоков и их размера.
4. Вместо непосредственного вызова утилиты `badblocks` используется ее неявный вызов с помощью опции `-c` команд `e2fsck` и `mke2fs`.
5. Основные опции команды `badblocks`:
    1. `-b` - размер блока (1024, 2048 или 4096 байт);
    2. `-c` - количество блоков, тестируемых за один цикл (по умолчанию - 16);
    3. `-i` - входной файл, содержащий список плохих блоков. Эти блоки не будут тестироваться;
    4. `-o` - выходной файл, содержащий список найденных плохих блоков;
    5. `-n` - запуск программы в неразрушающем режиме чтения-записи (rw). По умолчанию

программа использует неразрушающий режим только для чтения (ro);

6. `-w` - запуск программы в режиме чтения-записи (rw) С РАЗРУШЕНИЕМ ДАННЫХ;
7. `-s` - отображение процесса работы программы;
8. `-v` - режим вывода подробной информации.

6. Команда `e2fsck` позволяет проверять целостность файловых систем ext.
7. В качестве аргумента для этой команды задается имя блочного файла устройства с проверяемой файловой системой.
8. Основные опции команды `e2fsck`:
   1. `-b` - номер блока на блочном устройстве, содержащего суперблок. Для файловых систем ext2 и ext3 с размерами блоков 1К, 2К и 4К номера блоков, содержащих первые копии суперблока, соответственно, 8193, 16384 и 32768. Номера других копий суперблока могут быть получены с помощью команды `mke2fs -n`;
   2. `-B` - указывает размер используемых блоков данных;
   3. `-c` - использование программы badblocks для проверки наличия плохих блоков;
   4. `-f` - заставляет производить проверку, даже если файловая система выглядит исправной;
   5. `-l` - добавляет список плохих блоков из файла;
   6. `-L` - считывает список плохих блоков из файла;
   7. `-n` - открывает файловую систему в режиме только для чтения (ro), а также предусматривает автоматический ответ NO на все вопросы;
   8. `-y` - автоматический ответ yes на все вопросы;
   9. `-p` - автоматическое восстановление системы без выдачи вопросов;
   10. `-t` - подсчет статистики по времени выполнения процедур;
   11. `-v` - режим вывода подробной информации.

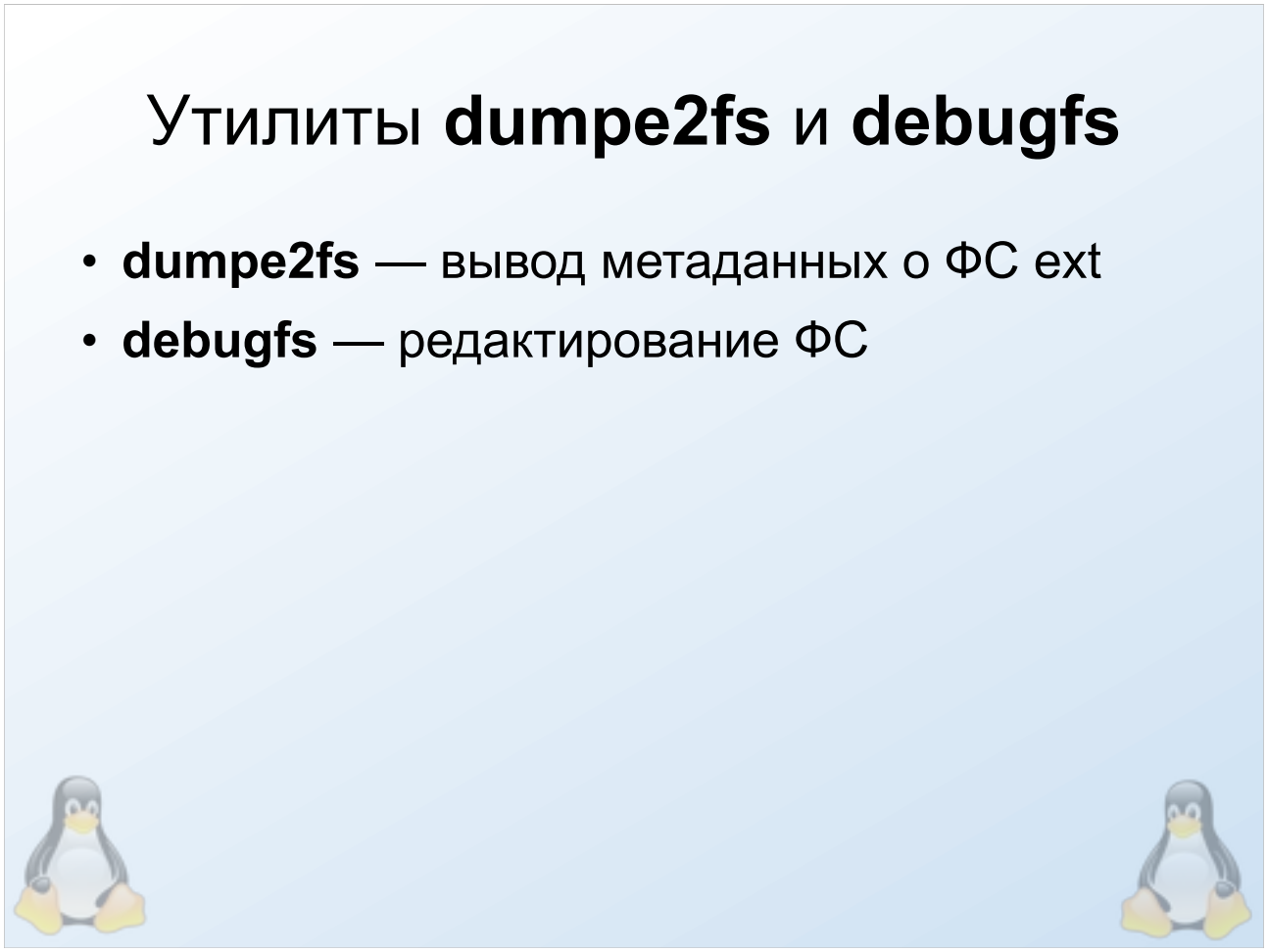
1.2.4. Настройка с помощью tune2fs



1. Утилита `tune2fs` позволяет изменять настраиваемые параметры файловых систем ext.
2. В качестве аргумента принимает имя блочного файла устройства настраиваемой файловой системы.
3. Основные опции команды `tune2fs`:
    1. `-c` - максимальное количество монтирования файловой системы между проверками ее целостности;
    2. `-C` - установка количества раз сколько была смонтирована файловая система;
    3. `-e`
        1. `continue` - при наличии ошибки в файловой системе продолжать нормальную работу;
        2. `remount-ro` - при наличии ошибки в файловой системе монтировать систему в режиме только для чтения;
        3. `panic` - при наличии ошибки в файловой системе вызывать панику ядра;
    4. `-i` - установка периода времени между проверками. Может содержать символы d, m и w для указания количества дней, месяцев или недель;
    5. `-j` - установить журнал в файловую систему;
    6. `-l` - распечатать содержимое суперблока;

7. `-L` - установить метку тома;
8. `-m` - определяет процент резервирования пространства в файловой системе для администратора (по умолчанию 5%);
9. `-M` - установка каталога "последнего монтирования";
10. `-T` - установка времени последней проверки.
11. `-O` – установить/снять опции ФС. Используется в т.ч. для преобразования между типами.

1.2.5. Утилиты dumpe2fs и debugfs

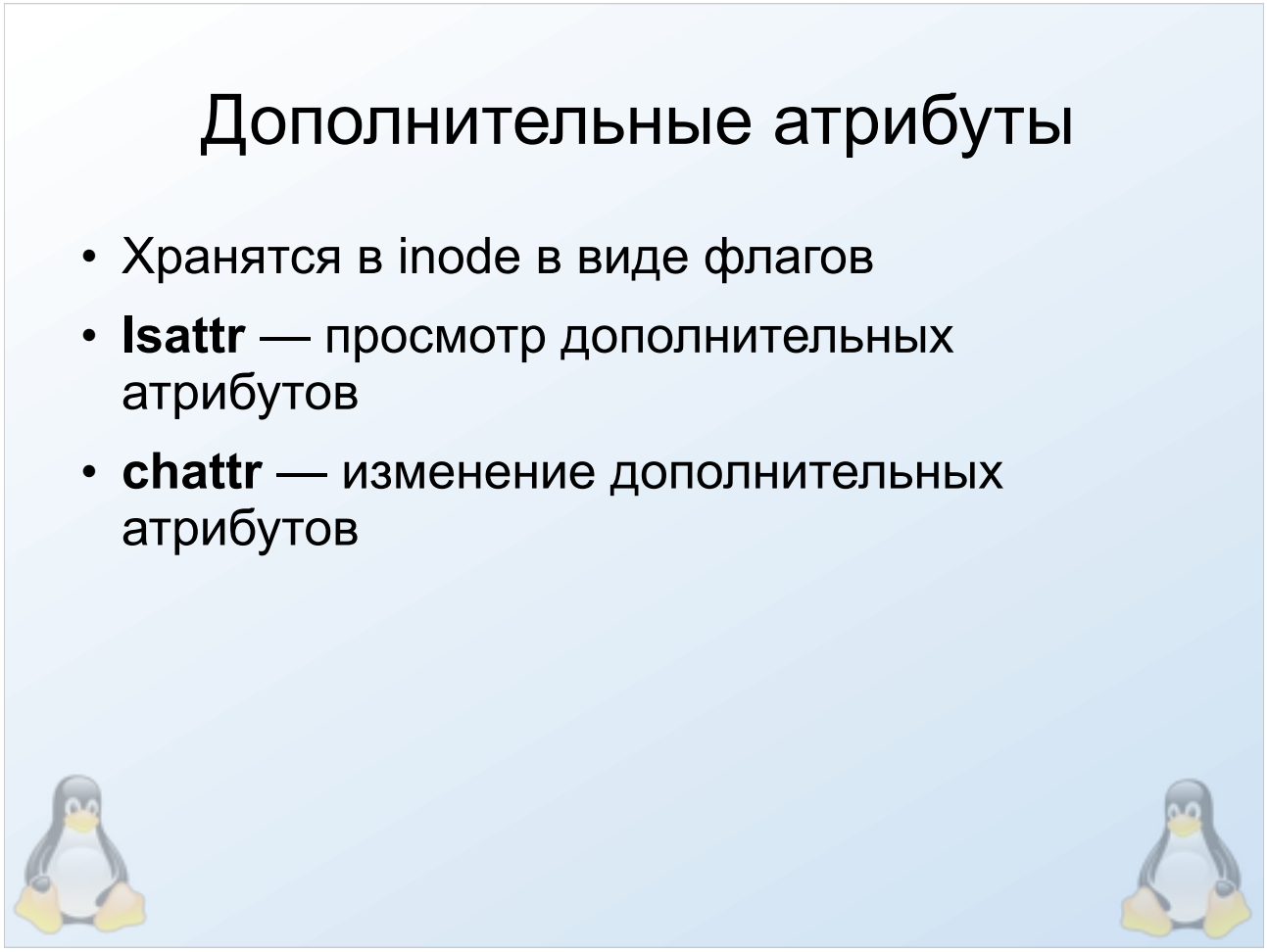


1. Утилита `dumpe2fs` печатает суперблок файловой системы заданного блочного устройства, а также информацию о группах блоков.
2. Основные опции команды:
   1. `-b` - вынуждает печатать плохие блоки;
   2. `-ob` - номер блока суперблока;
   3. `-oB` - устанавливает размер блока;
   4. `-f` - заставляет программу печатать информацию о файловой системе даже если встречаются неизвестные программе данные;

5. `-h` - выводить только суперблок;
6. `-i` - отображать данные о файловой системе из файла образа, созданного программой `e2image`;
7. `-x` - отображать данные в шестнадцатеричном формате.

3. Интерактивная утилита `debugfs` позволяет редактировать содержимое файловой системы. Основные опции:
   1. `-w` - открыть файловую систему для чтения и записи (по умолчанию только для чтения);
   2. `-c` - режим катастрофы, в котором массив метаданных и битовая матрица групп блоков не читаются;
   3. `-i` - использовать данные о файловой системе из файла образа, созданного программой e2image;
   4. `-b` - использовать заданный размер блока;
   5. `-s` - указывает номер блока суперблока (используется только вместе с `-b`);
   6. `-f` - чтение команд интерактивного режима команды из заданного файла;
   7. `-R` - выполнение заданного запроса.

1.2.6. Дополнительные атрибуты



1. Файловая система ext поддерживает несколько дополнительных атрибутов.
2. По умолчанию эти атрибуты при создании нового файла не устанавливаются, и их активизация производится вручную.
3. Атрибуты хранятся в блоке информационного дескриптора (inode) под заголовком flag (флаг) в виде шестнадцатеричного числа:
    1. **«A»** отключает обновление поля времени последнего доступа к файлу (поле atime), что позволяет снизить нагрузку на жесткий диск.
    2. **«S»** предписывает операционной системе асинхронно сохранять на диске все модификации файла.
    3. **«a»** запрещает выполнять над файлом какие-либо операции, кроме добавления данных. Установить может только суперпользователь.
    4. **«c»** включает сжатие файла. При записи файла на диск такой файл автоматически сжимается, а при чтении разжимается.
    5. **«d»** используется программой `dump`, которая игнорирует файлы с этим атрибутом и не осуществляет их резервное копирование.
    6. **«i»** блокирует любую возможность изменения файла. Файлы, обладающие таким атрибутом, являются неизменяемыми (immutable), их нельзя удалить, изменить или переименовать, кроме того, на него нельзя сослаться при помощи ссылки. Все эти действия блокируются не только для обычных пользователей, но даже и для пользователя root. Только суперпользователь может установить этот атрибут.

7. **«s»** предписывает ядру сопровождать удаление файла записью нулей в дисковые блоки, принадлежащие этому файлу.
8. **«u»** предназначен для восстановления файла после того, как файл удален обычным образом.

Примечание: не следует слишком полагаться на этот атрибут, так как зачастую восстановить удается лишь первый блок файла (обычно это 4096 байт). Если вы хотите обеспечить возможность восстановления удаляемых файлов, более правильным решением будет не удалять файл сразу, а перемещать его в специальный каталог, где он будет находиться некоторое время, по истечении которого cron удалит его по-настоящему.

4. Для настройки перечисленных ранее атрибутов используется команда `chattr`.
5. Как и для команды `chmod`, требуемая операция указывается при помощи оператора, за которым следуют символьные обозначения атрибутов.
   1. «+» предписывает команде установить указанные атрибуты;
   2. «–» – сбросить атрибуты;
   3. «=» – установить указанные атрибуты и сбросить все остальные.
6. Параметр `-R`, обозначает, что команда будет применена рекурсивно в отношении всех подкаталогов.
7. Параметр `-V`, обозначает вывод версии программы `chattr`, а также дополнительных сообщений во время ее работы.
8. Параметр `-v`**.** При использовании этого параметра после него следует указать число, которое будет установлено в качестве номера версии индексного дескриптора.

Примечание: Никакого смысла, кроме того, который вы сами ему дадите, в номере версии нет — это просто число, которое можно записать в индексный дескриптор. Это число никак не связано с самим файлом. При создании файла ему выделяется индексный дескриптор из числа свободных, номер версии этого индексного дескриптора устанавливается равным единице.

Пример: Если выполнить команду mv foo foo2, то файл foo2 унаследует свой индексный дескриптор от файла foo. Это означает, что если файл foo2 до этого существовал, его индексный дескриптор (а значит, и номер версии) будет заменен. Однако если при существующем foo2 выполнить cp foo foo2, индексный дескриптор foo2 останется прежним (если файл foo2 до этого не существовал, ему будет выделен свободный индексный дескриптор с номером версии, равным 1). К примеру Caldera использует эти номера при установке системы, давая всем устанавливаем файлам уникальный номер версии. Делается это для того, чтобы файлы, созданные после первоначальной установки, можно было отличить от файлов, созданных во время установки. Например, если обновить пакет (`rpm -U`), то номер версии новых файлов будет равен не заданному во время установки уникальному номеру, а простой единице.

9. Команда `lsattr` выводит список файлов и их ext2-атрибутов.
10. Подобно команде `chattr`, команда `lsattr` поддерживает параметр `-R`, при наличии которого она рекурсивно обрабатывает все подкаталоги.
11. Параметр `-a` означает отображение информации обо всех файлах, включая файлы, имена которых начинаются с точки.
12. При использовании опции `-d` выводятся сведения только о каталогах, но не о файлах.

13. Параметр `-l` позволяет получить сведения о файлах в расширенном формате, где каждый атрибут отображается не с помощью буквы, а с помощью слов.
14. Параметр `-v` включает вывод версии файлов.

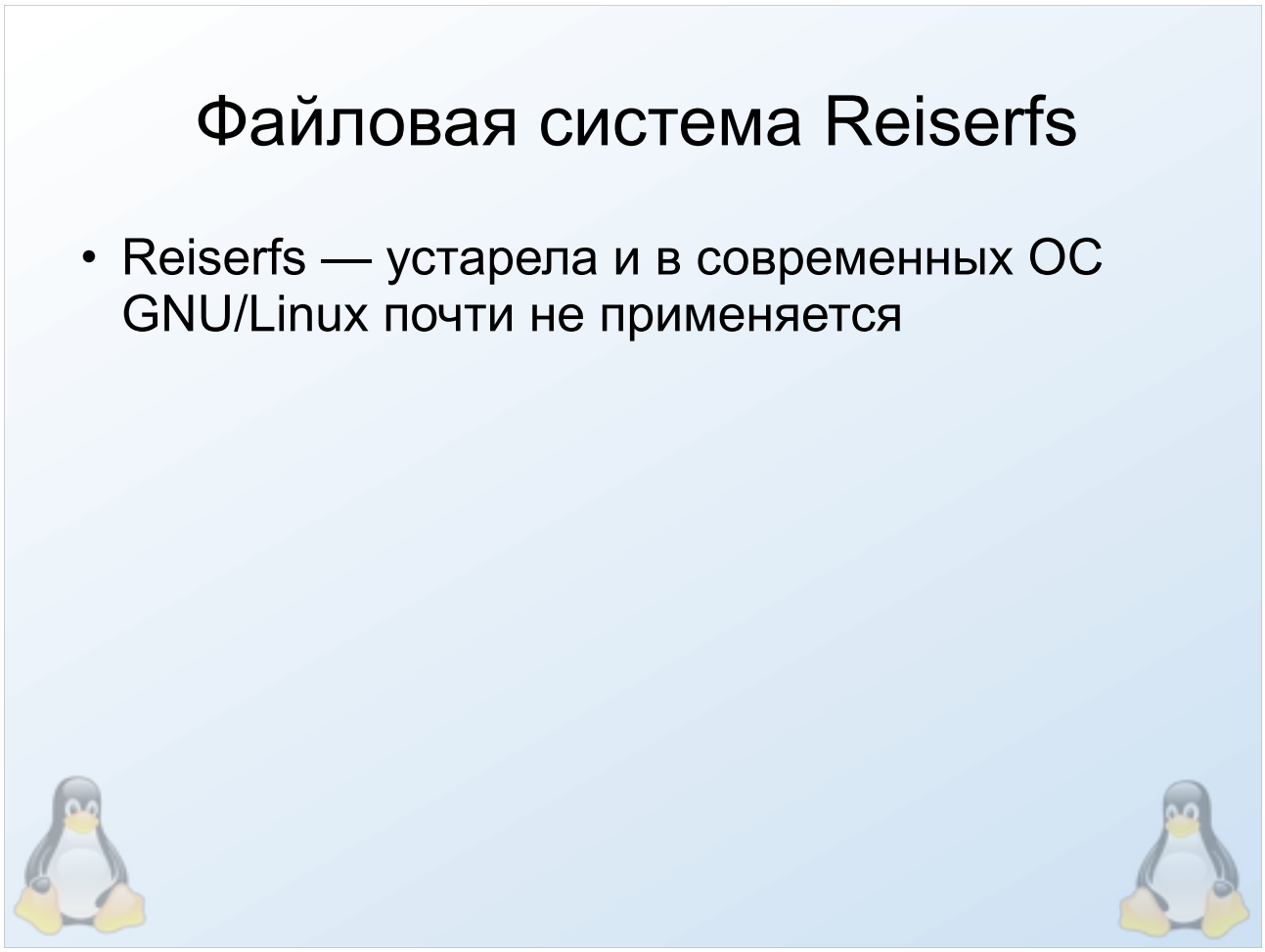
Пример: установка и проверка дополнительных атрибутов

```
# ls
file.txt
# chattr +i file.txt
# lsattr
----i-------- ./file.txt
# rm -rf file.txt
rm: cannot remove `file.txt': Operation not permitted
```

Примечание: обратите внимение после установки атрибута immutable (i) root не смог удалить этот файл.

1.3. Файловая система Reiserfs

1.3.1. Архитектура файловой системы



1. Проект Reiserfs стартовал в 90-х годах, первый прототип носил название TreeFS.
2. Разработана Хансом Райзером (Hans Reiser) и его компанией Namesys (http://www.namesys.com).

Примечание: В связи с тюремным заключением Ханса Рейзера, работа над проэктом, на данный момент, остановлена.

3. Разработчики системы работают над созданием не только файловой системы, но вообще механизма иерархического именования объектов.

Примечание: Они считают, что лучшая файловая система та, которая формирует единую общедоступную среду— namespace. Для этого файловая система должна выполнять часть работы, традиционно выполнявшуюся приложениями, что уменьшит количество несовместимых API узкоспециального назначения. При таком подходе пользователи смогут продолжать прямое использование файловой системы без необходимости формировать уровни специального назначения, типа баз данных и т.п.

4. Раздел ReiserFS разделен на блоки фиксированной длины, которые нумеруются начиная с 0.

5. В одном разделе может быть максимально 2^32 блоков.
6. Суперблок начинается после первых 64 кб, которые оставляются для меток раздела и загрузчиков.
7. Суперблок содержит следующую информацию:
    1. размер блоков;
    2. число блоков;
    3. корневой нод («начало» файловой системы для приложений);
    4. журнальные ноды.
8. Размер суперблока зависит от размера блоков, но всегда начинается с 65536 байта.
9. Размер блока по умолчанию 4096 б.
10. На файловую систему имеется только один суперблок.
11. Непосредственно за суперблоком располагается блок содержащий битовую карту (bitmap) свободных блоков.
12. Число блоков, отображенных в карте, напрямую зависит от размера блоков.
13. Файловая система составляется из сбалансированных деревьев (balanced tree).
14. Дерево составляется из внутренних и листовых (leaf) нодов.
15. Каждый нод это дисковый блок.
16. Каждому объекту (элементу, item) в reiserfs назначается уникальный ключ, который может быть связан с номером inode.
17. Внутренние элементы составляются из ключей и указателей на дочерние ноды.
18. Каждый нод имеет уровень от 1 (для листовых объектов) и выше.
19. У корневого нода наивысший уровень.
20. Листовые ноды состоят из заголовков элементов и собственно элементов.
21. Если элемент не вмещается в один блок, то в элементах содержатся указатели на неформатированные блоки.
22. С другой стороны, если элементы маленькие, то они могут располагаться в одном блоке.
23. Использование сбалансированных деревьев дает увеличение производительности, а также снимает ряд искусственных ограничений на размещение файловой системы.

Пример: становится возможным иметь каталог, содержащий 100,000 других подкаталогов.

24. Еще одна выгода от использования деревьев в том, что ReiserFS, подобно большинству других файловых систем нового поколения, динамически создает inodes вместо их статического набора, образующегося при создании "традиционной" файловой системы. Это дает большую гибкость и оптимальность в формировании пространства хранения.
25. ReiserFS также имеет ряд возможностей, нацеленных специально для улучшения работы с маленькими файлами.
26. ReiserFS не связана ограничением в выделении памяти для файла в целом числе 1-2-4 KB блоков. По необходимости для файла может выделяться точный размер.
27. ReiserFS также включает некоторые виды специальной оптимизации файловых "хвостов"

для хранения конечных частей файлов, меньших, чем блок файловой системы. Для увеличения скорости,

28. ReiserFS способен хранить содержимое файлов непосредственно внутри b*tree, а не в виде указателя на дисковый блок. Тем самым достигается две вещи:
    1. Увеличивается производительность, так как data и stat_data (inode) информация может хранится рядом и считываться одной дисковой операцией IO.
    2. Второе, ReiserFS способен упаковать хвосты, экономя дисковое пространство. Фактически, при разрешении ReiserFS выполнять упаковку хвостов (значение по умолчанию) будет экономиться примерно шесть процентов дискового пространства (в сравнении с ext2).

Пример: в RaiserFS есть понятие fastlink, когда содержимое "мягкой" ссылки до 60 байт хранится в inode.

Примечание: Следует иметь в виду, что упаковка хвостов требует дополнительной работы, так как при изменении размеров файлов необходима "переупаковка". По этой причине в ReiserFS упаковка хвоста может отключаться, позволяя администратору выбрать между скоростью и эффективностью использования дискового пространства

1.3.2. Создание файловой системы Reiserfs

1. Команда `mkreiserfs` позволяет создавать файловую систему `reiserfs`.
2. Синтаксис команды `mkreiserfs`:

```
mkreiserfs [ options ] device [ filesystem-size ]
```

`filesystem-size` задает размер файловой системы в блоках.

3. Опции команды `mkreiserfs`:
    1. `-b | --block-size N` – размер блока в байтах. Доступно только 4096.
    2. `-h` – позволяет указать тип хеш функции, используемой для индексирования:
        1. `-h rupasov` - использование алгоритма Ю.Рупасова;
        2. `-h tea` - использование функции Дэвиса-Мейера;
        3. `-h r5` - использование модифицированного алгоритма Ю.Рупасова;
    3. `--format FORMAT` – формат файловой системы.
        1. 3.6 для ядер 2.4 и выше;
        2. 3.5 для ядер 2.2
    4. `-u | --uuid UUID` – задает универсальный уникальный идентификатор.

        Пример:

```
c1b9d5a2-f162-11cf-9ece-0020afc76f16
```

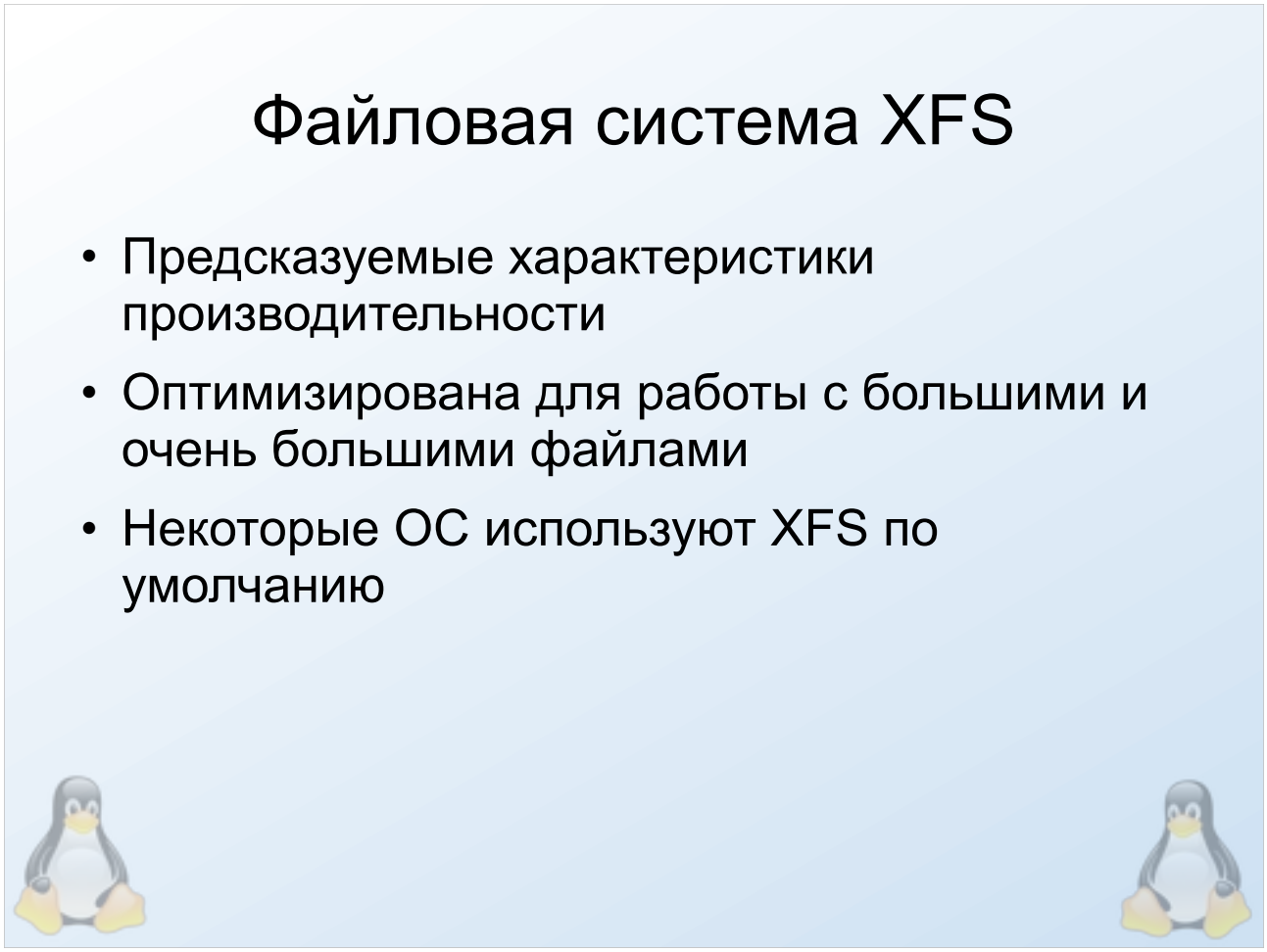
    5. `-l | --label LABEL` – устанавливает метку тома длинной до 16 символов.
    6. `-q | --quiet` – создает файловую систему без дополнительных сообщений и вопросов.

7. `-j | --journal-device FILE` – имя блочного устройства в котором хранится журнал.
8. `-o | --journal-offset N` – отступ начала журнала на указанном в опции -j устройстве. По умолчанию 0.
9. `-s | --journal-size N` – размер журнала.
10. `-t | --transaction-max-size N` – максимальный размер транзакции.
11. `-B | --badblocks file` – имя файла содержащего список плохих блоков.
12. `-f` – безусловное выполнение действий.
13. `-d` – выводить отладочную информацию.
14. `-V` – вывести версию.

1.3.3. Проверка целостности файловой системы reiserfs

1. Утилита `debugreiserfs` позволяет исправлять некоторые проблемы, возникающие при работе с файловой системой reiserfs.
2. При вызове без опций печатает содержимое суперблока файловой системы.
3. Основные опции команды `debugreiserfs`:
   1. `-j` - печать журнала;
   2. `-d` - форматированная печать узлов файловой системы;
   3. `-c` - печать связок;
   4. `-b` - печать заданного блока;
   5. `-p` - извлечение массива метаданных.
4. Утилита `reiserfsck` предназначена для проверки целостности файловой системы reiserfs.
5. Основные опции `reiserfsck`:
   1. `--check` – проверка целостности файловой системы (по умолчанию);
   2. `--rebuild_tree` – перестройка двоичного дерева файловой системы;
   3. `--rebuild_sb` – восстановление суперблока;
   4. `--no-journal-available` – позволяет работать, когда устройство с журналом не доступно.
6. Утилита `reiserfstune` позволяет настраивать журнал файловой системы.

1.4. Файловая система XFS



1. Основа этой файловой системы была создана в начале 90-х (1992—1993) фирмой Silicon Graphics (сейчас SGI) для мультимедийных компьютеров с ОС Irix, заменив уже не удовлетворявшую требованиям времени EFS, но версия 1.0 стала доступна только первого мая 2001.
2. В настоящий момент активно используется и продвигается компанией Red Hat.
3. Файловая система ориентирована на очень большие файлы (9 тыс. Петабайт — 9 млн. Терабайт — 1018 байт) и файловые системы (18 тыс. Петабайт)
4. Она является полностью 64-битной, что позволяет адресовать большие массивы данных.
5. Особенностью этой файловой системы является устройство журнала — в журнал пишется часть метаданных самой файловой системы таким образом, что весь процесс восстановления сводится к копированию этих данных из журнала в файловую систему.
6. Размер журнала задается при создании системы, он должен быть не меньше 32 мегабайт.
7. Тесты на производительность показывают преимущество XFS, при работе с большими и средними файлами.
8. Также эту файловую систему характеризует прямолинейность падения производительности при увеличении нагрузки и предсказуемость.

9. Она не генерирует излишнюю дисковую активность, т.к. пытается кэшировать как можно больше данных и «основанием» для сброса на диск является заполнение памяти, а не интервал времени, как это принято в других ФС.
10. Любое дисковое устройство при создании файловой системы XFS разбивается на несколько равных по размеру линейных областей (0.5—4 Гб), в терминологии XFS они именуются *allocation group*.
11. Уникальность *allocation group* в том, что каждая группа управляет своими собственными inodes и свободным местом, что превращает группы в своего рода автономные файловые системы, сосуществующие в рамках общей XFS.
12. Такая организация позволяет эффективно организовать параллельную обработку операций ввода/вывода, которая особенно ярко проявляется на многопроцессорных системах.
13. В каждой такой группе используется три B+-дерева, два из которых отвечают за свободные inodes (allocation).
14. В этой системе реализована возможность, позволяющая избежать фрагментации файлов, называемая *delayed allocation*.

Примечание: Файловая система, получая данные для записи, по началу лишь резервирует под них необходимое свободное место, откладывая саму запись до момента фактического сброса данных. Когда же такой момент наступает, XFS решает, куда необходимо их поместить. Если осуществляется дозапись, то подбираются соседние сектора. Но наибольший эффект от такой задержки получается еще и за счет того, что при создании временного файла с малым временем жизни последний вообще на диск не пишется (соответственно, не приходится занимать/освобождать метаданные).

15. Установив необходимые утилиты, можно создать файловую систему:

Пример:

```
#/sbin/mkfs.xfs /dev/hdb2 или mkfs -t xfs /dev/hdb2
```

16. Для увеличения производительности в некоторых случаях может помочь опция `-l size=32m`, фиксирующая размер журнала (32 Мб).
17. С помощью `-d agcount=x` можно установить минимально возможное количество allocation groups (т.е. взяв максимально возможные 4 Гб на группу).

Пример: при разделе 18 Гб устанавливаем:

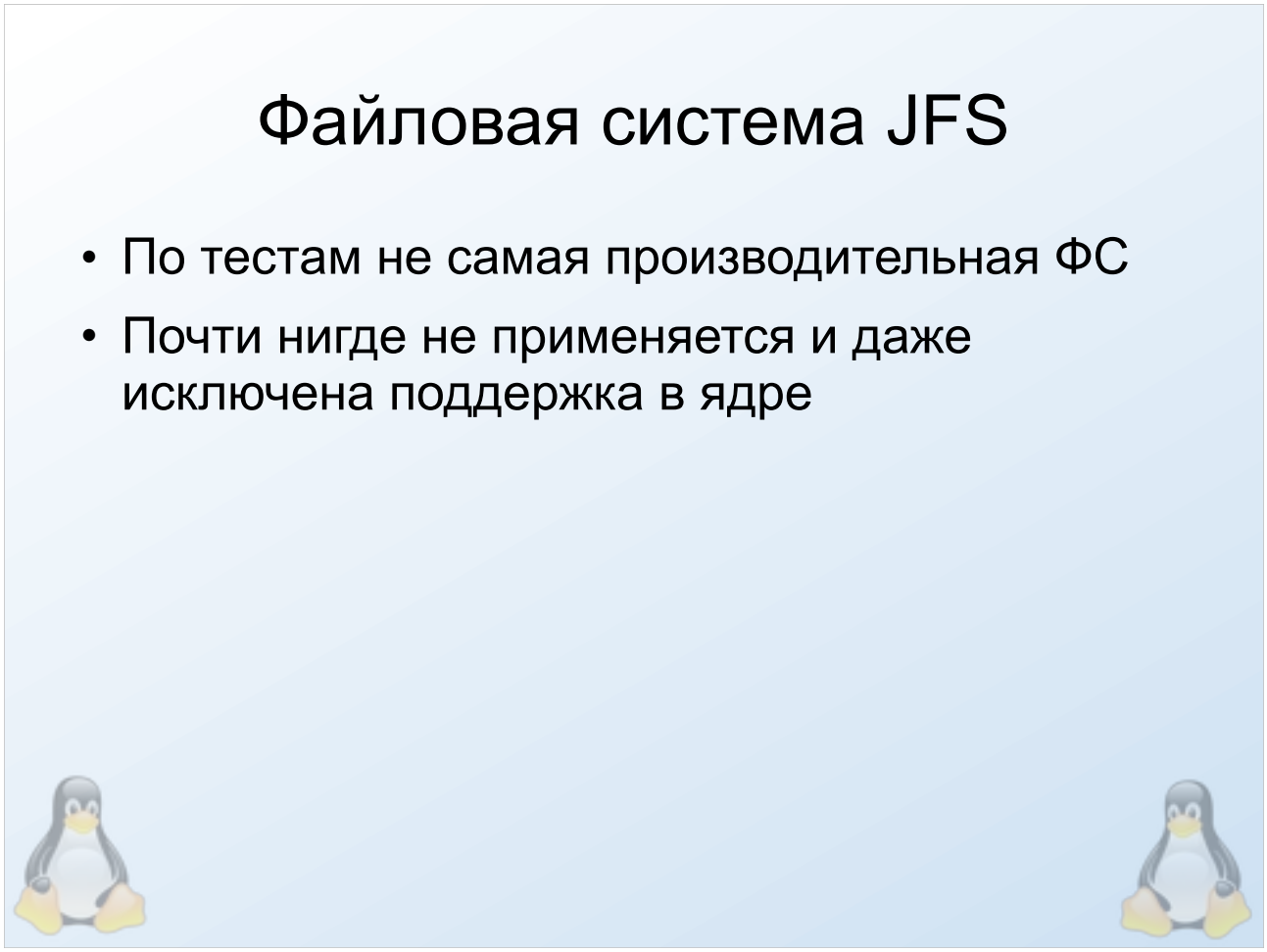
```
#/sbin/mkfs.xfs -d agcount=5 -l size=32m -f /dev/hdb2
```

18. Опция `-f` позволяет создать XFS поверх любой существующей ФС.

Примечание: при создании раздела поверх ReiserFS (и наоборот) необходимо заполнить нулями начальный раздел, содержащий метаданные перед переформатированием, т.к. команда mount может неправильно определить, какая из файловых систем установлена.

19. Для повышения производительности можно задать некоторые опции `noatime, nodiratime, osyncisdsync`, вместе помогающие добиться асинхронного вывода информации и практически имитировать поведение ext2, а также `logbufs`, устанавливающую размер буфера (по умолчанию равен 2).

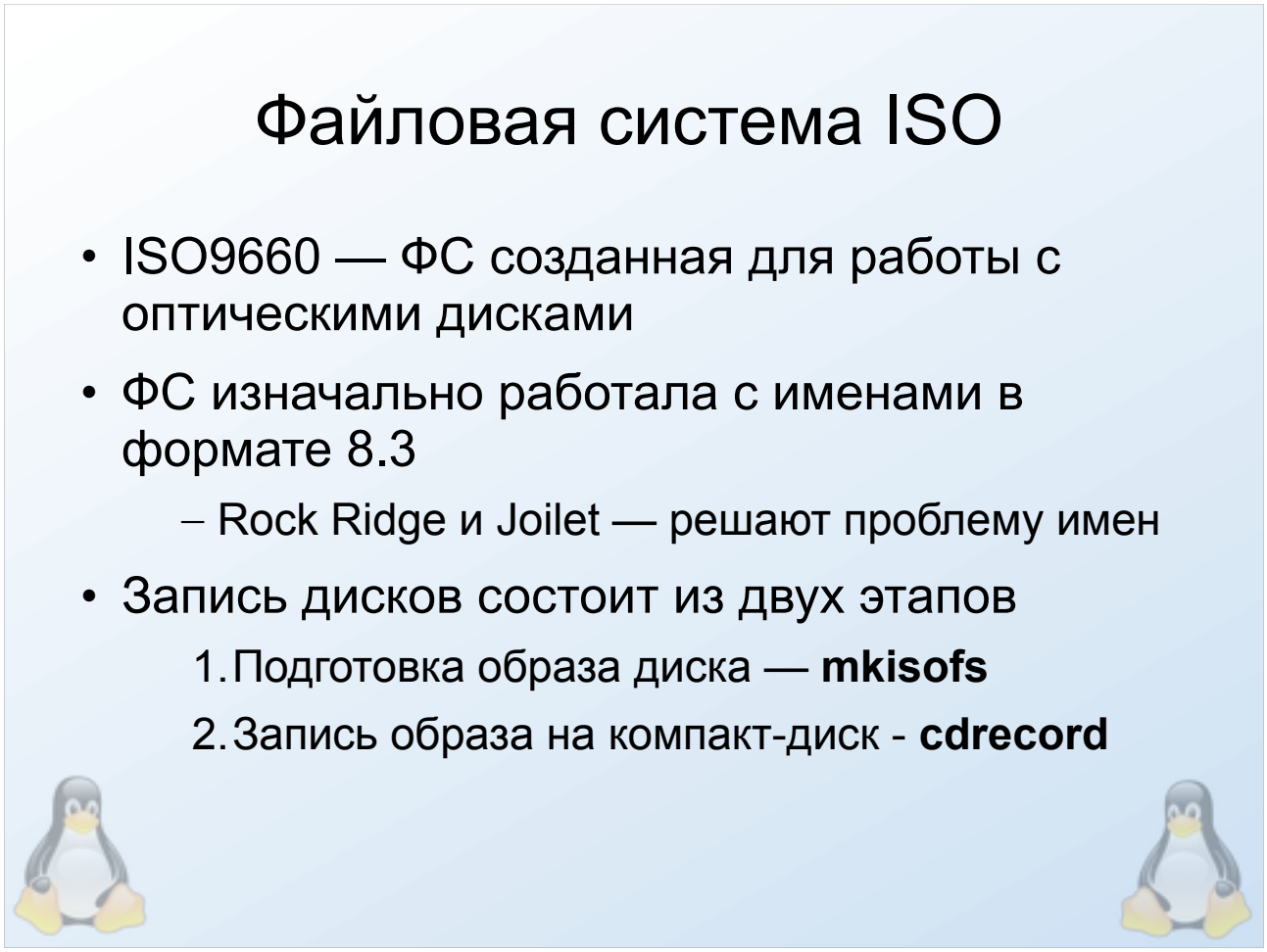
1.5. Файловая система JFS



1. Первоначально создана фирмой IBM для своей OS/2 Warp, а затем выпущена по лицензии GPL и портирована под Linux.
2. Всю необходимую информацию можно получить по адресу `http://oss.software.ibm.com/jfs`.
3. По своим характеристикам и архитектуре очень схожа с XFS.
4. Как и в XFS, в этой файловой системе раздел логически подразделяется на «агрегаты», но последние включают, кроме данных, еще и отдельный журнал, при этом каждый из таких сегментов можно монтировать отдельно.
5. Имеется возможность хранения маленьких файлов в пределах inode.
6. Если каталог имеет до 8 файлов, то информация о них содержится в самом inode, при увеличении же их количества используются уже B+-деревья.
7. По тестам это, наверное, самая медленная файловая система из рассматриваемых, хотя и разрабатывалась она для работы на высокопроизводительных серверах.
8. Для установки необходима утилита `jfsutils`, патч к ядру jfs-2.4.x-patch и код ФС jfs-2.4-1.0.20.tar.gz.

1.6. Файловая система ISO.

1.6.1. Создание ISO образов mkisofs.



1. В файловой системе ISO9660 применяется стандартное соглашение об именах DOS в формате 8.3, и все имена используют верхний регистр символов.
2. В настоящее время в большинстве компакт-дисков используются расширения этого стандарта: Rock Ridge или Joliet.
3. В Rock Ridge используются специальные файлы для отображения коротких имен в длинные и хранения информации о владении и правах доступа.
4. Поддержка Rock Ridge по умолчанию включена, но может быть и выключена.
5. Расширение Joliet разработано для ОС семейства Windows.
6. В Joliet для длинного имени отводится 64 символа + имя в формате 8.3.
7. Обычно запись CD под Linux выполняется в 2 шага:
   1. упаковка желаемых данных (файлы, музыка или и то, и другое) в файлы в специальном формате
   2. запись файлов на CD-R с помощью, например, `cdrecord`

Примечание: некоторое количество дискового пространства CD используется для помещения информации о ISO-9660 файловой системе (обычно несколько мегабайт). 620Мб данных всегда поместится на 650Мб CD-R.

Примечание: Перед использованием любого носителя (например гибкого диска, жесткого диска или CD) надо создать файловую систему (DOS формулировка: отформатировать). Эта файловая система ответственна за организацию и объединение файлов, которые должны быть сохранены на носителе. Обычно утилиты для создания файловой системы на разделе жесткого диска записывают пустую файловую систему на них, которая затем монтируется и заполняется фалами, как нужно пользователю. Записываемый CD только одноразово записываем, так если мы записали бы пустую файловую систему, он станет форматированным - но останется полностью пустым навсегда.: Это также справедливо для перезаписываемых носителей, т.к. Вы не можете произвольно изменять сектора, но Вы можете стереть все содержимое. Поэтому необходимо подготовить образ будущего CD.

8. Первый шаг создание файловой системы в файле и перенос туда файлов. Это делается утилитой `mkisofs`.

Пример:

```
mkisofs -r -o cd_image dir_for_image/
```

Примечание: Опция '-r ' устанавливает права всех файлов на чтение всем на CD и разрешает расширение Rock Ridge. Без '-r ' точка монтирования получает права доступа `dir_for_image/`).

9. `mkisofs` пробует отобразить все имена файлов в формате 8.3, используемым DOS, чтобы гарантировать самую высокую возможную совместимость.
10. В случае конфликтов имен (различные файлы имеют тот же самое 8.3 имя), в именах файла используются числа, и информация относительно выбранного имени файла печатается в `STDERR` (обычно экран).
11. Есть три причины, почему выход `mkisofs` непосредственно не послан на устройство записи CD:
    1. `mkisofs` ничего не знает о устройствах записи CD.
    2. Можно протестировать образ перед записью.
    3. На медленных машинах это было бы не надежно.

Примечание: Можно создать дополнительный раздел для записи образа взамен файла. Имеются несколько соображений против такой стратегии, 1. если Вы записываете на неправильный раздел (из-за ошибки при наборе команды), Вы можете полностью потерять Linux-систему. 2. Кроме того, это - трата дискового пространства, потому что CD-изображение - временные данные, которые могут быть удалены после записи CD. Однако использование сырого раздела сохраняет время при удалении файла размером 650Мб (Если у вас достаточно дискового пространства, дополнительный раздел может сохранить время на создание/удаление образа).

12. Linux может монтировать файлы, как разделы диска.
13. Этот свойство полезно для проверки расположения каталогов образа CD и разрешений на доступ к файлам CD.

Пример: Чтобы монтировать файл `cd_image` созданный выше в каталог `/mnt/cdrom`, выполните команду

```
mount -t iso9660 -o ro,loop cd_image /mnt/cdrom
```

Примечание: Для того, чтобы `mount` была способна работать с петлевыми устройствами, необходима соответствующая поддержка в ядре.

1.6.2. Запись CD образа на CD

1. Перед записью образа на диск следует выяснить как подключено устройство для записи. Сделать это можно командой `cdrecord -scanbus`

Примечание: эта команда будет работать только для SCSI-подобных устройств (SCSI, USB, FireWire).

Пример:

```
[root@localhost root]# cdrecord -scanbus
Cdrecord 2.0 (i686-pc-linux-gnu) Copyright (C) 1995-2002 Jrg
Schilling
Linux sg driver version: 3.1.25
Using libscg version 'schily-0.7'
cdrecord: Warning: using inofficial libscg transport code version
(schily - Red
Hat-scsi-linux-sg.c-1.75-RH '@(#)scsi-linux-sg.c        1.75
02/10/21 Copyright
1997 J. Schilling').
scsibus0:
        0,0,0     0) 'MITSUMI ' 'CR-48XATE ' '1.0E' Removable CD-
ROM
        0,1,0     1) *
        0,2,0     2) *
        0,3,0     3) *
        0,4,0     4) *
        0,5,0     5) *
        0,6,0     6) *
        0,7,0     7) *
```

Примечание: команда показала к какому SCSI-устройству подсоединен CD-writer. Если у вас CD-writer подключен через IDE интерфейс, то последняя команда возможно работать не будет.

Примечание: Устройства записи CD должны обеспечиваться постоянным потоком данных, потому что у них маленький объем буферной памяти. Т.о. процесс записи образа CD не должен прерываться, или в результате CD будет испорчен. Достаточно просто прервать поток данных удаляя большой файл. Например: если Вы удаляете предыдущий образ размером 650Мб, ядро должно обновить информацию о 650000 блоках на жестком диске (предполагается размер блока 1Кб на Вашей файловой системе). Это требует некоторое время и очень похоже на замедление дисковой активности на несколько секунд. Однако, чтение почты, просмотр web или компиляция ядра обычно не сказывается на процессе записи на современных машинах. Обратите внимание, что нет устройства записи, которое может снова устанавливать лазер и продолжать в первоначальном месте на CD, если оно сбилось. Следовательно любые сильные колебания или механический удар возможно уничтожит CD, который вы пишете.

2. Для записи диска следует выполнить команду

```
# cdrecord dev=$SCSI_BUS,$SCSI_ID,$SCSI_LUN cd_image
```

Пример:

```
# cdrecord -v speed=4 dev=0,6,0 -data  cd_image
```

Примечание: Здесь добавлены несколько полезных опций. Опция -v подробное отображения процесса записи. Опция speed=4 указывает скорость записи. Опция -data делает командную строку подобную той, которая используется для записи audio-CD, но она не обязательна.

3. Для записи дисков на устройствах ATAPI необходимо указывать параметр `dev=ATAPI`.

4. Для упрощения процесса записи можно настроить конфигурационный файл `cdrecord -/etc/cdrecord.conf`.

5. В конфигурационном файле можно задавать устройство и параметры записи по умолчанию.
6. Если вы использует `cdrecord` для перезаписи CD-RW, Вы должны добавить опцию "`blank=`*тип*" для стирания старого содержимого.

Примечание: прочтите man-страницу для понимания различных методов очистки содержимого CD-RW.

Пример: вывод образа `mkisofs` прямо на cdrecord:

```
$ IMG_SIZE=`mkisofs -R -q -print-size private_collection/  2>&1 \
| sed -e "s/.* = //"`
$ echo $IMG_SIZE
$ [ "0$IMG_SIZE" -ne 0 ] && mkisofs -r \
|private_collection/ | cdrecord speed=2  dev=0,6,0 \
|tsize=${IMG_SIZE}s -data -
```

Примечание: Первая команда запускается для определения размера образа (для этого Вам нужен mkisofs из дистрибутива cdrecord). Может быть Ваш writer не требует указания размера образа, тогда Вы можете это пропустить. Полученный размер должен быть подставлен как tsize-параметр для cdrecord (он помещается в переменную окружения IMG_SIZE). Вторая комманда - последовательность mkisofs и cdrecord, соединенные в поток.

7. Запись audio-CD имеет некоторые отличия:
   1. audio-CD состоит из аудио треков, которые организованы как отдельные образы. Т.о. если вы хотите создать 10 треков на вашем CD, вы должны создать десять образов.
   2. Второе отличие - формат образов не ISO9660 (или подготовленной файловой системы), а "16 бит стерео выборки в PCM кодировании на 44100 выборок/секунду (44.1кГц)".
8. Одна из утилит для конвертирования звуковых файлов в требуемый формат - `sox`.

Пример: Использования `sox`:

```
$ sox song.wav  song.cdr
```

Примечание: Эта команда должна преобразовать песню song из WAV-формата в CDR-формат. См. man-страницу sox для подробностей о форматах и расширениях файлов, которые распознает sox. Т.к. преобразование требует много дискового пространства, в cdrecord была встроена функция преобразования из WAV и AU. Теперь, если Ваши файлы имеют расширение .wav или .au (и формат стерео 16 бит 44.1 кГц), Вы можете использовать их как образы без ручного преобразования.

9. `cdrecord` записывает образы как audio-треки, если указана опция `-audio`.
10. Другие опции идентичны используемым при записи CD с данными (если у вас нет особых требований).

Пример: Три примера делают одно и то же, но читают треки из различных форматов звуковых файлов:

```
# cdrecord -v speed=2 dev=0,6,0  -audio  track1.cdr track2.cdr...
# cdrecord -v speed=2 dev=0,6,0  -audio  track1.wav track2.wav...
# cdrecord -v speed=2 dev=0,6,0  -audio  track1.au  track2.au...
```

11. Специальный случай MP3 файлы, которые могут быть конвертированы в требуемый формат командой "`mpg123 -s track1.mp3 > track.cdr`".

Предостережение: *это создает файлы в байтовом порядке, что требует обращения к опции -swab в cdrecord.*

Пример: создания CD_R из нескольких MP3 файлов:

```
for I in *.mp3
do
    mpg123 -s $I | cdrecord -audio -pad -swab -nofix -
done
cdrecord -fix
```

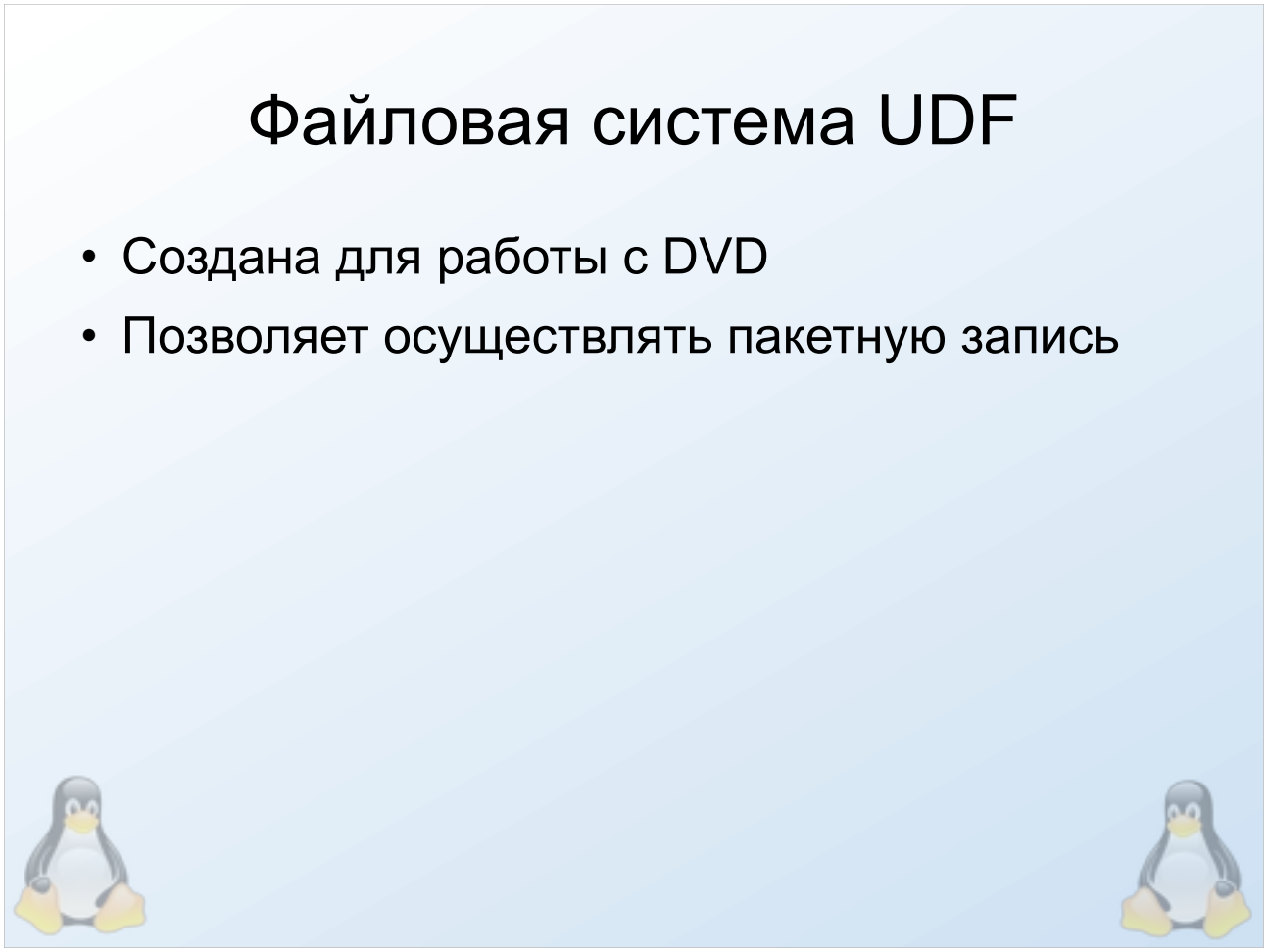
Примечание: В зависимости от скорости вашей машины, вы можете занизить запись до "speed=1" (опция cdrecord). Если вы используете "speed=4", ваша машина должна быть способна проигрывать MP3 файлы с учетверенной скоростью. mpg123 требует много процессорного времени!

12. Если вы не уверенны в возможности правильной записи, попробуйте использовать `cdrecord` с опцией `-dummy`, что оставит лазер выключенным.
13. Если вы хотите избежать пауз между звуковыми треками, нужно использовать режим записи disk-at-once (DAO) вместо track-at-once (TAO) записи описанной выше. Опция `-dao`
14. Для записи комбинированного диска data-audio просто укажите последовательно типы образов с опцией `-data` и `-audio`.

Пример:

```
cdrecord  -v dev=0,0,0 -data cd_image -audio track*.cdr
```

1.7. Файловая система UDF (Universal Disk Format)



1. UDF это файловая система, которая позволяет пакетную запись на CD/DVD-диски.
2. Файловая система UDF изначально разрабатывалась для записи на DVD-диски.
3. CD или DVD-диск в таком случае выглядит как маленький жесткий диск.

Примечание: Вам не нужно создавать образы дисков, и не надо открывать/закрывать сессии.

4. Поддержка файловой системы UDF появилась в ядрах серии 2.4.
5. Но для CD-R/CD-RW только в режиме чтения. Поддержка записи на UDF в ванильное ядро 2.4 не включена.
6. Стандартные программы записи дисков под Linux (`cdrecord` и `mkisofs`) не поддерживают `udf` в них это только планируется, поэтому приходится пользоваться средствами, которые предлагают разработчики проекта `linux-udf`.
7. В пакете `udftools` для этого предусмотрены 3 программы:
   1. `pktsetup`, служит для установки связи с `packet device` (т.е CD-RW приводом).
   2. `cdrwtool`, нужна для начальной разметки/форматирования диска CD-RW.
   3. `mkudffs`, служит для создания udf-образов (dvd, dvdram, cd-r).

Примечание: Запись таких образов производится cdrecord.

4. Для проверки дисков есть программа `chkudf`, пока только проверяющая целостность записи и не входит в состав пакета.
5. В будущем планируется и `udffsck`, которая будет восстанавливать запись.

1. После установки пакета добавляем в `/etc/modules.conf` следующие строчки, чтобы модуль для записи загружался автоматически для ядра до 2.6:

```
# Packet CD writing
alias block-major-97 pktcdvd
alias /dev/pktcdvd[0-9]* pktcdvd
```

2. Потом добавляем в `/etc/rc.d/rc.local` (если такого файла нет, создаем и даем ему права на выполнение) следующие строки:

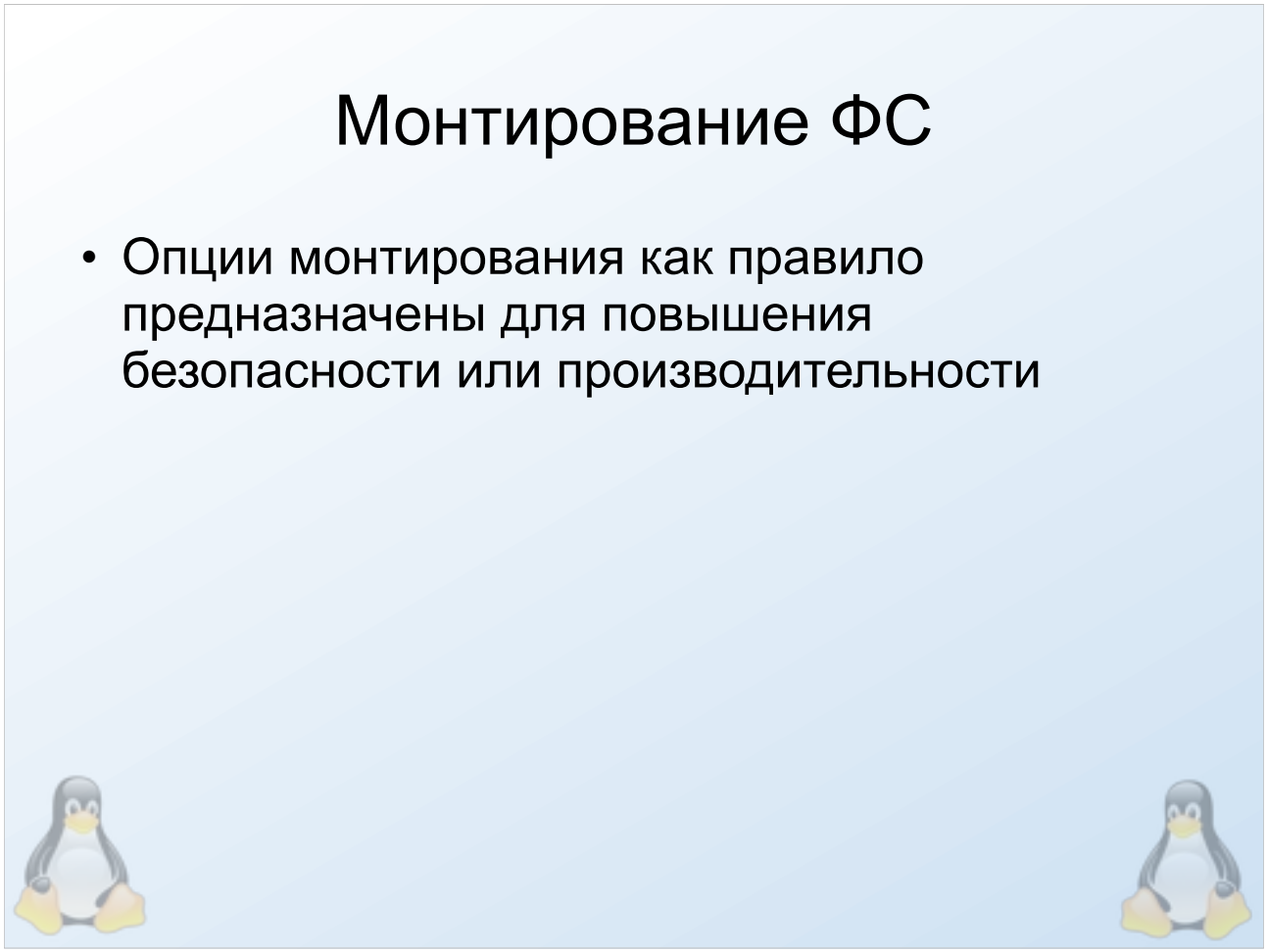
```
#!/bin/sh
pktsetup /dev/pktcdvd0 /dev/scd0
```

Примечание: pktsetup используется для ассоциации пакетного устройства с блочными CD или DVD устройствами, после чего пакетное устройство может быть монтировано, как файловая система с возможностью чтения/записи. Без этого после перезагрузки придется каждый раз выполнять команду pktsetup /dev/pktcdvd0 /dev/scd0. Пример работает, когда CD-ROM работает через эмуляцию SCSI, поэтому /dev/scd0.

3. Перед записью диска необходимо подготовить CDRW болванку (стереть и отформатировать ее под UDF): `cdrwtool -d устройство -q`

1.8. Монтирование ФС

1.8.1. Опции монтирования для различных ФС



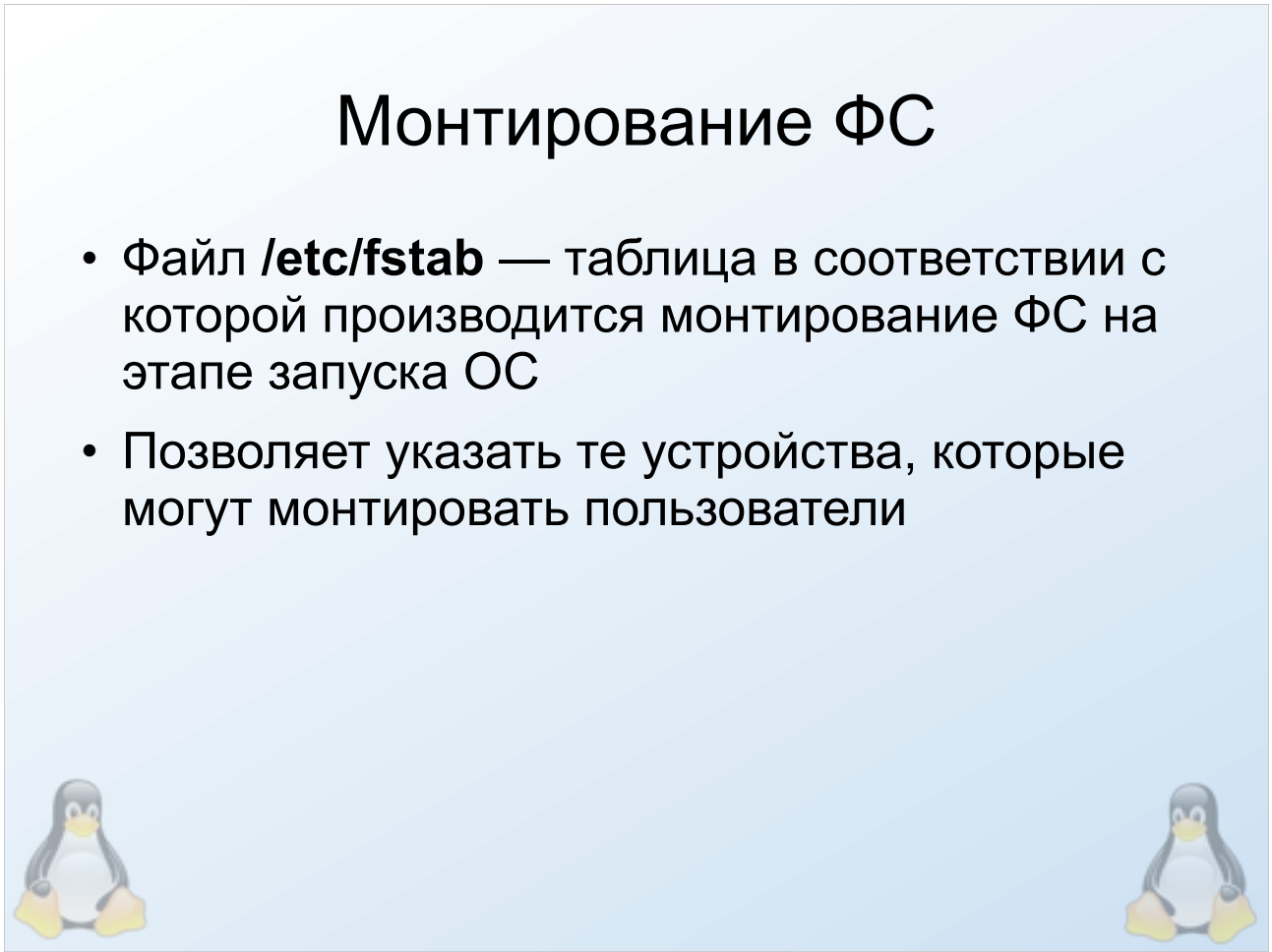
1. Для монтирования файловых систем применяется команда `mount`.
2. Опция `-t` позволяет указать тип файловой системы для монтирования.
3. Основные опции команды `mount` общего применения:
   1. `-f` - симулировать монтирование (обычно применяется вместе с `-v`);
   2. `-n` - монтировать без занесения соответствующей записи в файл `/etc/mtab`;
   3. `-l` - добавление LABEL к файловой системе;
   4. `-L` - монтирование файловой системы по LABEL;
   5. `-r` или `-o ro` - монтирование в режиме только для чтения;
   6. `-w` или `-o rw` - монтирование для чтения и записи.
4. Опции команды `mount`, характерные для ext2:

1. `-o check=normal` - во время монтирования будет проверяться битовая матрица блоков и массив индексных дескрипторов;
2. `-o check=strict` - проверка блоков для удаления в области данных;
3. `-o check=none` или `nocheck` - проверка не производится;
4. `-o errors=continue` - при наличии ошибки продолжать работу;
5. `-o errors=remount-ro` - при наличии ошибки монтировать только для чтения;
6. `-o errors=panic` - при наличии ошибки порождать панику ядра.
7. `-o sb` - указатель номера блока суперблока.
8. `-o grpid / -o nogrpid` – использовать / не использовать BSD стиль для задания прав доступа на вновь создаваемые файлы.

Примечание: По умолчанию используется стиль System V (nogrpid) стиль, когда файлы получают группу доступа такую же как процесс создавший их. В BSD стиле файлы имееют группу такую же как каталог, в котором они созданы. BSD стиль можно задать установив бит SGID на каталог.

9. `-o sb=n` - использовать суперблок с указанным номером.
10. `-o acl` – включить POSIX ACL.

5. Опции команды `mount`, характерные для `ext3`:
   1. `-o noload` - не загружать журнал;
   2. `-o data=journal` - все данные заносятся в журнал до записи данных в файловой системе;
   3. `-o data=ordered` - данные заносятся в файловую систему до записи информации об их метаданных в журнал;
   4. `-o data=writeback` - неупорядоченная запись данных в журнал.



```
UUID=4693409a-9326-4d88-8dcd-57683013905d /
btrfs   defaults        0 0
UUID=88d6e2f6-4893-4106-8523-b21dab65b728 /boot
ext4    defaults        1 2
UUID=AD1C-AC87          /boot/efi               vfat
umask=0077,shortname=winnt 0 2
UUID=4693409a-9326-4d88-8dcd-57683013905d /home
btrfs   subvol=HOME,compress=zstd:1 0 0
UUID="c2aabb60-fe36-4585-bba4-05bbadcb6a98" /hdd btrfs defaults 0
0
UUID=a9575ef5-baa6-49fc-ae87-f9fa57be3fa2 none
swap    defaults        0 0
```

3. Строки, начинающиеся с символа # являются комментариями.
4. Таблица, содержащаяся в файле `/etc/fstab` , состоит из колонок полей, назначение которых следующее:
   1. Поле `<device>` содержит имя файла монтируемого устройства. Принято указывать UUID ФС. UUID можно узнать командой `blkid`.
   2. Поле `<mount point>` указывает каталог - точку монтирования.
   3. Поле `<type>` содержит тип файловой системы на данном носителе.
   4. Поле `<options>` содержит опции команды `mount` , которые должны быть использованы при монтировании данной файловой системы. Для разделов swap это поле должно содержать `sw`.
   5. Поле `<dump>` указывает надо ли для данной файловой системы производить автоматическое резервное копирование (backup) командой `dump` . Если в этом поле находится:
      1. `1` – резервное копирование производится,
      2. `0` – нет. Для файловой системы `Reiserfs` это поле должно быть установлено в `0` .
   6. Поле `<pass>` предназначено для порядка проверки целостности файловых систем при загрузке операционной системы.
      1. `1` – для корневой файловой системы.
      2. `2` – для других файловых систем, которые необходимо проверять при загрузке.
      3. `0` – если проверка не требуется.
5. Ниже приведена таблица, содержащая часто используемые опции команды mount, указываемые в поле `<options>` файла `/etc/fstab` .

Опция	**Назначение**
`defaults`	Установки: `rw,suid,dev,exec,auto,nouser,asynch.`
`asynch/sync`	Асинхронный/синхронный режим ввода/вывода.
`auto/noauto`	Монтировать/Не монтировать во время загрузки.

exec/noexec	Разрешение/Запрет выполнения файлов с машинным кодом.
suid/nosuid	Разрешение/Запрет установки бита SUID.
user/nouser	Разрешение/Запрет монтироваия устройства обычным пользователем.
atime/noatime	Обновлять/Не обновлять дату последнего доступа
dev/nodev	Интерпретировать/Не интерпретировать символьные или специальные блочные устройства.
ro/rw	Режим только для чтения/чтения-записи.
remount	Если ФС смонтирована, то попытаться ее перемонтировать.

6. Имеет значения порядок следование опций монтирования.

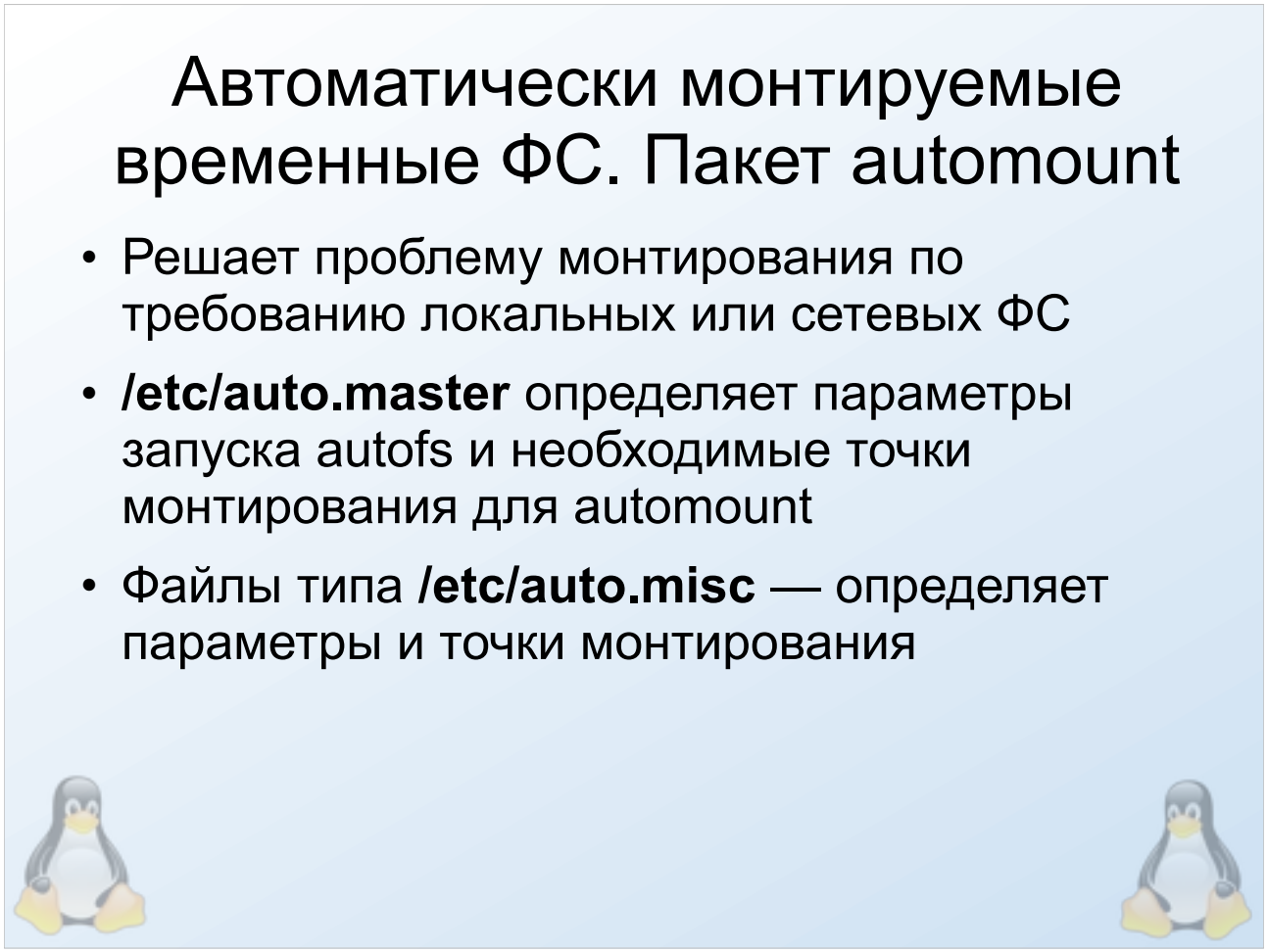
Пример: записи `user,defaults` и `defaults,user` будут иметь разное значение. В первом случае опция `defaults` переопределит опцию `user`, следовательно такая запись не имеет смысла. Обычно `defaults` следует указывать первой в списке опций.

7. Наличие записи в файле `/etc/fstab` означает, что данная файловая система может быть смонтирована без указания обоих аргументов командной строки `mount` – файла устройства и каталога – точки монтирования. Для монтирования файловой системы, указанной в `/etc/fstab` , достаточно указать либо точку монтирования, либо файл устройства.

1.8.3. Файлы /etc/mtab и /proc/mounts

1. Файлы `/etc/mtab` и `/proc/mounts` содержат список сонтированных в данное время файловых систем.
2. В файл `/etc/mtab` утилита `mount` вносит записи о смонтированных ей устройствах, а также о опциях монтирования этих устройств.
3. `mount` использует `/etc/mtab` для предотвращения повторного монтирования файловых систем.
4. Файл `/proc/mounts` (в ядре 2.6 `/proc/self/mounts`) отображается структура ядра о смонтированных устройствах.

1.9. Автоматически монтируемые временные ФС. Пакет automount



1. Пакет *automount* позволяет автоматически монтировать временные файловые системы.
2. В состав пакета входят:
   1. демон автомонтирования `autofs`;
   2. утилита автомонтирования `automount`.
3. Демон автомонтирования отслеживает доступ к каталогам точкам монтирования и запускает `automount` для фактического монтирования ФС.
4. Для настройки автомонтирования используются два файла: `/etc/auto.master` и `/etc/auto.misc`
5. В файле `/etc/auto.master` определяется параметры запуска `autofs` и необходимые точки монтирования для `automount`.

Пример:

```
/misc     /etc/auto.misc --timeout=60
```

Примечание: в этом примере определяется “map” файл, в котором описанны нужные точки монтирования. Параметр timeout задает время в секундах по истечении, которого происходит размонтирование ресурса.

6. В файле `/etc/auto.misc` указываются точки монтирования, которые будут монтироваться автоматически при доступе к указанным в нем каталогам, а также опции монтирования.

Пример:

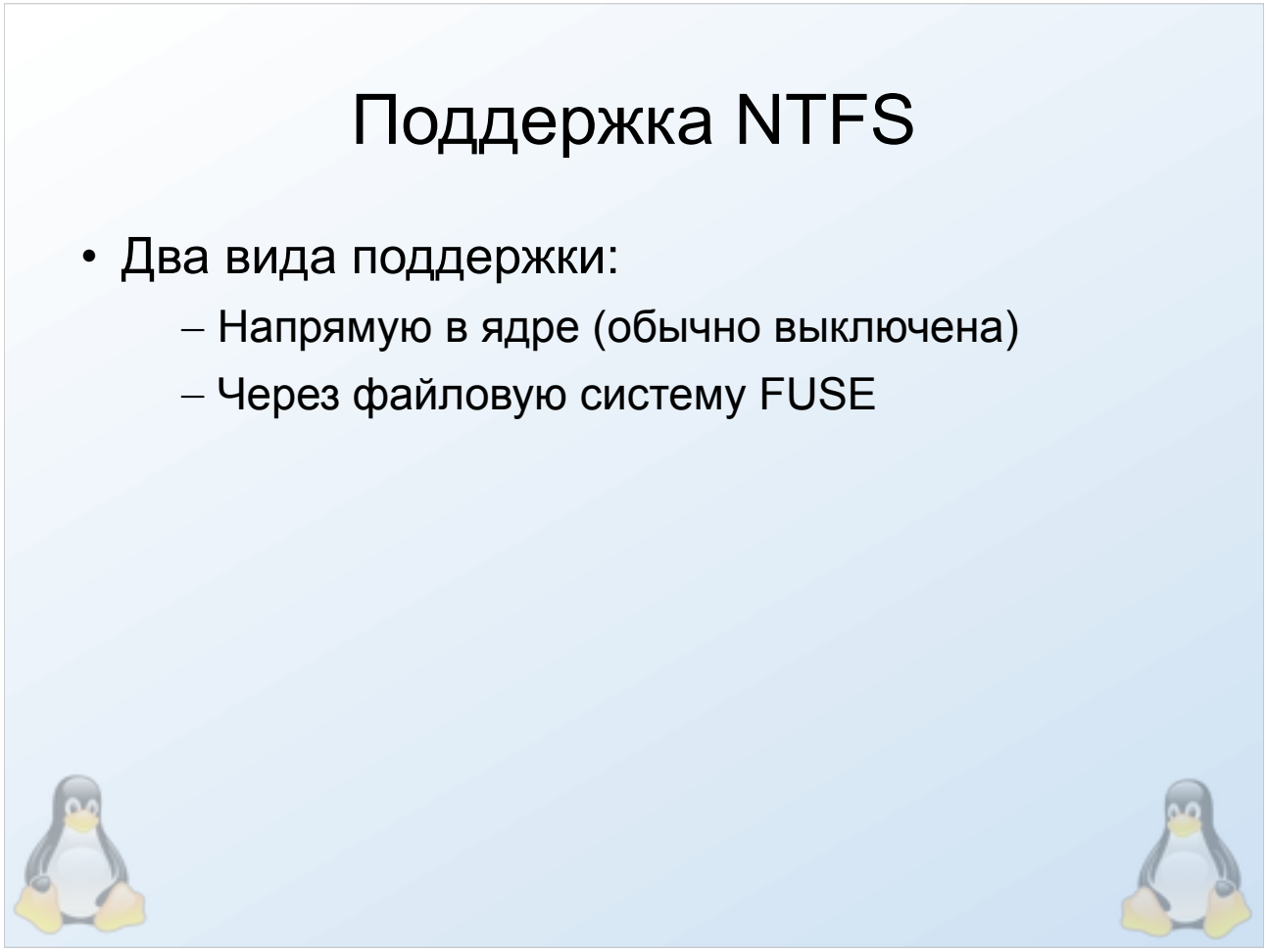
```
kernel          -ro,soft,intr           ftp.kernel.org:/pub/linux
cd              -fstype=iso9660,ro      :/dev/cdrom
```

Примечание: здесь задаются две точки монтирования kernel и cd. Значит при попытке доступа в каталог /misc/kernel смонтируется каталог /pub/linux на узле ftp.kernel.org, и при попытке открыть /misc/cd смонтируется локальное устройство /dev/cdrom.

7. Файл `/etc/auto.misc` может называться по другому, достаточно указать его в `/etc/auto.master`

1.10. Поддержка NTFS в Linux

1.10.1. Варианты поддержки



1.В ядрах Linux, начиная с версии 2.2.0 имеется поддержка чтения данных с разделов NTFS.

2.Ядра 2.6.x поддерживают чтение, перезапись и изменение размера данных.

3.Для поддержки чтения и записи данных можно воспользоваться как коммерческими, так и свободными программными средствами. Среди которых:

1.Linux-NTFS project. (www.linux-ntfs.org)

2.Tuxera. (Имеется свободная версия NTFS-3G и коммерческая для встроенных сисстем. www.tuxera.com)

3.Paragon Software Group (предлагает свободную и коммерческую реализацию поддержки NTFS. www.paragon-software.com)

4.Captive NTFS (www.jankratochvil.net/project/captive)

4.Свободные реализации в основном работают через драйвер FUSE (Filesystem in Userspace fuse.sourceforge.net)

5.Проверить наличие модуля fuse можно командой:

```
modeprobe -l fuse
```

6.Драйвер FUSE добовляет поддержку новых файловых систем без компиляции ядра.

7.Этот драйвер также позволяет использовать «нестандартные» файловые системы, например: HTTPFS, FTPFS, vmware-mount или шифровать файловые системы.

8.Работа драйверов файловых систем, использующих FUSE, производится userspace, а, значит, такие файловые системы имеют ограничения по производительности.

1.10.2. Настройка NTFS-3G

1.Пакет NTFS-3G получить по адресу http://www.tuxera.com/community/ntfs-3g-download/

2.Пакет распаковать:

```
tar xzvf ntfs-3g-2010.3.6.tgz
```

3.Перейти в каталог с исходными кодами, и выполнить команды:

```
./configure
make
make install
```

4.Монтирование NTFS раздела выполняется командой:

```
mount -t ntfs-3g /dev/sda1 /mnt/windows
```

5.Для постоянного монтирования раздела в /etc/fstab необходимо добавить строку вида:

```
/dev/sda1 /mnt/windows ntfs-3g defaults 0 0
```

1.10.3. Настройка ntfsprogs

1.В рамках проекта Linux-NTFS разрабатываются патчи на ядро и набор утилит для работы с файловой системой NTFS.

2.Утитлиты ntfsprogs не требуют изменения ядра, так как работают через драйвер FUSE.

3.Получите пакет

4.Распакуйте и перейдите в каталог с исходным кодом.

5.Запустите скрипт `configure` с опциями

1. `--enable-ntfsmount`: компилирует `ntfsmount`
2. `--enable-crypto`: добавляет поддержку шифрованных файлов в libntfs и компилирует `ntfsdecrypt`.
3. `--enable-gnomevfs`: включает поддержку NTFS в gnome-vfs.

Примечание: для компиляции в CentOS вам потребуются пакеты: fuse-devel, gnutls версии 1.4.4 или выше, libconfig версии 1.0.1 или выше.

6.Команда `make all` компилирует все утилиты

7.Команда `make extra` компилирует дополнительные (нестабильные) утилиты.

8.После установки получаем утилиты для работы с файловой системой NTFS, среди которых:

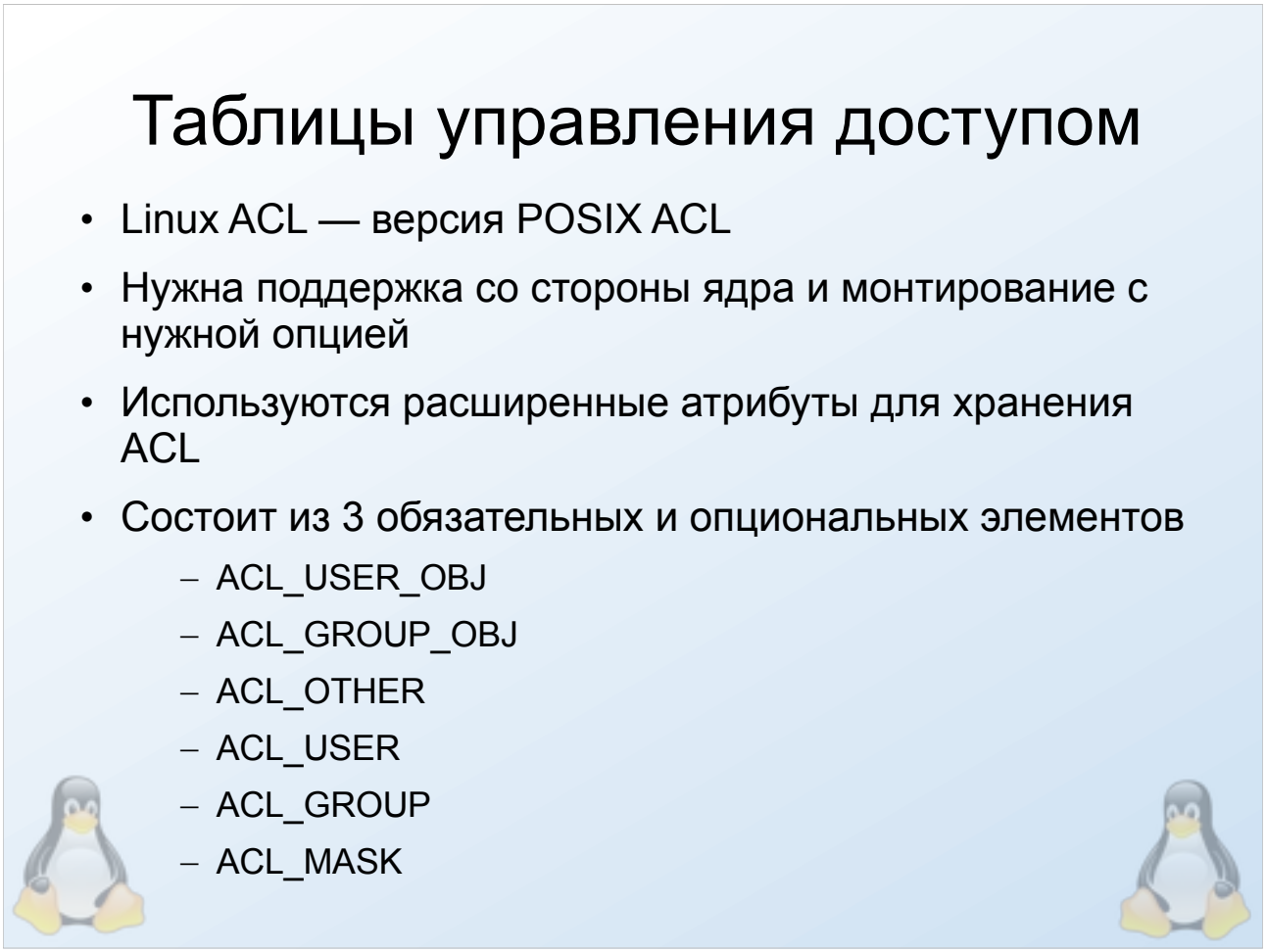
1. `ntfsmount`

2.ntfsresize

3.ntfsundelete

1.11. Таблицы управления доступом

1.11.1. Общие сведения о ACL



1. Стандартная система доступа к файлам Linux-систем предусматривает 12 бит, которые определяют вид доступа для трех категорий пользователей: владельца, группы владельцев и всех остальных.
2. Кроме этого в такой системе нет наследования прав доступа.
3. Такая система хорошо работает в простых ситуациях, но в сложных случаях она создает проблемы для администраторов.
4. Для решения данных проблем можно воспользоваться Linux ACL. (Access Control Lists - списки контроля доступа) - версией POSIX ACL для Linux.
5. Linux ACL — это набор патчей для ядра операционной системы и программ для работы с файловой системой и несколько утилит, дающих возможность устанавливать права доступа к файлам не только для пользователя-владельца и группы-владельца файла, но и для любого пользователя или группы.
6. В версии ядра 2.6 ACL включен в стандартную поставку.
7. Linux ACL использует расширенные атрибуты для хранения данных о правах доступа к

файлам пользователей и групп.

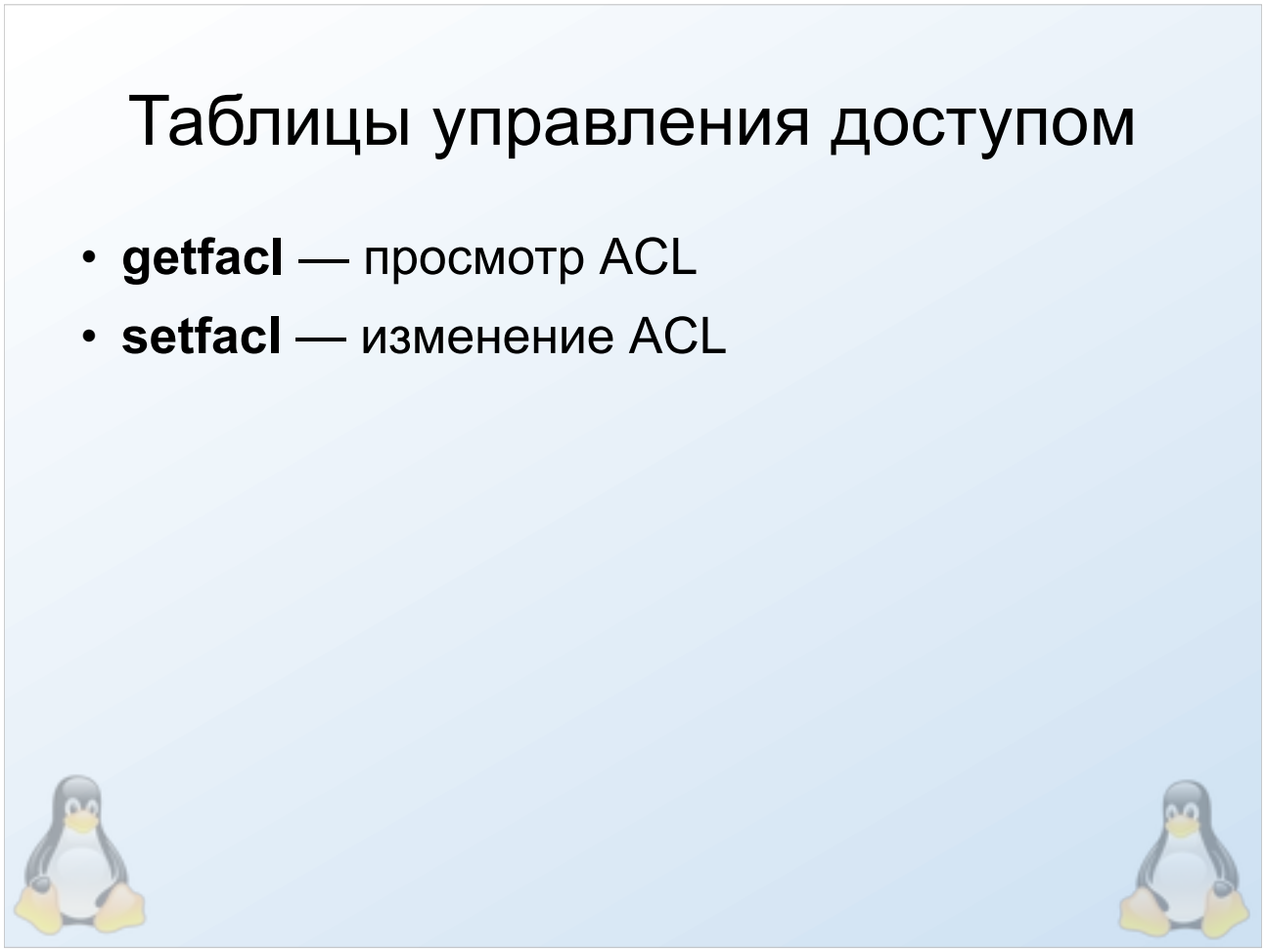
8. Расширенные атрибуты — это пара “имя/значение’, привязанная к определенному файлу.
9. Список расширенного контроля доступа существует для каждого inode и состоит из шести компонентов.
    1. Первые три являются копией стандартных прав доступа к файлу. Они содержатся в единственном экземпляре в ACL и есть у каждого файла в системе:
        1. ACL_USER_OBJ — режим доступа к файлу пользователя-владельца;
        2. ACL_GROUP_OBJ — режим доступа к файлу группы-владельца;
        3. ACL_OTHER — режим доступа к файлу остальных пользователей.
    2. Следующие компоненты устанавливаются для каждого файла в отдельности и могут присутствовать в ACL в нескольких экземплярах:
        1. ACL_USER — содержит UID и режим доступа к файлу пользователя, которому установлены права, отличные от основных.
        2. ACL_GROUP — то же самое, что и ACL_USER, но для группы пользователей;
        3. ACL_MASK — маска действующих прав доступа для расширенного режима.
10. На каждого пользователя со своими правами на данный файл хранится отдельная запись. Не может существовать более одной записи на одного и того же пользователя.
11. При установке дополнительных прав доступа присваивается значение и элементу ACL_MASK.
12. Каталоги также могут иметь список контроля доступа по умолчанию. В отличие от основного ACL, он действует на создаваемые внутри данного каталога файлы и каталоги.
13. При создании файла внутри такого каталога файл получает ACL, равный ACL. по умолчанию (default ACL) этого каталога - наследование.
14. Поддержка ACL должна быть включена в ядре для нужной вам ФС.

Пример: проверяем в каких ФС есть поддержка ACL.

```
$ grep ACL /boot/config-$(uname -r)
CONFIG_XILINX_EMACLITE=m
CONFIG_EXT4_FS_POSIX_ACL=y
CONFIG_REISERFS_FS_POSIX_ACL=y
CONFIG_JFS_POSIX_ACL=y
CONFIG_XFS_POSIX_ACL=y
CONFIG_BTRFS_FS_POSIX_ACL=y
CONFIG_F2FS_FS_POSIX_ACL=y
CONFIG_FS_POSIX_ACL=y
CONFIG_NTFS3_FS_POSIX_ACL=y
CONFIG_TMPFS_POSIX_ACL=y
CONFIG_JFFS2_FS_POSIX_ACL=y
CONFIG_EROFS_FS_POSIX_ACL=y
```

```
CONFIG_NFS_V3_ACL=y
CONFIG_NFSD_V2_ACL=y
CONFIG_NFSD_V3_ACL=y
CONFIG_NFS_ACL_SUPPORT=m
CONFIG_CEPH_FS_POSIX_ACL=y
CONFIG_9P_FS_POSIX_ACL=y
```

1.11.2. Установка и изменение прав доступа



1. Управление списками контроля доступа производится при помощи двух утилит: `getfacl` и `setfacl`.
2. С помощью `getfacl` можно просмотреть текущие параметры доступа любого файла.

Пример: при вызове `getfacl` для домашнего каталога пользователя `user1` мы получим следующее:

```
$getfacl /home/user1
file: home/user1
owner: user1
group: users
user::rwx
group::---
other:---
```

Примечание: Как можно видеть, каталог /home/user1 принадлежит пользователю user1, группе users и значение прав доступа к каталогу — 0700. Каталог имеет только основные параметры доступа, поскольку изначально дополнительные права не устанавливаются.

3. Дополнительные права доступа к файлу устанавливаются и изменяются при помощи утилиты `setfacl`.
4. Для этого используется следующий формат вызова:

```
setfacl -<опции> <ACL_структура>, <ACL_структура>,…,
<ACL_структура> <имя_файла> <имя_файла>
```

5. ACL_структура представляет собой одну из следующих конструкций:
   1. `[d[efault]:] [u[ser]:] <пользователь> [:<режимы_доступа>]` — определяет режим доступа к файлу или каталогу пользователя. Если пользователь не указан, определяет режим доступа пользователя-владельца;
   2. `[d[efault]:] g[roup]: <группа> [:<режимы_доступа>]` — то же, что и предыдущая конструкция, но для группы;
   3. `[d[efault]:] m[ask][:] [:<режимы_доступа>]` — определяет действующие права доступа;
   4. `[d[efault]:] o[ther][:] [:<режимы_доступа>]` — определяет режим доступа для остальных пользователей.
6. Параметр `default` определяет режим наследование прав доступа.
7. Для установки и изменения ACL. используются следующие опции:
   1. `--set-acl` — заменяет полностью ACL-файл, на указанный в командной строке;
   2. `-m` — изменяет режимы доступа к файлу (каталогу);
   3. `-x` — убирает правила доступа из ACL.

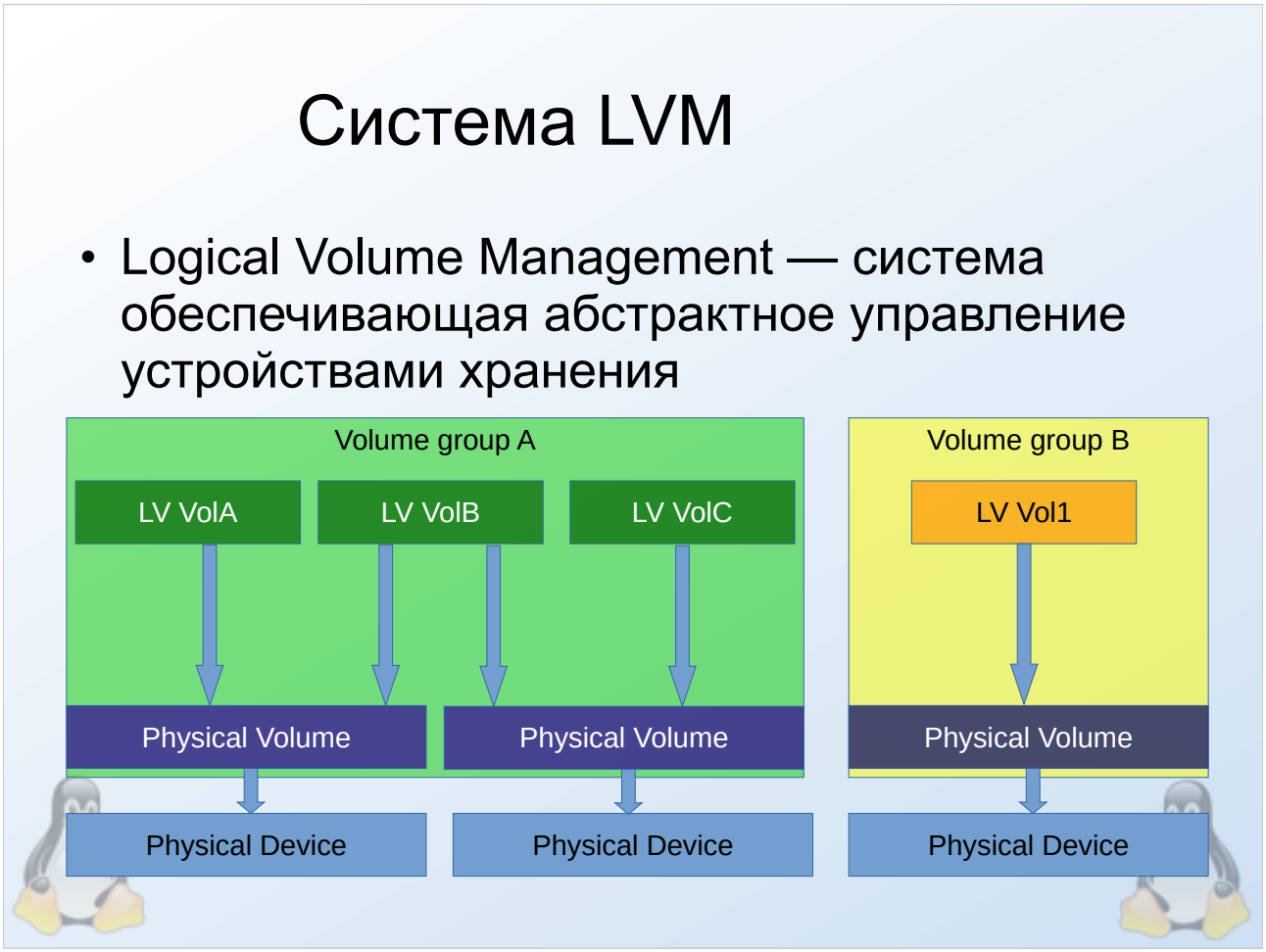
Пример: вот что мы получим, применив `setfacl` к каталогу `user1`:

```
$setfacl \
--set-acl=u::rwx,g::---,o::---,u:user2:rwx,g:users2:rx,u:user3:--- \
 /home/user1
$getfacl /home/user1
file: home/user1
owner: user1
group: users
user::rwx
user:user2:rwx
user:user3:---
group::---
group:users2:r-x
mask:rwx
other:---
```

Глава 2. Моногодисковые накопители информации

2.1. Система LVM.

2.1.1. Преимущества использования LVM



1. Система логического управления томами LVM (Logical Volume Management) обеспечивает дополнительный уровень абстракции от дисковых разделов и физических накопителей в системе. Вместо работы с системой дисковых разделов LVM предполагает операции с логическими томами, объединяющими, возможно, несколько разделов или жестких дисков.
2. Использование LVM позволяет объединять физические накопители в логические тома, обеспечивая более удобную работу с ними.
3. При использовании LVM администратор работает не с обычными файлами устройств, имена которых никоим образом не отражают суть хранимых на них данных, а с логическими устройствами, которым могут быть назначены удобные имена.
4. Система LVM обеспечивает удобные возможности изменения размеров логических томов, существенно упрощая таким образом планирование исходного разбиения жесткого диска на разделы при установке системы.

5. Система LVM обеспечивает возможность получения так называемых snapshots - моментальных снимков состояния логического тома. Суть их состоит в фиксации информации на диске в определенный момент времени. Это используется, например, при необходимости резервного копирования на лету.

2.1.2. Терминология LVM

1. Наивысший уровень абстракции в LVM - это группа томов (volume group VG). Она объединяет наборы логических и физических томов в единую административную сущность.
2. Физический том (physical volume - PV) - обычно жесткий диск, устройство RAID или раздел жесткого диска.
3. Логический том (logical volume - LV) - субъект LVM, содержащий конкретную файловую систему. Доступен, как блочный файл устройства.
4. Физический экстент (physical extent - PE) - последовательность блоков жесткого диска, предназначенная для хранения одного какого-либо файла. Физические экстенты имеют одинаковые размеры со всеми логическими экстентами в одной группе томов.
5. Логический экстент (logical extent - LE) - непрерывная последовательность блоков логического тома. Размер логических экстентов одинаков для всех логических томов в группе томов.
6. Используется в основном две стратегии отображения логических экстентов в физические:
    1. линейное отображение, в случае которого логические экстенты размещаются на физических томах последовательно. То есть, например, экстенты с первого по сотый - на первом физическом томе, а со сто первого по двухсотый - на втором физическом томе.
    2. отображение с чередованием, в котором допускается части логических экстентов размещать на различных физических томах.

2.1.3. Использование LVM.

1. Перед использованием LVM физический диск или дисковый раздел необходимо инициализировать командой `pvcreate`.

Пример:

```
pvcreate /dev/sdd
```

2. Если для LVM используется раздел диска, то вначале необходимо установить с помощью команды `fdisk` тип этого раздела `0x8e`. Далее следует использовать команду `pvcreate` аналогично тому, как она применялась при инициализации диска.

Пример:

```
pvcreate /dev/sda5
```

3. Далее требуется создать группу томов (VG). Для этого используется команда `vgcreate`.

Пример: Создание группы томов на двух устройствах:

```
vgcreate testvg /dev/hda2 /dev/hdb1
```

4. В случае использования в системе `devfs` необходимо использовать реальные имена файлов устройств вместо символических ссылок на эти файлы.

Пример: Для системы, использующей devfs:

```
vgcreate testvg /dev/ide/host0/bus0/target0/lun0/part2 \
```

```
/dev/ide/host0/bus0/target1/lun0/part1
```

5. Для вновь созданной группы томов будет установлен размер экстента по умолчанию, равный 4 Mb. При необходимости использования иного размера экстента следует указать требуемый размер после ключа `-s`.
6. После создания группы томов необходимо активировать эту группу томов. Это позволяет сделать команда `vgchange -ay`.

Пример: Активирование группы томов:

```
vgchange -ay testvg
```

7. Эта же команда `vgchange`, но с ключами `-an` позволяет деактивировать группу томов.

Пример: Деактивация группы томов:

```
vgchange -an testvg
```

8. После деактивации группы ее можно удалить с помощью команды `vgremove`. Однако, перед удалением следует убедиться, что в этой группе отсутствуют логические тома.

Пример: Удаление группы томов:

```
vgremove testvg
```

9. Команда `vgextend` позволяет добавлять физический том в существующую группу томов. Добавляемый физический том должен быть инициализирован.

Пример: добавлние физического тома `/dev/sdd` в группу томов `testvg`

```
vgextend testvg /dev/sdd
```

10. Существует возможность удаления физического тома из группы томов. Это можно сделать с помощью команды `vgreduce`.
11. Перед удалением физического тома из группы томов необходимо убедиться в том, что данный физический том не используется ни одним логическим томом. Такую информацию можно получить, используя команду `pvdisplay`.

Пример:

```
//    Проверка используется ли физический том:
   pvdisplay /dev/sdd
//    Удаление физического тома:
   vgreduce testvg /dev/sdd
```

12. Команды `vgdisplay` и `pvdisplay` предоставляют информацию, соответственно, о состоянии группы томов и физических томов в группе. Используя полученную с помощью этих команд информацию следует решить, на каких физических томах будут созданы логические тома.
13. Собственно создание логического тома осуществляется командой `lvcreate`, причем логический том может быть создан с использованием как линейного отображения на физические тома, так и с использованием чередования.

Пример:

```
   lvcreate -L2000 -nfirstlv testvg
```

Примечание: данная команда создаст линейный логический том (LV), принадлежащий группе (VG) testvg. Логический том будет в данном случае именоваться, как firstlv, и ему будет соответствовать блочное устройство /dev/testvg/firstlv. Размер тома - 2 Гб.

```
lvcreate -l100 -i2 -I16 -nsecondlv testvg
```

Примечание: Команда, приведенная выше, создаст логический том с чередованием двух участков

(опция -i2). Каждый участок имеет размер 16 Кб. Размер данного логического тома определяется количеством логических экстентов (LE), установленному в 50 (опция -l100).

14. Если необходимо удалить логический том, то для этого можно использовать команду `lvremove`.

Пример:

```
umount /dev/testvg/firstlv
lvremove /dev/testvg/firstlv
```

Примечание: В этом случае будет удален логический том /dev/testvg/firstlv.

15. Размер логического тома может быть увеличен с помощью команды `lvextend`, которой следует указать после опции `-L` либо размер, на который необходимо увеличить размер логического тома (`-L+1G`), либо желаемый размер увеличиваемого тома (`-L10G`).

Пример:

```
lvextend -L+1G /dev/testvg/secondlv
```

Примечание: Команда с использованием размера, на который увеличивается том:

```
lvextend -L10G /dev/testvg/secondlv
```

Примечание: Команда, увеличивающая размеры логического тома до 10 Гб:

16. Увеличение логического тома не означает автоматическое изменение размера файловой системы на нем.

17. Размер файловой системы могут быть изменены с помощью утилит, предназначенных для конкретных файловых систем: -

    1. ext - утилита `resize2fs`;
    2. reiserfs - утилита `resize_reiserfs`;
    3. xfs - утилита `xfs_growfs`.

18. Файловая система ext может быть как размонтирована так и смонтирована перед изменением ее размера.

19. При использовании xfs она может быть подвергнута изменению размера только тогда, когда она смонтирована. Причем, при изменении размера xfs нельзя указывать файл устройства. Вместо него необходимо указать точку монтирования:

Пример:

```
xfs_growfs /home
```

Примечание: Здесь будет произведено изменение размера файловой системы xfs, смонтированной в каталоге /home.

20. Пакет LVM предоставляет команду `lvreduce`, позволяющую уменьшать размеры логических томов. Размер логического тома не может быть меньше, чем размер файловой системы на нем.

21. ФС xfs уменьшать нельзя.

Пример:

```
lvreduce -L-2G /dev/testvg/secondlv
```

Примечание: Данная команда уменьшает размер логического тома на 2 Гб.

22. Уменьшение размера тома производится в три этапа.

    1. Уменьшается ФС на томе, так чтобы размер ФС был чуть меньше предполагаемого

размера тома

2. Уменьшается сам том.
3. ФС увеличивается, чтобы занть весь том.

Пример:

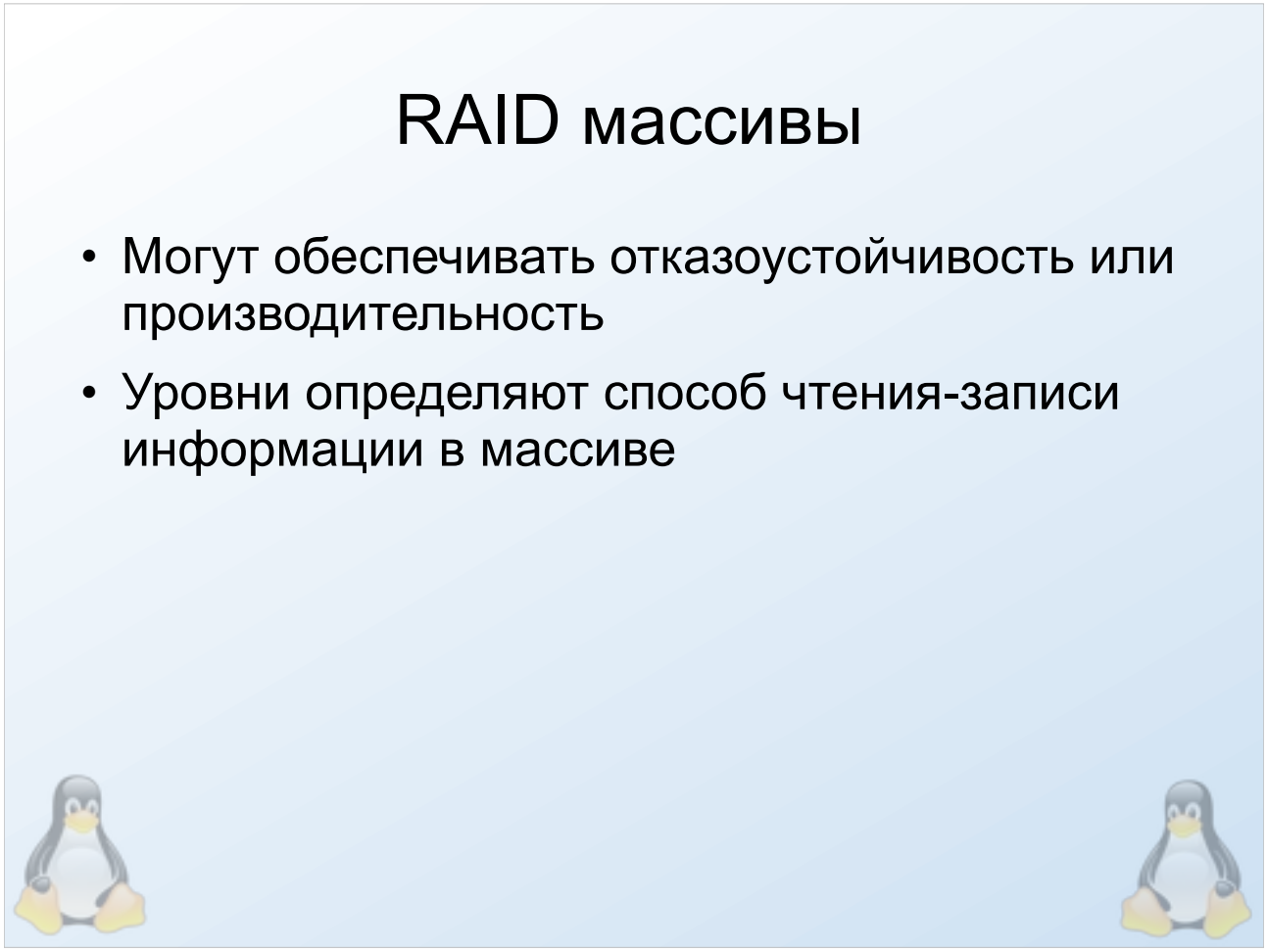
```
resize2fs /dev/testvg/firstlv 990M
```

Примечание: Только для размонтированной файловой системы ext2:

```
lvreduce -L1G /dev/testvg/firstlv
resize2fs /dev/testvg/firstlv
```

2.2. RAID массивы.

2.2.1. Использование RAID.



1. Избыточные массивы независимых (недорогих) дисков (Redudant Array of Independent (Inexpensive) Disks) применяются в нескольких случаях:
   1. при необходимости обеспечения надежного хранения данных;
   2. при необходимости повышения производительности дисковой подсистемы;
   3. для обеспечения возможности горячей замены дисков в массиве.
2. Массивы RAID могут быть организованы как аппаратно с помощью специальных контроллеров, так и программно.
3. Варианты организации RAID массивов, предназначенные для выполнения определенного класса задач, отличающиеся организацией, называются уровнями RAID. Наиболее известны следующие уровни:

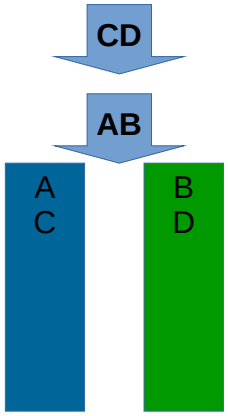
RAID массивы

- Линейный RAID (RAID -1)
  - Составной том — простое объединение нескольких дисков в общее пространство

1. LINEAR (RAID -1) — линейный или составной, простое объединение дискового пространства из нескольких устройств. Объединяемые разделы могут быть разного размера.

RAID массивы

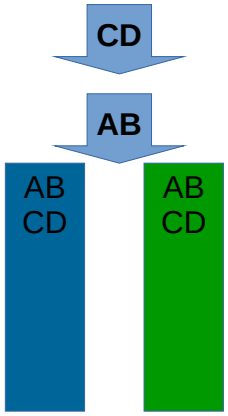
- Чередующийся том (RAID0)
  - Чередование повышает скорость операций чтения-записи
  - Дисковое пространство разбивается на страйпы



2. RAID 0 - для обеспечения высокой производительности дисковой подсистемы, необходимой, например, в системах видеомонтажа;

RAID массивы

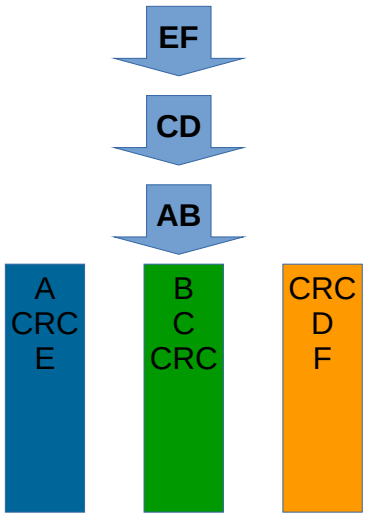
- Зеркальный том (RAID1)
  - Данные дублируются на дисках
  - Можно повысить скорость считывания данных



3. RAID 1 - для обеспечения зеркалирования дисков;
4. RAID 2 - при необходимости использования параллельного доступа с коррекцией ошибок (ECC);
5. RAID 3, обеспечивающий эффективный последовательный доступ к данным и механизм контроля четности с помощью функции XOR;
6. RAID 4, состоящий из нескольких дисков с данными и общего диска для контроля четности с помощью функции XOR;

RAID массивы

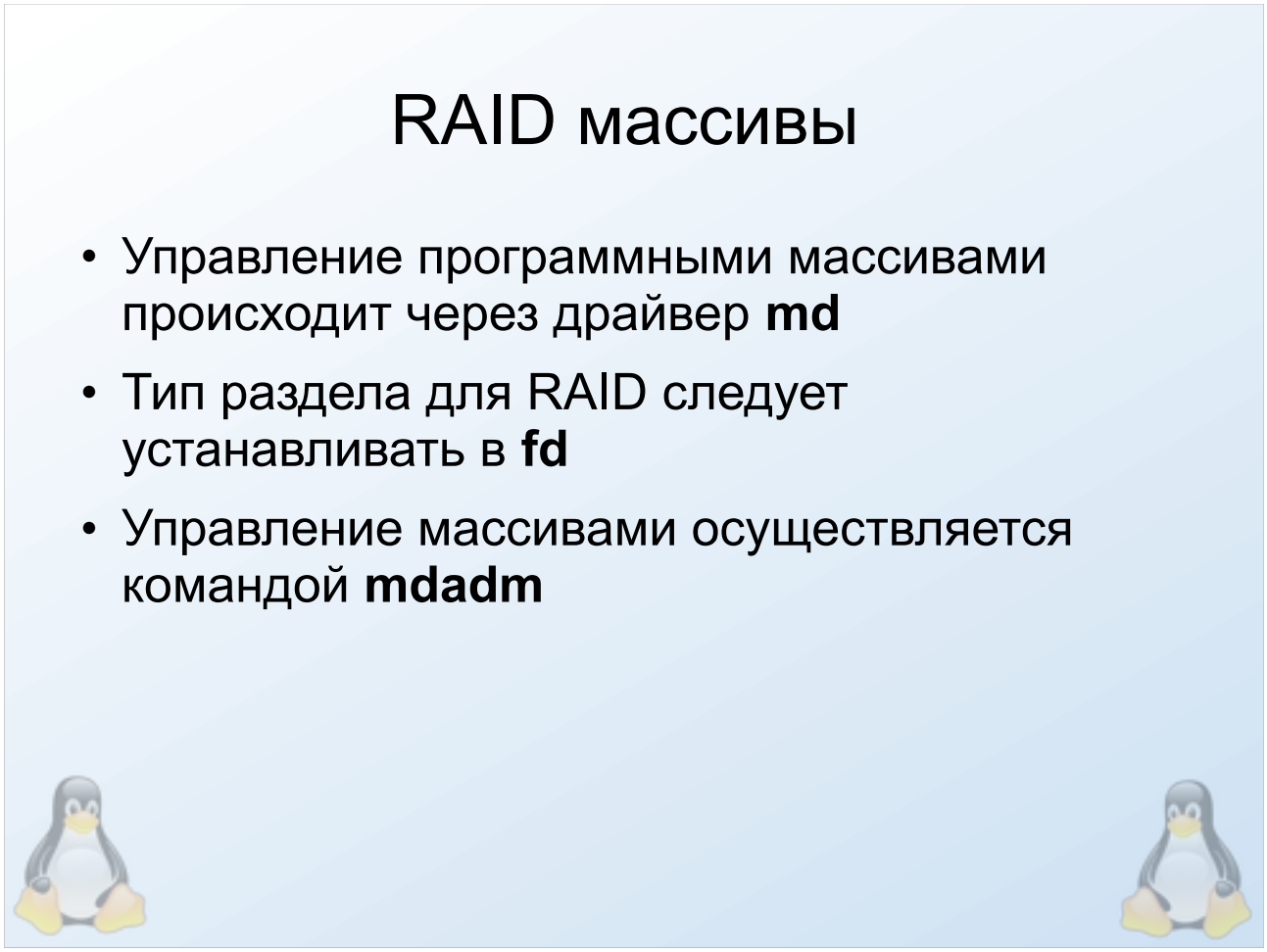
- Чередующийся том с четностью (RAID5)
  - Повышенная отказоустойчивость при меньших затратах чем в зеркале



7. RAID 5, состоящий из нескольких дисков, на каждом из которых хранятся как данные, так и контрольные XOR суммы;
8. RAID 10 с чередованием зеркальных наборов дисков;
9. RAID 0/1 - зеркальная схема чередующихся наборов.

4. Наиболее часто используются RAID массивы 0, 1 и 5 уровней.
5. Помимо указанных выше уровней RAID существует так называемый линейный режим. Суть его заключается в последовательном объединении нескольких жестких дисков в один диск для получения устройства с суммарной емкостью всех входящих в RAID дисков.

2.2.2. Программная реализация RAID в Linux.



1. Linux поддерживает программную реализацию RAID для следующих уровней (`man (8) mdadm`) :
   1. линейный режим RAID;
   2. RAID 0;
   3. RAID 1;
   4. RAID 4;
   5. RAID 5;
   6. RAID 6;
   7. RAID 10;
   8. MULTIPATH, FAULTY и CONTAINER.
2. Программная поддержка реализована в ядре с помощью драйвера `multiple device driver (md)`.
3. Для управления RAID массивами применяется программа `mdadm`.
4. RAID массива может быть собран из разделов и жестких дисков.

5. При создании RAID из раделов следует устанавливать тип раздела `fd`.
6. Во время создания RAID массива на устройства записывается специальный блок метаданных, который содержит полное описание массива. Этот блок используется для восстановления конфигурации RAID после перезагрузки ОС или после переноса устройств на другой компьютер.
7. Метаданные записываются в конец раздела
8. `mdadm` имеет следующие режимы работы:
   1. `--create` — создание нового RAID массива;
   2. `--assemble` — сборка предварительно созданного массива;
   3. `--build` — создание и сборка массива без записи метаданных на диски;
   4. `--manage` — управление существующими массивами;
   5. `--monitor` — мониторинг работы («здоровья») массива;
   6. `--grow` — изменение размеров массива (добавление или удаление членов массива и изменение дискового пространства, которое используется на членах массива);
   7. `--incremental` — добавление нового устройства в массив;
   8. `--auto-detect` — стартует в ядре активацию любых массивов с автоопределением (этот режим работает не во всех конфигурациях);
   9. режим Misc запуск `mdadm` без выше указанных опций, используется для управления и мониторинга массивами и устройствами.
9. `mdadm` поддерживает работу с конфигурационным файлом.
10. Использование конфигурационного файла не обязательно, но он упрощает администрирование RIAD.
11. По умолчанию предполагается, что конфигурационный называется `/etc/mdadm.conf`.
12. В файле описываются устройства, под управлением RAID. Ключевое слово DEVICE. Массивы, которые используются с системе, описываются ключевым словом ARRAY

Пример:

```
DEVICE          /dev/sda1 /dev/sdb1 /dev/sdc1 /dev/sdd1
ARRAY           /dev/md0 devices=/dev/sda1,/dev/sdb1
ARRAY           /dev/md1 devices=/dev/sdc1,/dev/sdd1
```

В примере показано описание устройств, из которых можно собирать массивы, а также имена имеющихся массивов и устройства, на которых они располагаются. Остальные параметры массивов находятся в метаданных массивов.

13. Для создания записи о массиве в конфигурацион файле можно, после создания RAID массива, воспользоваться командой:

```
mdadm --detail –scan >> /etc/mdadm.conf
```

Не забудьте предварительно создать строку с описанием устройств в файле /etc/mdadm.conf.

14. Последущая «сборка» массивов описанных в конфигурационном файле осуществляется командой:

```
mdadm -A -s
```

Эта команда запускается на этапе инициализации системы.

15.Опция `--create` предназначена для создания новых массивов.

Приер:

Создание зеркала:

```
mdadm --create /dev/md0 --level=1 -n 2 /dev/hda5 /dev/hdb6
```

Создание RAID5 с размером полосы чередования 128kB.

```
mdadm -Cv /dev/md1 -l5 -n3 -c128 /dev/sdb1 /dev/sdc1 /dev/sdd1
```

16.Зеркальный том можно создать из существующего раздела, без уничтожения данных. Обязательным условием является наличие свободного места на разделе.

17.Устройство, которое содержит данные для зеркалирования, необходимо поставить первым в команде по созданию зеркала.

18.Перед созданием зеркала из существующего раздела, рекомендуется предварительно уменьшить размер файловой системы, например утилитой resize2fs. Иначе запись метаданных массива в раздел может привести к потере данных.

19.Создать RAID массивы с чередованием без уничтожения данных на разделе не возможно.

20.Просмотр состояния массива осуществляется командой:

```
mdadm --detail /dev/md0
/dev/md0:
        Version : 0.90
  Creation Time : Sun Apr 25 17:53:42 2010
     Raid Level : raid1
     Array Size : 248896 (243.10 MiB 254.87 MB)
  Used Dev Size : 248896 (243.10 MiB 254.87 MB)
   Raid Devices : 2
  Total Devices : 2
Preferred Minor : 0
    Persistence : Superblock is persistent

    Update Time : Sun Apr 25 17:53:58 2010
          State : clean, resyncing
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

 Rebuild Status : 72% complete

           UUID : 259fd1fc:40aac815:0ef6a80b:2433f718
         Events : 0.12

    Number   Major   Minor   RaidDevice State
       0       3        5        0      active sync   /dev/hda5
       1       3        6        1      active sync   /dev/hdb6
```

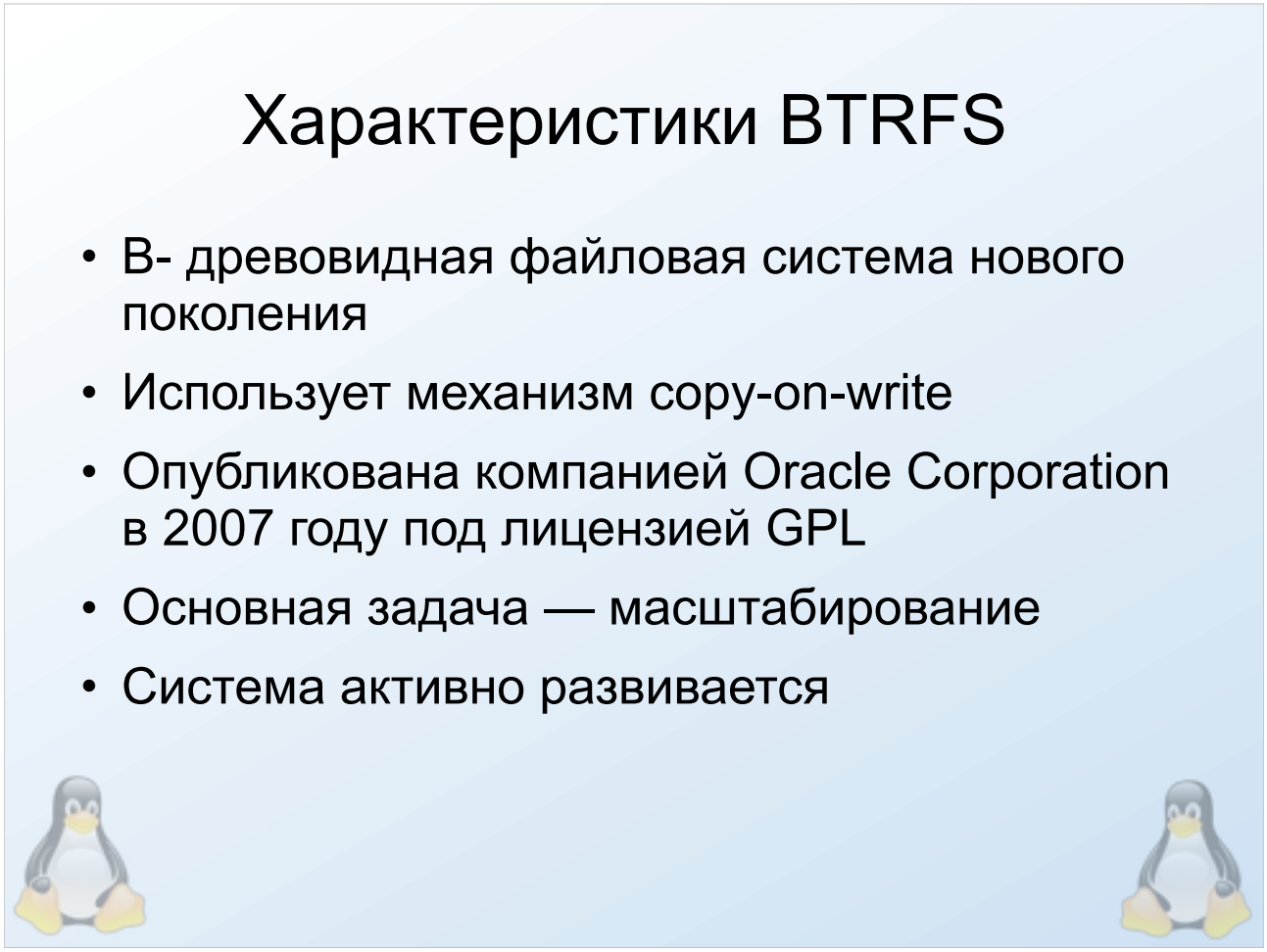
21.Замена рабочего диска в массиве, который эту операцию поддерживает:

Пример:

```
# mdadm /dev/md0 --fail /dev/hdb6 --remove /dev/hdb6 --add /dev/hdb7
mdadm: set /dev/hdb6 faulty in /dev/md0
mdadm: hot removed /dev/hdb6
mdadm: added /dev/hdb7
```

Глава 3. BTRFS.

3.1. Характеристики BTRFS.

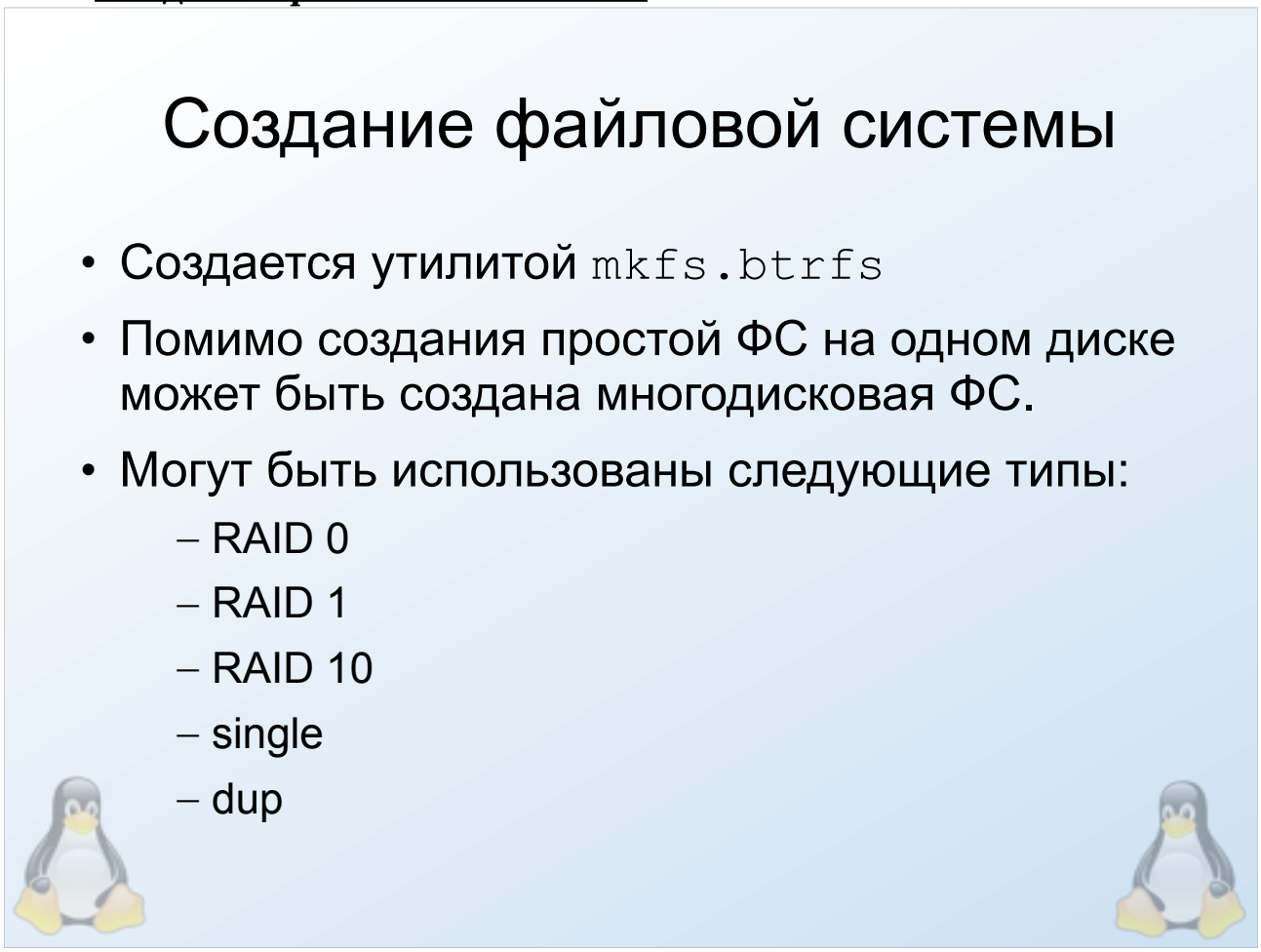


1. BTRFS (B-tree FS, «Better FS» или «Butter FS») — файловая система (ФС) для Linux, основанная на структурах Б-деревьев.
2. Работает по принципу «копирование при записи» (copy-on-write).
3. Опубликована компанией Oracle Corporation в 2007 году под лицензией GNU General Public License (GPL).
4. Формат ФС был зафиксирован 12 июня 2010 года.
5. Основная задача — позволить ФС масштабироваться под вновь появляющиеся системы хранения. Масштабирование — это не просто возможность адресовать доступное пространство, это возможность управлять ФС с помощью простого интерфейса. Это сделает ФС более надёжной, и позволит видеть используемые ресурсы.
6. Основные возможности на 2016 год:
    1. Хранение файлов в extent
    2. Максимальный размер файла 2^64 byte == 16 EiB (практический лимит 8 EiB из-за Linux VFS)

3. Эффективная упаковка маленьких файлов и индексация каталогов
4. Динамическое выделение inode
5. Записываемые snapshots и read-only snapshots
6. Subvolumes с квотами
7. Контрольные суммы данных и метаданных
8. Сжатие (zlib и LZO)
9. Многодисковое хранилище
10. File Striping, File Mirroring, File Striping+Mirroring, Striping с Single и Dual Parity
11. Оптимизация для SSD
12. Эффективное инкрементальное архивирование
13. Фоновый процесс поиска и устранение ошибок файлов
14. Онлайновая дефрагментация
15. Оффлайновая проверка ФС
16. Конвертация из ext3/4
17. Затравочные устройства, которые используются как исходные данные
18. Пакетная или внешняя дедупликация (работает после записи данных)

7. Планируемые возможности:
   1. Быстрая оффлайн проверка ФС
   2. Онлайн проверка ФС
   3. Зеркалирование и чередование на уровне объектов
   4. Альтернативные алгоритмы контрольных сумм и сжатия данных
   5. Встроенная дедупликация (на этапе записи)
   6. «Горячее» отслеживание и зеркалирование на быстрые диски

3.2. Создание файловой системы.



1. Простое создание файловой системы — `mkfs.btrfs`
   Пример:

```
# mkfs.btrfs /dev/sdb1
```

2. Создание многодисковых файловых систем:

```
# mkfs.btrfs /dev/disk1 /dev/disk2 /dev/disk3 /dev/disk4
```

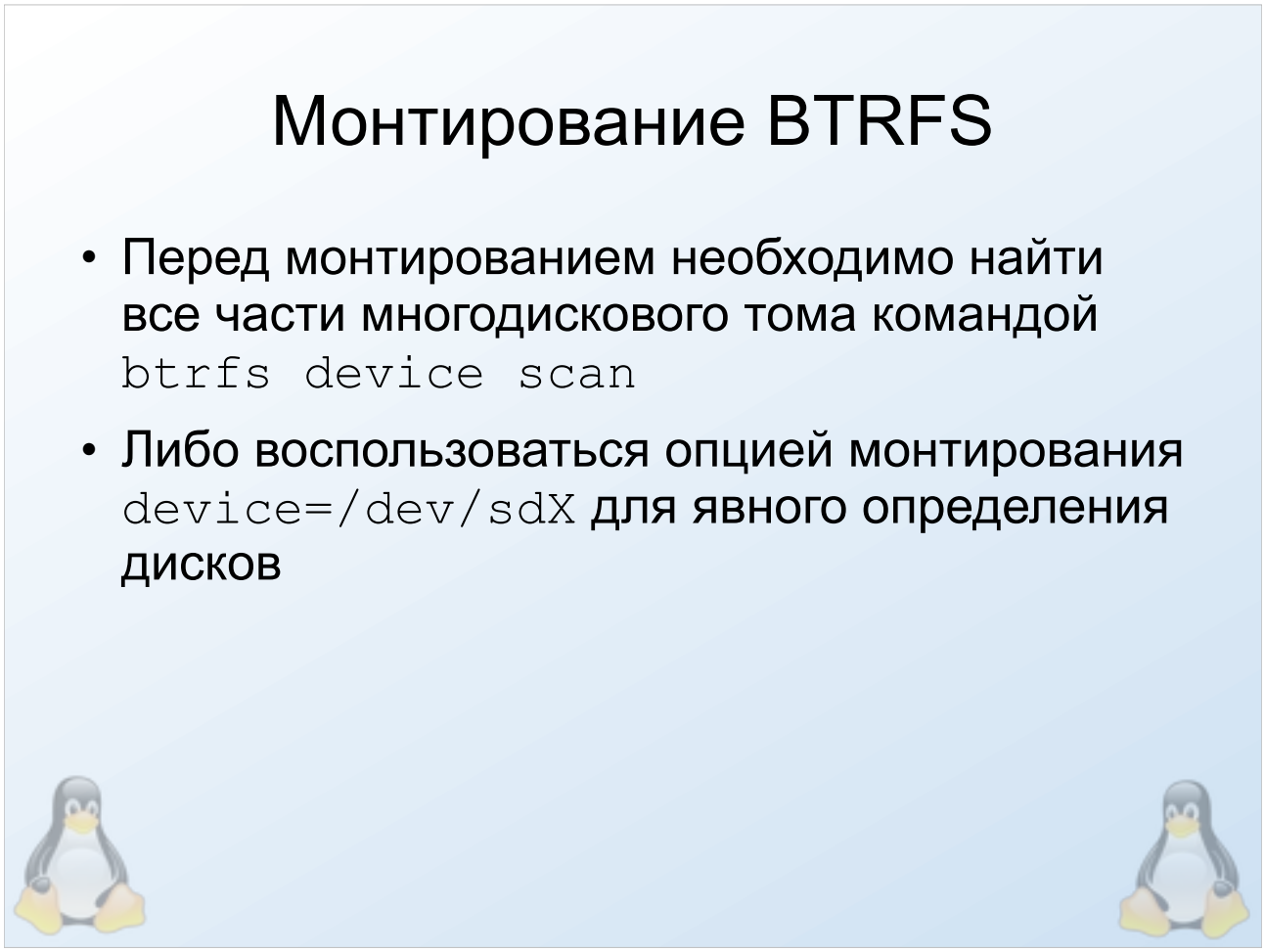
Примечание: Здесь создается ФС с зеркалированием метаданных и чередованием данных

3. Опции `-m` и `-d` определяют тип RAID для, соответственно, метаданных и данных.
4. Возможны следующие типы:
   1. `raid0`
   2. `raid1`
   3. `raid10`
   4. `dup` — дублирование метаданных
   5. `single` — не использовать дублирование или чередование, например, полезно при использовании на SSD дисках отключать дублирование метаданных

```
# mkfs.btrfs -d single /dev/disk1 /dev/disk2 /dev/disk3
```

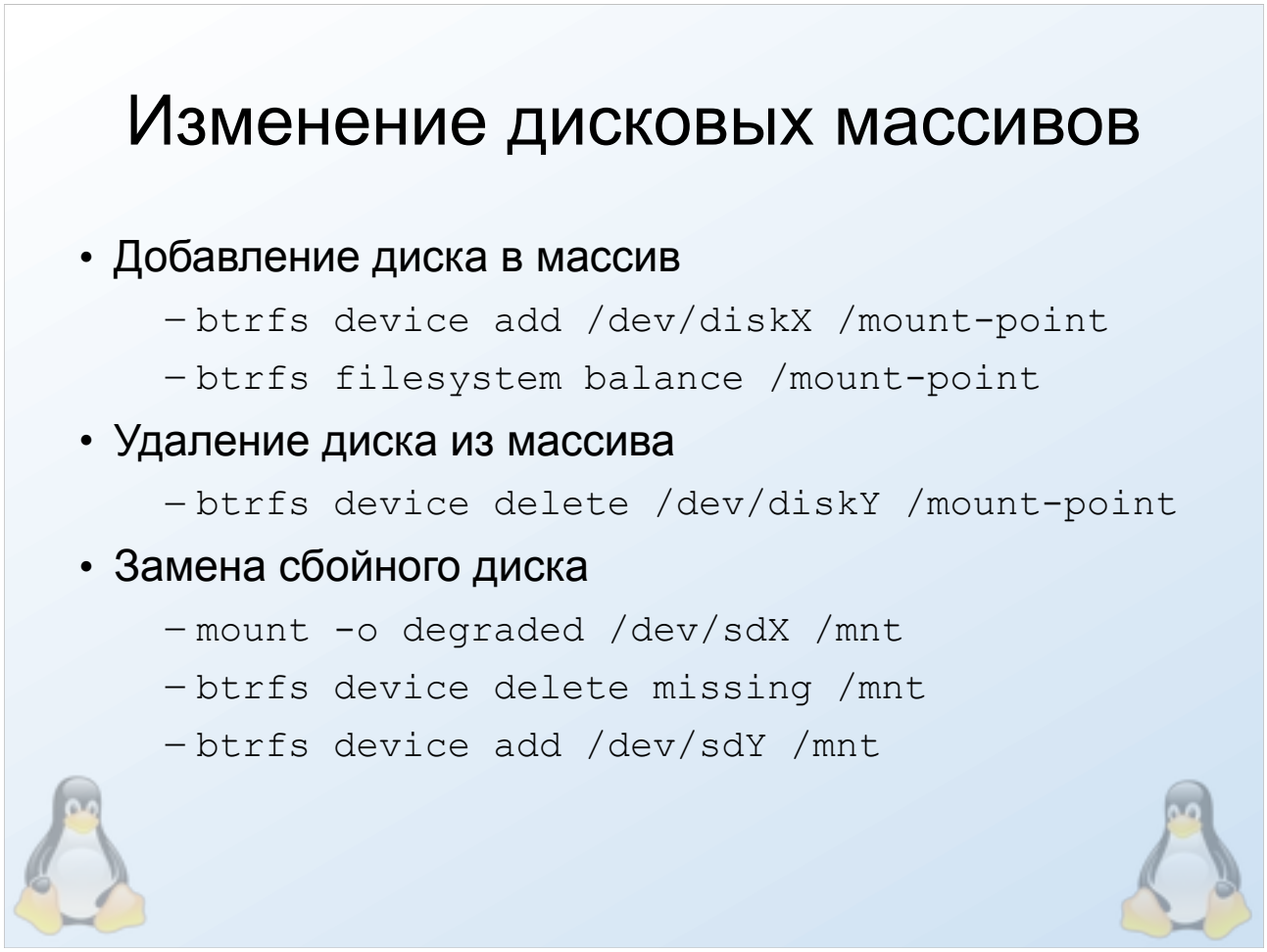
Примечание: в примере выше не используется ни чередование, ни зеркалирование, поэтому разделы могут иметь разный размер.

3.3. Монтирование BTRFS



1. Команда `btrfs device scan` используется для поиска всех многодисковых массивов BTRFS.
2. В случае монтирования без предварительного использования команды `btrfs device scan` можно применить опцию монтирования `device=/dev/sd`*`X`*
3. Монтирование SSD дисков полезно проводить с опциями:
   1. `ssd` — оптимизирует работу файловой системы и ядра и твердотельным диском
   2. `ssd_spread` — оптимизирует свободное место на диске
   3. `discard` — включает механизм discard/TRIM для свободных блоков. Для некоторых дисков может замедлить работу. Альтернатива опции — команда `fstrim`

3.4. Изменение дисковых массивов



1. Добавление устройства в массив возможно только тогда, когда он подмонтирован.
2. Команда `btrfs device add` добавляет диск к массиву:

```
# btrfs device add /dev/diskX /mount-point
```

3. Простого добавления диска не достаточно, необходимо еще перераспределить данные с учетом нового носителя:

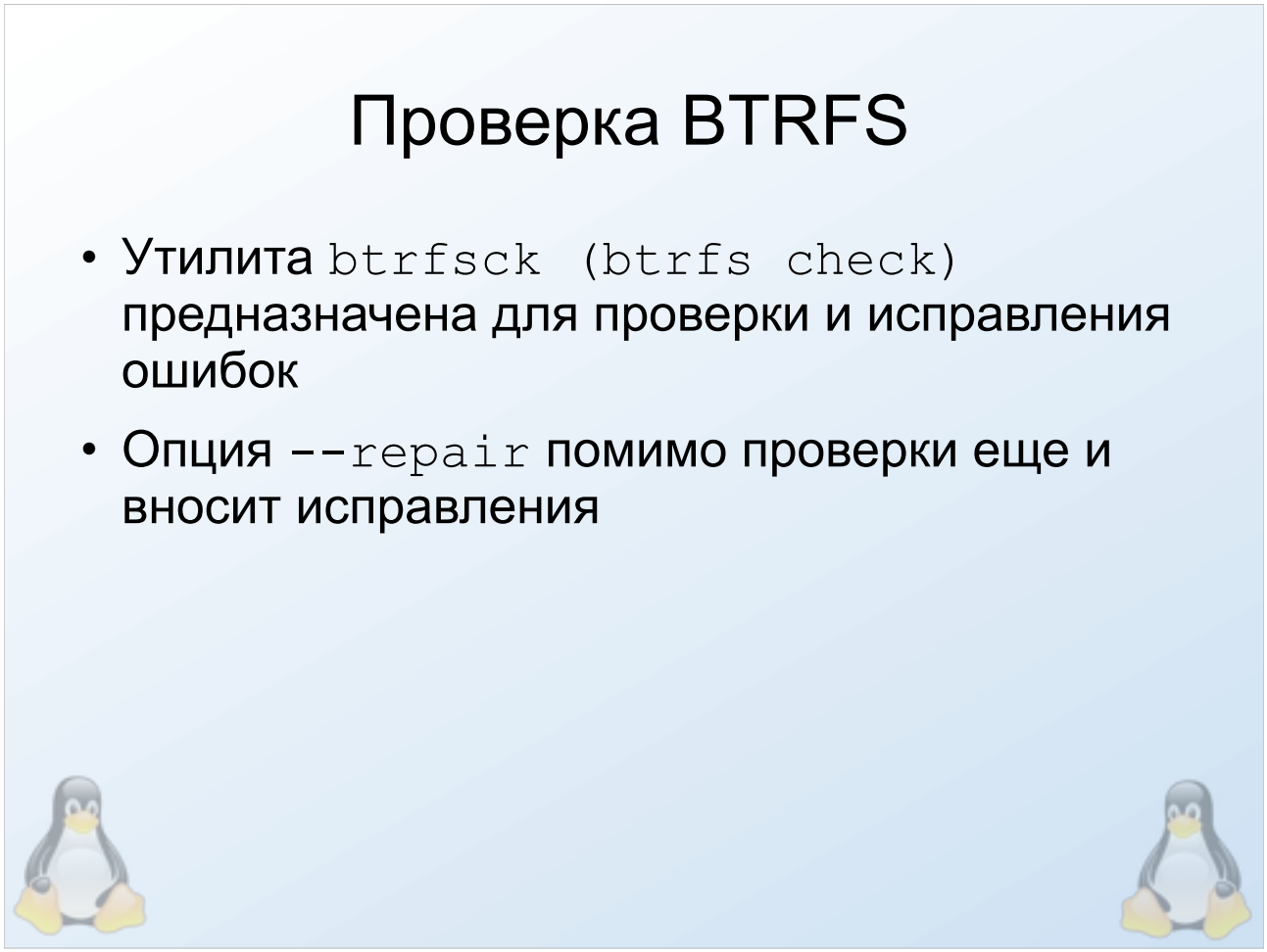
```
# btrfs filesystem balance /mount-point
```

4. Удаление диска из массива:

```
# btrfs device delete /dev/diskY /mount-point
```

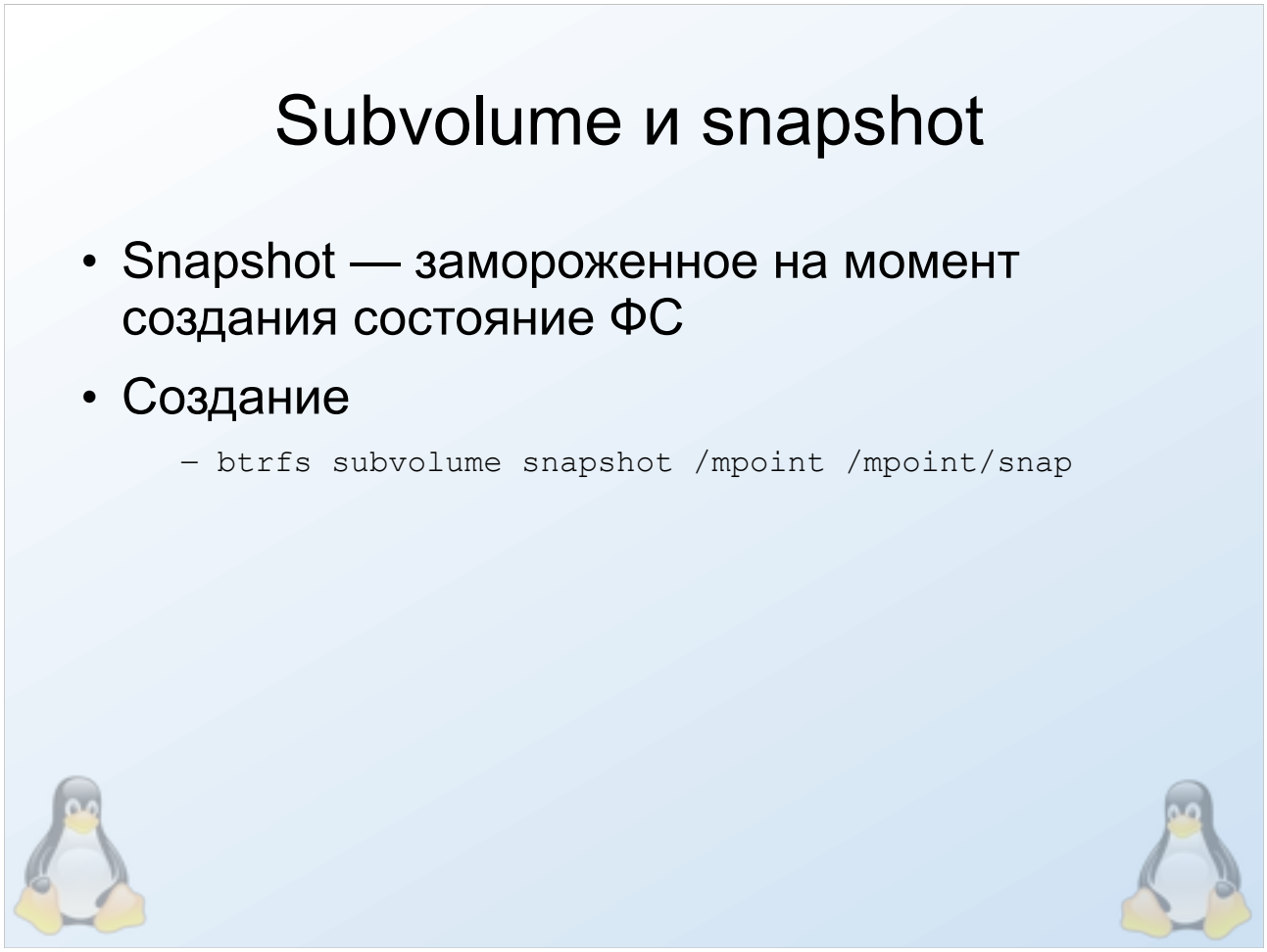
5. Замена сбойного диска:
    1. Монтируем массив с опцией degraded:
       `# mount -o degraded /dev/sdX /mnt`
    2. Удаляем отсутствующие устройства
       `# btrfs device delete missing /mnt`
    3. Добавляем новые
       `# btrfs device add /dev/sdY /mnt`

3.5. Проверка BTRFS



1. Утилита `btrfsck (btrfs check)` предназначена для проверки и исправления ошибок в BTRFS
2. По-умолчанию `btrfsck` не исправляет ошибки. Опция `--repair` помимо проверки еще и вносит исправления.

3.6. Subvolume и snapshot



1. Subvolume в BTRFS являются частью (sub-tree) основного дерева (tree) файловой системы. Они создаются внутри существующей файловой системы, но ведут себя подобно самостоятельным файловым системам, со своими собственными точками монтирования, опциями и правами.
2. В отличие от создания диска, разделенного на несколько разделов, subvolume не требуют для себя дополнительного дискового пространства; они представляют из себя просто пустую директорию. Но как только вы начнете сохранять в них данные, размер subvolume начнет расти прямо пропорционально объему добавляемых в них файлов.
3. Преимущество такого подхода не только в экономии места на диске; это означает, что вы можете создать столько подтомов в одной файловой системе, сколько вам необходимо. И присоединять дополнительные съемные устройства к subvolume, когда устройства будут заполнены, вне зависимости от того, какой том имеет наибольший объем.
4. Создание subvolume:

```
# btrfs subvolume create /path/to/volume/volume_name
```

5. Если вы не укажете явным образом путь в файловой системе для subvolume, то он будет создан в текущей директории.
6. Вы можете смонтировать subvolume куда угодно, указав опцию subvol команде

монтирования mount:

```
# mount -o subvol=subvol1 /dev/sdb1 /mnt
# df -h | grep sdb1
/dev/sdb1         150G        71G         79G         48% /
/dev/sdb1         150G        71G         79G         48% /mnt
```

7. Для просмотра имеющихся subvolume, можно воспользоваться командой:

```
# btrfs subvolume list /
ID 335 gen 5336 top level 5 path subvol1
```

8. Удаление subvolume:

```
# btrfs subvol delete /subvol1
```

9. Вы можете создать столько subvolume, сколько вам необходимо, и все в одной файловой системе.
10. BTRFS поддерживает специальный тип подтомов, необходимый для сопровождения файловой системы — снимки (snapshot).
11. Синтаксис, в сущности, тот же самый; нужно только заменить опцию create опцией snapshot.

```
# btrfs subvolume snapshot / /snap_01.01.16
```

12. Созданный снимок не занимает на диске дополнительное место, до тех пор пока вы не измените данные в исходном дереве.
13. Существует еще одна возможность, основанная на свойстве copy-on-write - конвертирование файловой системы ext3/ext4 в BTRFS.
14. Утилита `btrfs-convert` создает файловую систему BTRFS, переписывая существующую ext3/ext4 путем чтения ее и создания b-деревьев на свободном дисковом пространстве. Как и при создании снимков, эта вторая файловая система не занимает дополнительного места, если файлы не изменяются. Когда же файлы изменены, оригинальная версия файловой системы ext сохраняется, так что вы можете "откатить" назад весь процесс конвертирования и восстановить файловую систему в ее прежнем, до BTRFS, состоянии.
15. До конвертирования необходимо запустить утилиту `fsck` для проверки файловой системы на возможные ошибки. После этого можно запускать конвертацию.

```
# btrfs-convert device
```

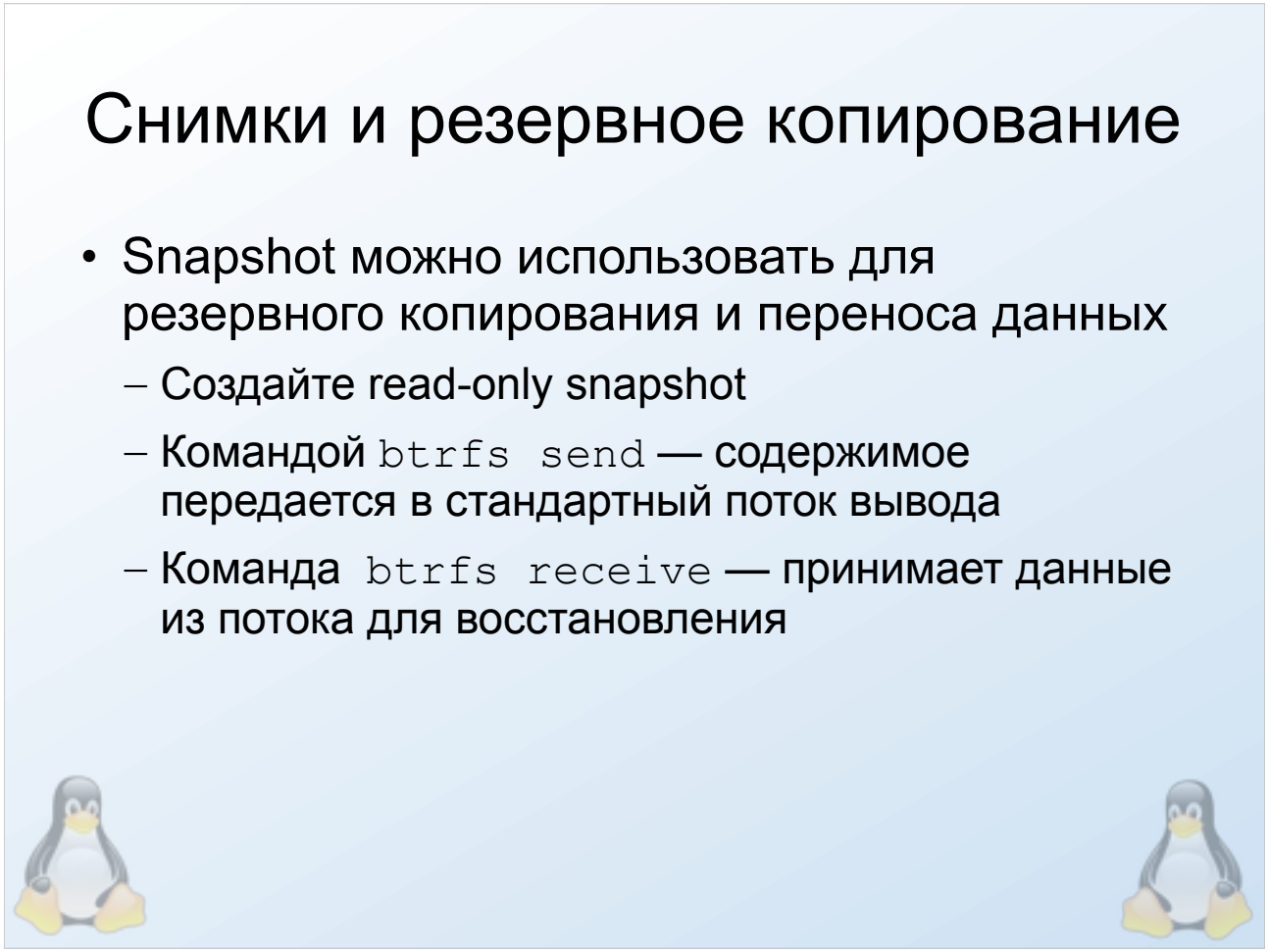
16. Оригинальная ext-система сохраняется в виде снимка с именем `ext2_saved` (даже если она имела формат ext3 или ext4). Вы также можете примонтировать снимок с помощью команды:

```
# mount -t btrfs -o subvol=ext2_saved device /mnt/ext2_saved
```

17. Вернуть снимок файловой системы ext в первоначальное состояние (до изменений) вы можете так:

```
# btrfs-convert -r device
```

3.7. Использование snapshot для копирования данных



1. Снимки могут быть использованы для создания резервных копий или переноса данных.
2. Процедура создания резервной копии
    1. Создается read-only (**!**) snapshot из subvolum'а или всей файловой системы:

```
# btrfs subv snap -r /sub1 /sub1_ro
```

Примечание: Приложения которые работают с данными в subvolume не заметят такой операции и продолжат работать как и прежде.

    2. Командой `btrfs send` содержимое снимка направляется в стандартный поток вывода:

```
# btrfs send /sub1_ro | gzip -c - > sub1_backup.gz
```

Примечание: Не забывайте после создания резервной копии удалять снимки.

    3. Командой `btrfs receive` данные можно восстановить:

```
# zcat sub1_backup.gz | btrfs receive /sub1_restored_ro
```

    4. Следует учесть, что восстановленный subvolume будет read-only. Если вы его хотите использовать, то необходимо создать из него снимок, а read-only subvolume удалить.

Пример: восстанавливаем исходный subvolume.

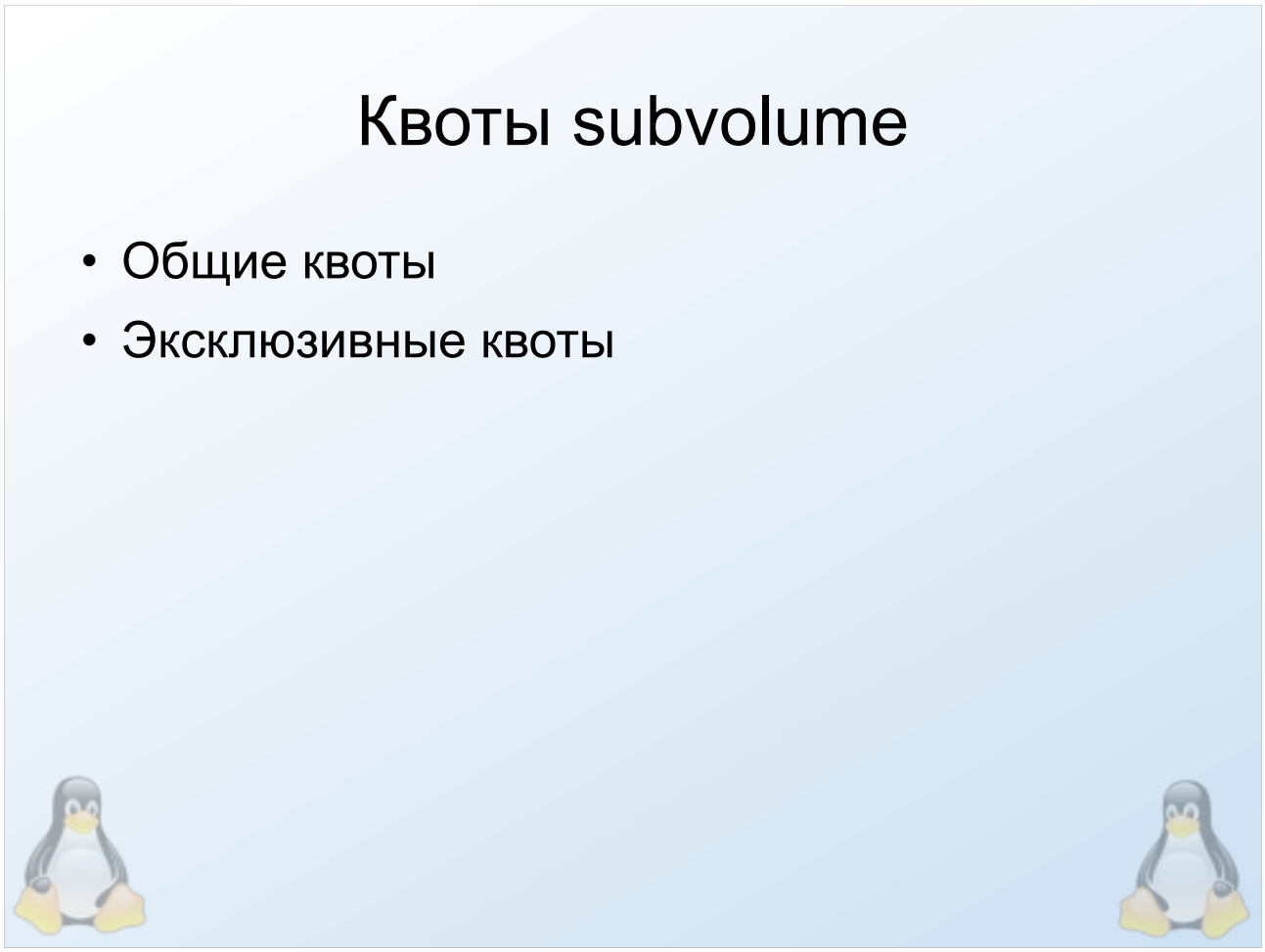
```
# btrfs subv del -c /sub1
# btrfs subv snap /sub1_restored_ro /sub1
# btrfs subv del -c /sub1_restored_ro
```

3. Для переноса данных, например на другой диск необходимо объединить команды `btrfs send` и `btrfs receive` в конвейер.

Пример:

```
# btrfs send /sub1_ro | btrfs receive /new_path_to_subv1_ro
```

3.8. Квоты subvolume



1. Для subvolume вы можете настроить квоты.
2. Квоты в btrfs пока не являются достаточно стабильными и рекомендованными к использованию. Цитата из `man btrfs-quota`:

```
PERFORMANCE IMPLICATIONS

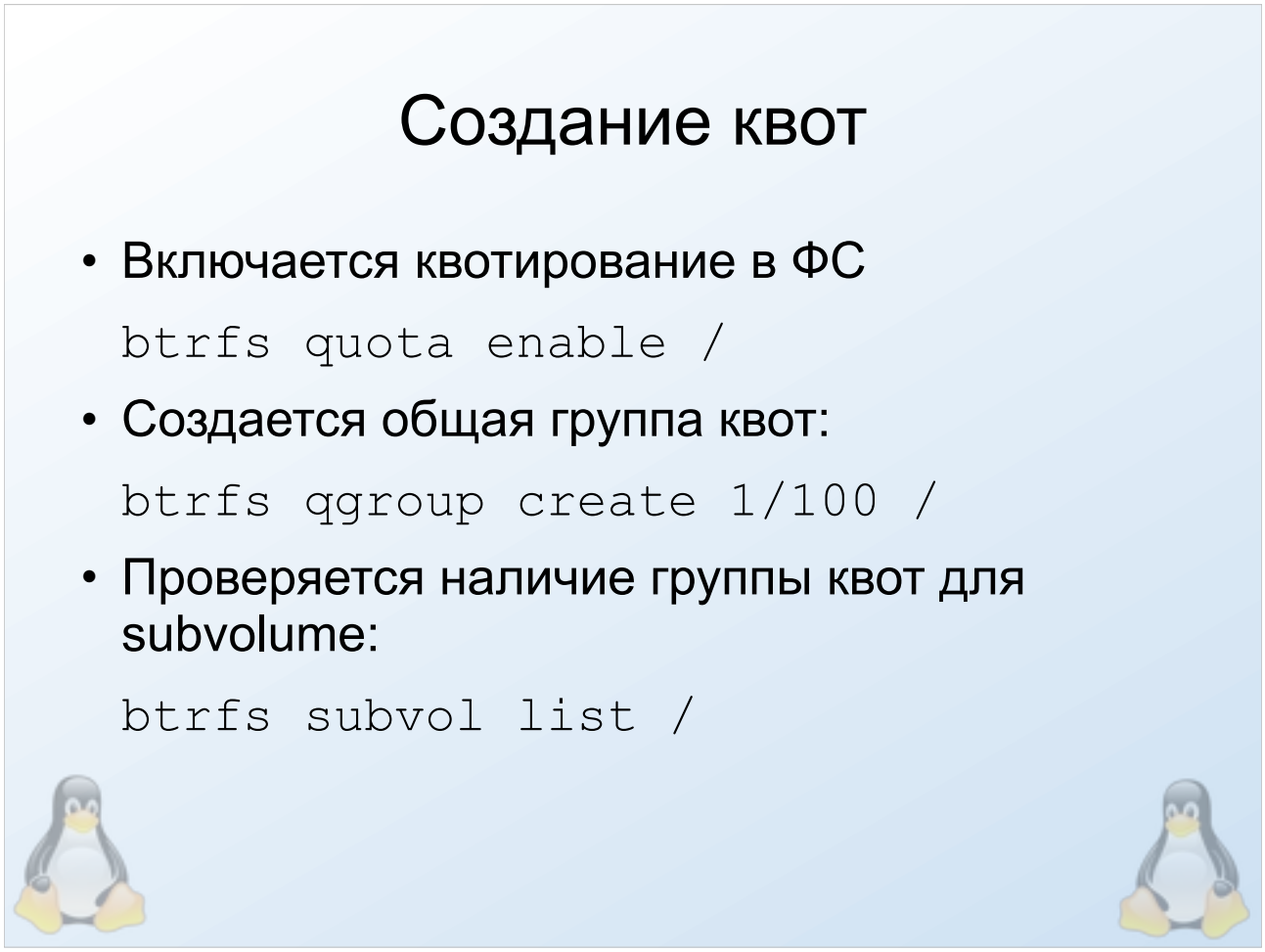
    When quotas are activated, they affect all extent processing, which
    takes a performance hit. Activation of qgroups is not recommended
    unless the user intends to actually use them.


STABILITY STATUS

    The qgroup implementation has turned out to be quite difficult as it
    affects the core of the filesystem operation. Qgroup users have hit
    various corner cases over time, such as incorrect accounting or system
    instability. The situation is gradually improving and issues found and
    fixed.
```

3. При настройке квот используется концепция группы квот.

4. Система квот состоит из двух уровней:
    1. Общие (**shared**) квоты — ограничение действующее на несколько subvolume, которые объединены этой группой квот.
    2. Эксклюзивные (**exclusive**) квоты — ограничение для отдельного снимка.



5. Процесс настройки квот состоит из следующих шагов:
   1. Включается квотирование в ФС

      ```
      # btrfs quota enable /
      ```

   2. Создается общая группа квот:

      ```
      # btrfs qgroup create 1/100 /
      ```

Примечание: Здесь 1/100 идентификатор группы. Может быть любой 1/XXX.

   3. Проверяется наличие группы квот для subvolume:

      ```
      # btrfs subvol list /
      ID 338 gen 5358 top level 5 path subvol2
      ID 339 gen 5362 top level 5 path subvol1
      ID 340 gen 5363 top level 5 path snap_01.01.16

      # btrfs qgroup show  -p /
      qgroupid         rfer         excl parent
      --------         ----         ---- ------
      ```

   4. ```
      0/5          69.73GiB    128.00KiB ---
      0/338        15.00MiB     15.00MiB ---
      ```
      ```

```
0/340         69.73GiB     80.00KiB ---
1/100            0.00B        0.00B ---
```

Примечание: В команде `btrfs qgroup show` использовалась опция -p, чтобы выявить зависимости групп квот. В примере выше этих зависимостей нет.

5. Если группы квот для subvolume нет, то ее можно создать командой:

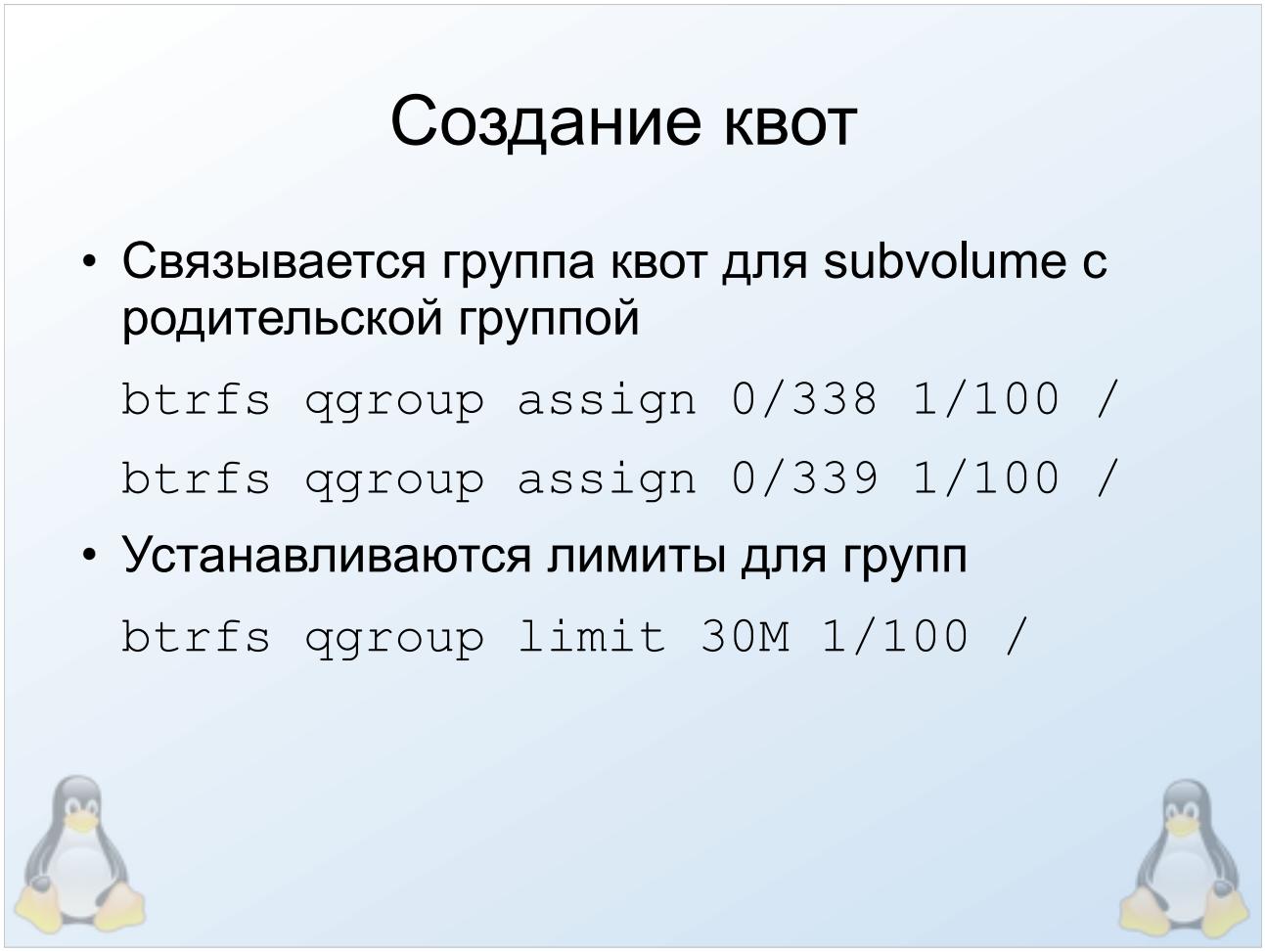
```
# btrfs qgroup create 0/339 /
# btrfs qgroup show  -p /
qgroupid         rfer         excl parent
--------         ----         ---- ------
0/5           69.73GiB    112.00KiB ---
0/338         15.02MiB     15.02MiB ---
0/339            0.00B        0.00B ---
0/340         69.73GiB    112.00KiB ---
1/100         15.02MiB     15.02MiB ---
```

Примечание: обратите внимание на группу 0/339, показано, что в ней нет данных, но это может быть не так.

```
# btrfs quota rescan /
```

Примечание: дождитесь окончания сканирования. Если сканирование не закончено, то после команды `btrfs qgroup show`, вы увидите сообщение: `WARNING: Rescan is running, qgroup data may be incorrect`

```
# btrfs qgroup show  -p /
qgroupid         rfer         excl parent
--------         ----         ---- ------
0/5           69.73GiB    128.00KiB ---
0/338         15.00MiB     15.00MiB ---
0/339          4.52MiB      4.52MiB ---
0/340         69.73GiB     80.00KiB ---
1/100            0.00B        0.00B ---
```



6. Свяжите группы квот для subvolume с родительской группой

```
# btrfs qgroup assign 0/338 1/100 /
# btrfs qgroup assign 0/339 1/100 /
# btrfs qgroup show  -p /
qgroupid         rfer         excl parent
--------         ----         ---- ------
0/5          69.73GiB    112.00KiB ---
0/338        15.02MiB     15.02MiB 1/100
0/339         4.52MiB      4.52MiB 1/100
0/340        69.73GiB    112.00KiB ---
1/100        19.53MiB     19.53MiB ---
```

7. Установите квоты для групп

```
# btrfs qgroup limit 30M 1/100 /
# btrfs qgroup show  -rep /
qgroupid  rfer      excl      max_rfer  max_excl  parent
--------  ----      ----      --------  --------  ------
0/5       69.74GiB  112.00KiB none      none      ---
0/338     15.02MiB  15.02MiB  none      none      1/100
0/339     4.52MiB   4.52MiB   none      none      1/100
0/340     69.73GiB  112.00KiB none      none      ---
```

```
1/100     19.53MiB  19.53MiB  30.00MiB  none      ---

# btrfs qgroup limit -e 10M 0/339 /
# btrfs qgroup show  -rep /
qgroupid  rfer      excl      max_rfer  max_excl  parent
--------  ----      ----      --------  --------  ------
0/5       69.74GiB  112.00KiB none      none      ---
0/338     15.02MiB  15.02MiB  none      none      1/100
0/339     4.52MiB   4.52MiB   none      10.00MiB  1/100
0/340     69.73GiB  112.00KiB none      none      ---
1/100     19.53MiB  19.53MiB  30.00MiB  none      ---
```

Примечание: В первом случае установлен общий лимит 30 МБ на группу 1/100. Во втором — эксклюзивный лимит 10 МБ на группу 0/339. Это означает, что если в 0/339 ничего на записывать, то 0/338 получит 30МБ. Subvolume 0/339 может использовать не более 10МБ.

8. Если вы установите эксклюзивный лимит на subvolume, а затем сделаете его снимок, то эксклюзивный лимит перейдет на снимок.

Пример:

Создаем subvolume и квоты

```
# mount /home/btrfs.img /mnt/
# cd /mnt/
# ls
# btrfs sub cre s1
Create subvolume './s1'
# btrfs sub cre s2
Create subvolume './s2'
# btrfs qu ena .
# btrfs qg cre 1/100 .
# btrfs qg assign 0/256 1/100 .
# btrfs qg assign 0/257 1/100 .
# btrfs qg limit 30M 1/100 .
# btrfs qg limit -e 10M 0/256 .
# btrfs qg sh -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/256        16.00KiB     16.00KiB         none     10.00MiB 1/100
0/257        16.00KiB     16.00KiB         none         none 1/100
1/100        32.00KiB     32.00KiB     30.00MiB         none ---
```

Заполняем данными

```
# dd if=/dev/urandom of=s1/dd1 count=8 bs=1M
8+0 записей получено
8+0 записей отправлено
```

```
 скопировано 8388608 байт (8,4 MB), 0,531934 с, 15,8 MB/с
```

Важно выполнять команду `sync` для получения достоверных сведений

```
# sync
# btrfs qg sh -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/256         8.02MiB      8.02MiB         none     10.00MiB 1/100
0/257        16.00KiB     16.00KiB         none         none 1/100
1/100         8.03MiB      8.03MiB     30.00MiB         none ---
```

Создаем снимок для s1

```
# btrfs sub snap s1 snap1
Create a snapshot of 's1' in './snap1'
# btrfs qg sh -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/256         8.02MiB     16.00KiB         none     10.00MiB 1/100
0/257        16.00KiB     16.00KiB         none         none 1/100
0/258         8.02MiB     16.00KiB         none     10.00MiB ---
1/100         8.03MiB      8.03MiB     30.00MiB         none ---
```

Сейчас расхождений в снимке и subvolume нет. Если мы начнем изменять исходные данные в s1, то эксклюзивная квота снимка начнет расходоваться. А при простом добавлении данных будет увеличиваться общая квота.

```
# dd if=/dev/urandom of=s1/dd2 count=18 bs=1M
dd: ошибка записи «s1/dd2»: Превышена дисковая квота
10+0 записей получено
9+0 записей отправлено
 скопировано 9437184 байта (9,4 MB), 0,65749 с, 14,4 MB/с
# sync
# btrfs qg sh -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/256        17.02MiB      9.02MiB         none     10.00MiB 1/100
0/257        16.00KiB     16.00KiB         none         none 1/100
0/258         8.02MiB     16.00KiB         none     10.00MiB ---
1/100        17.03MiB     17.03MiB     30.00MiB         none ---
# dd if=/dev/urandom of=s1/dd3 count=18 bs=1M
dd: ошибка записи «s1/dd3»: Превышена дисковая квота
```

```
1+0 записей получено
0+0 записей отправлено
 скопировано 802816 байт (803 kB), 0,124798 с, 6,4 MB/с
# dd if=/dev/urandom of=s2/dd2 count=18 bs=1M
dd: ошибка записи «s2/dd2»: Превышена дисковая квота
12+0 записей получено
11+0 записей отправлено
 скопировано 12451840 байт (12 MB), 0,758601 с, 16,4 MB/с
# sync
# btrfs qg sh -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/256        17.12MiB      9.12MiB         none     10.00MiB 1/100
0/257        11.89MiB     11.89MiB         none         none 1/100
0/258         8.02MiB     16.00KiB         none     10.00MiB ---
1/100        29.02MiB     29.02MiB     30.00MiB         none ---
```

Удалим некоторые данные, в том числе те, что были созданы до создания снимка

```
# rm -f s1/dd3
# rm -f s1/dd1
# sync
# btrfs qg sh -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/256         9.02MiB      9.02MiB         none     10.00MiB 1/100
0/257        11.89MiB     11.89MiB         none         none 1/100
0/258         8.02MiB      8.02MiB         none     10.00MiB ---
1/100        20.91MiB     28.91MiB     30.00MiB         none ---
```

Попробуем добавить данные в снимок

```
# dd if=/dev/urandom of=snap1/dd2 count=18 bs=1M
dd: ошибка записи «snap1/dd2»: Превышена дисковая квота
2+0 записей получено
1+0 записей отправлено
 скопировано 1048576 байт (1,0 MB), 0,183623 с, 5,7 MB/с
# sync
# btrfs qg sh -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
```

```
0/5          16.00KiB     16.00KiB          none          none ---
0/256         9.02MiB      9.02MiB          none      10.00MiB 1/100
0/257        11.89MiB     11.89MiB          none          none 1/100
0/258         9.02MiB      9.02MiB          none      10.00MiB ---
1/100        20.91MiB     28.91MiB     30.00MiB          none ---
```

Удалим снимок и subvolume s1

```
# btrfs sub del snap1/
Delete subvolume (no-commit): '/mnt/snap1'
# btrfs sub del s1
Delete subvolume (no-commit): '/mnt/s1'
```

Теперь проверим квоты

```
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB          none          none ---
0/256           0.00B        0.00B          none      10.00MiB 1/100
0/257        11.89MiB     11.89MiB          none          none 1/100
0/258           0.00B        0.00B          none      10.00MiB ---
1/100        11.89MiB     11.89MiB     30.00MiB          none ---
```

Квота-группы не удаляются вместе со снимками или subvolume. Удаление производится отдельно.

```
# btrfs qg destroy 0/256 .
# btrfs qg destroy 0/258 .
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB          none          none ---
0/257        11.89MiB     11.89MiB          none          none 1/100
1/100        11.89MiB     11.89MiB     30.00MiB          none ---
```

Теперь посмотрим, как общая квота действует на снимки

```
# btrfs qg lim 10M 0/257 .
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB          none          none ---
0/257        16.00KiB     16.00KiB     10.00MiB          none 1/100
1/100        16.00KiB     16.00KiB     30.00MiB          none ---
# dd if=/dev/urandom of=s2/dd3
dd: запись в «s2/dd3»: Превышена дисковая квота
19969+0 записей получено
```

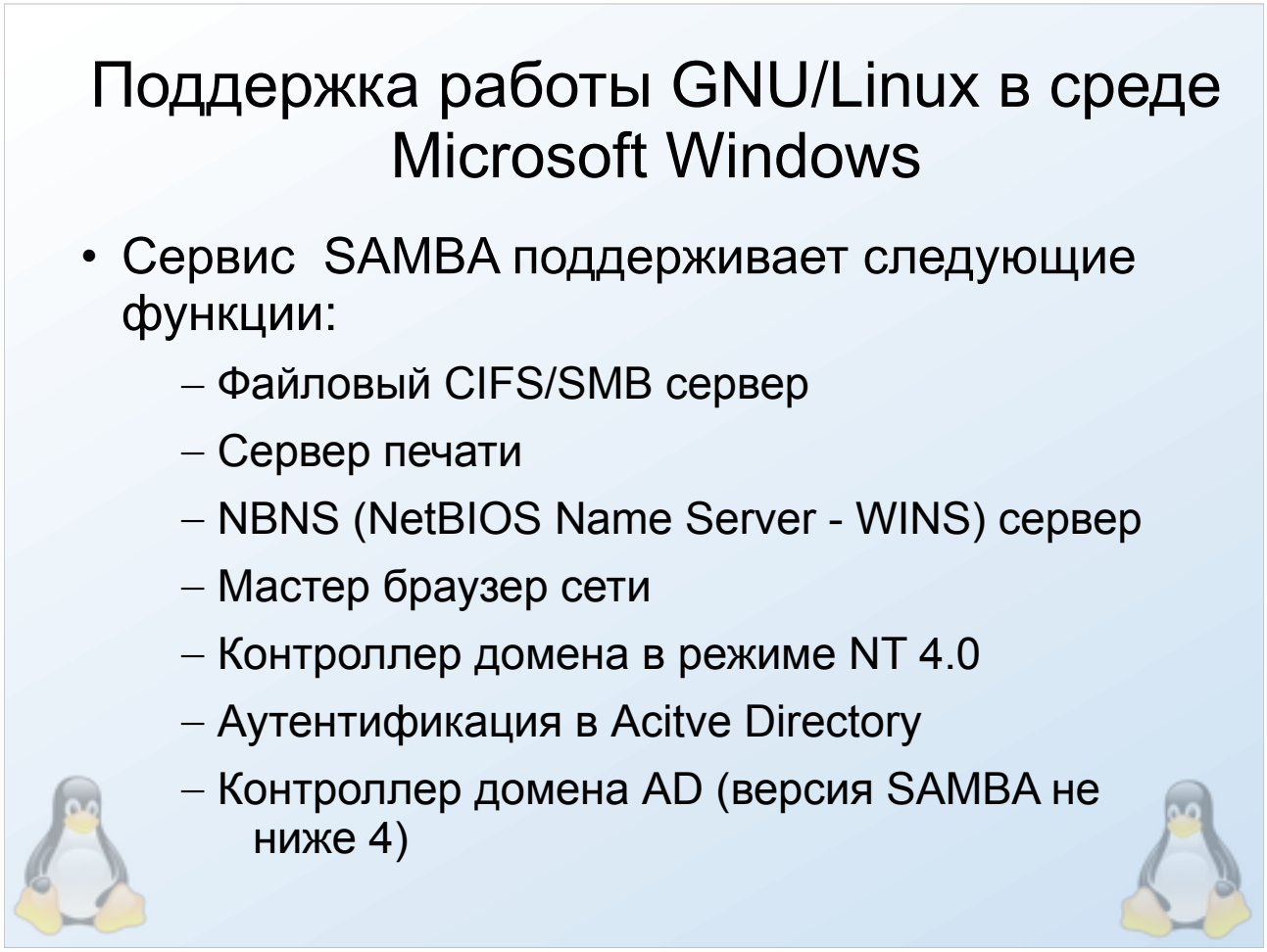
```
19968+0 записей отправлено
 скопировано 10223616 байт (10 MB), 0,688847 с, 14,8 MB/с
# sync
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/257         9.77MiB      9.77MiB     10.00MiB         none 1/100
1/100         9.77MiB      9.77MiB     30.00MiB         none ---
# btrfs sub snap s2 snap2
Create a snapshot of 's2' in './snap2'
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/257         9.77MiB     16.00KiB     10.00MiB         none 1/100
0/260         9.77MiB     16.00KiB     10.00MiB         none ---
1/100         9.77MiB      9.77MiB     30.00MiB         none ---
# dd if=/dev/urandom of=s2/dd4
dd: запись в «s2/dd4»: Превышена дисковая квота
9+0 записей получено
8+0 записей отправлено
 скопировано 4096 байт (4,1 kB), 0,00171946 с, 2,4 MB/с
# dd if=/dev/urandom of=snap2/dd4
dd: запись в «snap2/dd4»: Превышена дисковая квота
9+0 записей получено
8+0 записей отправлено
 скопировано 4096 байт (4,1 kB), 0,00178404 с, 2,3 MB/с
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/257         9.77MiB     16.00KiB     10.00MiB         none 1/100
0/260         9.77MiB     16.00KiB     10.00MiB         none ---
1/100         9.77MiB      9.77MiB     30.00MiB         none ---

# rm -f snap2/*
# rm -f s2/*
# dd if=/dev/urandom of=s2/dd5 count=5 bs=1M
```

```
5+0 записей получено
5+0 записей отправлено
 скопировано 5242880 байт (5,2 MB), 0,367809 c, 14,3 MB/c
# btrfs sub snap s2 snap22
Create a snapshot of 's2' in './snap22'
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/257         5.02MiB     16.00KiB     10.00MiB         none 1/100
0/260        16.00KiB     16.00KiB     10.00MiB         none ---
0/261         5.02MiB     16.00KiB     10.00MiB         none ---
1/100         5.02MiB     14.77MiB     30.00MiB         none ---
# rm -f s2/dd5
# sync
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/257        16.00KiB     16.00KiB     10.00MiB         none 1/100
0/260        16.00KiB     16.00KiB     10.00MiB         none ---
0/261         5.02MiB      5.02MiB     10.00MiB         none ---
1/100        16.00KiB     14.77MiB     30.00MiB         none ---
# dd if=/dev/urandom of=snap22/dd6  bs=1M
dd: ошибка записи «snap22/dd6»: Превышена дисковая квота
5+0 записей получено
4+0 записей отправлено
 скопировано 4997120 байт (5,0 MB), 0,357293 c, 14,0 MB/c
# sync
# btrfs qg sho -rep .
qgroupid         rfer         excl     max_rfer     max_excl parent
--------         ----         ----     --------     -------- ------
0/5          16.00KiB     16.00KiB         none         none ---
0/257        16.00KiB     16.00KiB     10.00MiB         none 1/100
0/260        16.00KiB     16.00KiB     10.00MiB         none ---
0/261         9.78MiB      9.78MiB     10.00MiB         none ---
1/100        16.00KiB     14.77MiB     30.00MiB         none ---
```

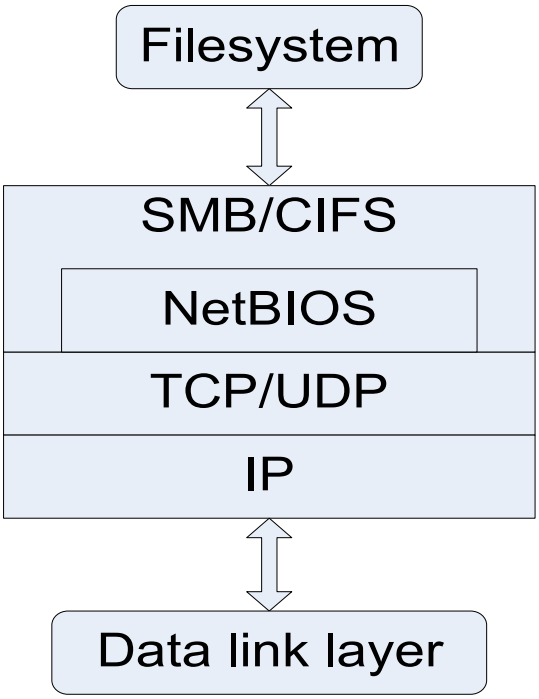
Глава 4. Разделение ресурсов с помощью SAMBA

4.1. Поддержка работы GNU/Linux в среде Microsoft Windows



1. Сетевое взаимодействие операционных систем Microsoft основано на использовании протокола NetBIOS (Network Basic Input/Output System).
2. Архитектура протокола NetBIOS не предусматривает использование маршрутизаторов.
3. В качестве конечной точки взаимодействия (службы) в сети NetBIOS используется 16-ти байтное имя NetBIOS.
4. Имя NetBIOS состоит из 15 символов в верхнем регистре (Machine name) плюс один байт для идентификации службы, работающей поверх NetBIOS.
   Пример: <00> - служба workstation, <20> - служба server.
5. Имена NetBIOS динамические. Приложения по мере необходимости регистрируют и освобождают имена.
6. Имеются имена двух типов уникальные (unique) и групповые (group).
7. В современных версиях Windows протокол NetBIOS не используется, а используется только программный интерфейс NetBIOS (NetBIOS API), который делает возможным для приложений разработанных для протокола NetBIOS работать поверх других протоколов.
8. Для работы NetBIOS поверх TCP/IP в Windows используется служба NBT (NetBIOS over

TCP/IP).



9. В NBT водится понятие типа ноды (node type), который определяет порядок разрешения имен.
10. Существует четыре типа нод:
    1. B-node (broadcast), для регистрации, разрешения и освобождения имен используются служба браузера, которая основана на широковещательных рассылках.
    2. P-node (peer), для регистрации, разрешения и освобождения имен используются сервер NetBIOS имен (NetBIOS Name Service – NBNS).

Примечание: В Windows NBNS называется Windows Internet Name Service (WINS).

    3. M-node (mixed), комбинированная. Сначала используется B-node, затем P-node.
    4. H-node (hybrid), комбинированная. Сначала используется P-node, затем B-node.
11. Для обмена файлами в сети Windows используется протоколы SMB (Server Message Block) или CIFS (Common Internet File System).
12. Протокол SMB использующий интерфейс NetBIOS, определяет серию команд, которые передаются между компьютерами.
13. SMB использует четыре типа сообщений
    1. session control
    2. file
    3. printer
    4. message
14. Протокол CIFS дальнейшее развитие SMB.
15. Основным отличием протокола CIFS от SMB, является поддержка различных схем разрешения имен, не зависящих от NetBIOS.

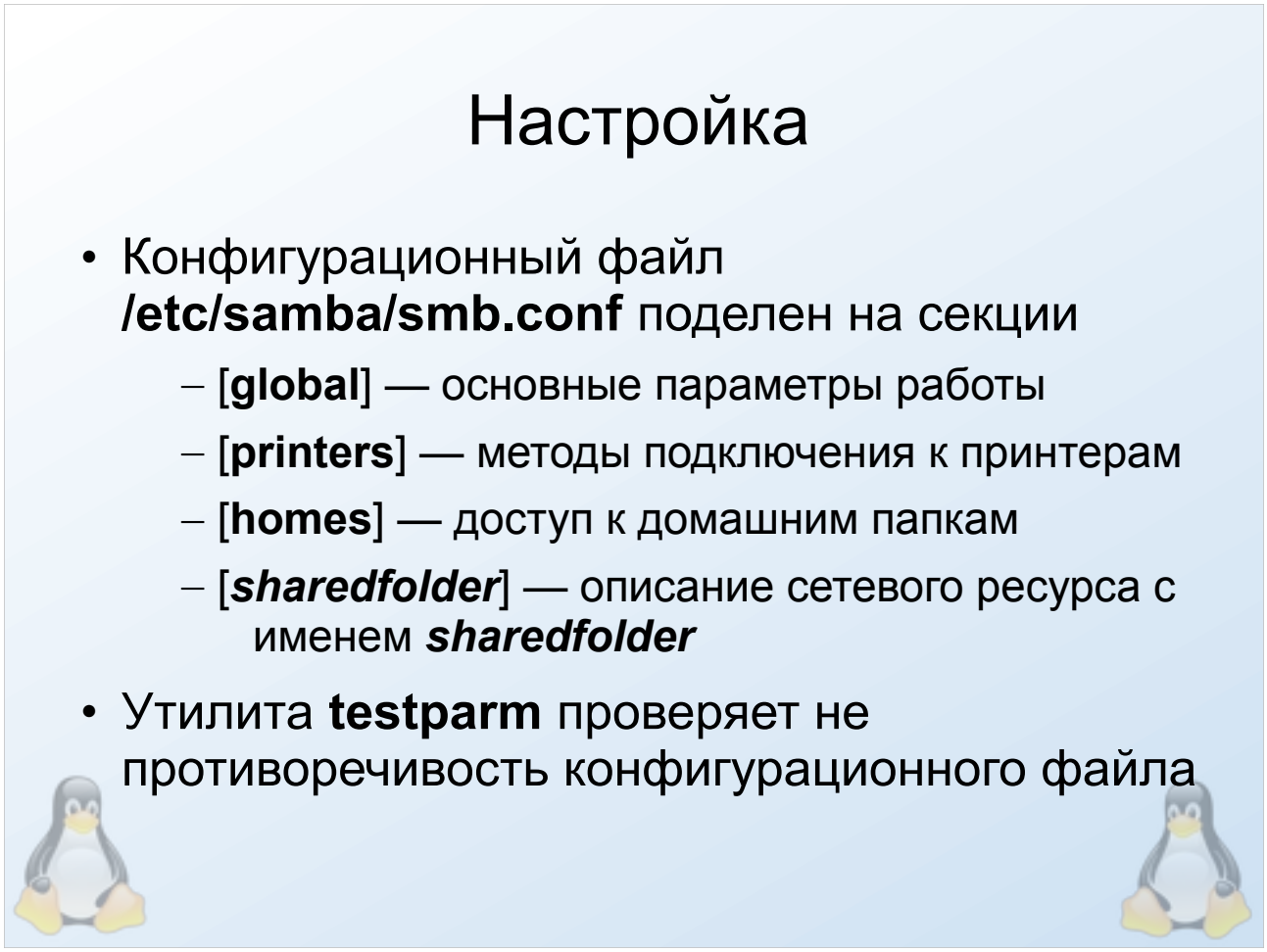
16.Поддержка протоколов SMB и CIFS настраивается в ядре в разделе `File Systems/ Network File Systems`.

17.Для реализации сервера SBM/CIFS в Linux используется пакет SAMBA.

18.В основе SAMBA лежат два демона:

1. `smbd` – предоставляет доступ к общим ресурсам
2. `nmbd` – демон регистрации и разрешения имен.

4.2. Настройка SAMBA



1. Основной конфигурационный файл, задающий поведение демонов `smbd` и `nmbd` - это `smb.conf`, который обычно находится в каталоге `/etc/samba`.

1. Файл `smb.conf` содержит секции, каждая из которых имеет свое имя, указываемое в квадратных скобках.
2. Имеются три особые секции:
    1. `global` - раздел общих настроек;
    2. `printers` - описывает методы подключения к принтерам;
    3. `homes` - настраивает доступ к домашним каталогам пользователей.
3. Остальные секции описывают общие ресурсы (shares).
4. Наиболее часто используемые параметры в глобальной секции настроек SAMBA:
    1. `netbios name` - имя компьютера в сети NetBIOS;
    2. `workgoup` - имя рабочей группы NetBIOS;
    3. `security` - задает вариант аутентификации пользователя и момента авторизации его для получения доступа к разделяемым ресурсам:

   1. `share` - права доступа к ресурсу проверяются при попытке подключения к этому ресурсу, но после подключения к компьютеру, предоставляющему эти ресурсы;
   2. `user` - права доступа проверяются в момент подключения к компьютеру, предоставляющему ресурсы (используется по умолчанию и подходит как для члена одноранговой сети Windows, так и для использования SAMBA в качестве PDC);
   3. `domain` - данный компьютер является членом домена NT и аутентификация пользователей производится на PDC (Primary Domain Controller), а не на этой машине;
   4. `server` - задает имя иного сервера аутентификации.

5. Основные параметры в описании общих папок:
   1. `comment` - необязательный комментарий, устанавливаемый на ресурс;
   2. `path` - (обязательный) путь доступа на UNIX машине к каталогу, предоставляемому в качестве разделяемого файлового ресурса, либо для принтеров – к очереди печати;
   3. `browseable` – разрешить/запретить отображать этот ресурс в сетевом окружении;
6. В описании ресурсов печати необходимо установить параметр- `printable`.

Пример: конфигурации сервера SAMBA, позволяющего ему предоставлять ресурсы в сеть Windows машин:

```
# Global parameters
[global]
        netbios name = AXIS
        workgoup = ITC
        security = user
[homes]
        comment = Home directories
        read only = No
# Files sharing
[PUB]
        comment = Public share (Guest OK!)
        path = /home/samba/pub
        read only = No
        guest ok = Yes
[ACC]
        comment = Accountant's private share. Hidden.
        path = /home/samba/accountant
        browseable = No
        valid users = accountant
# Printers sharing
[PRN1]
        comment = Company's printer.
        path = /var/spool/cups/prn1
        guest ok = Yes
        printable = Yes
```

Примечание: В приведенном выше примере в разделе `global` *установлено NetBIOS имя машины и принадлежность ее к рабочей группе NetBIOS. Также задан порядок аутентификации и авторизации - права доступа к ресурсам этой машины будут проверяться при попытке доступа к ней.*

Для каждого разделяемого ресурса в файле `smb.conf` *устанавливается собственный раздел с именем этого ресурса. В этом примере имеется два файловых ресурса:*

1. `PUB` *- для публичного доступа;*
2. `ACC` *- только для пользователя* `accountant`.

Причем для `PUB` *разрешен доступ для записи, а для* `ACC` *- нет. Более того, для ресурса* `ACC` *запрещено его отображение в сетевом окружении.*

Здесь задан также и разделяемый ресурс печати - сетевой принтер `PRN1`.

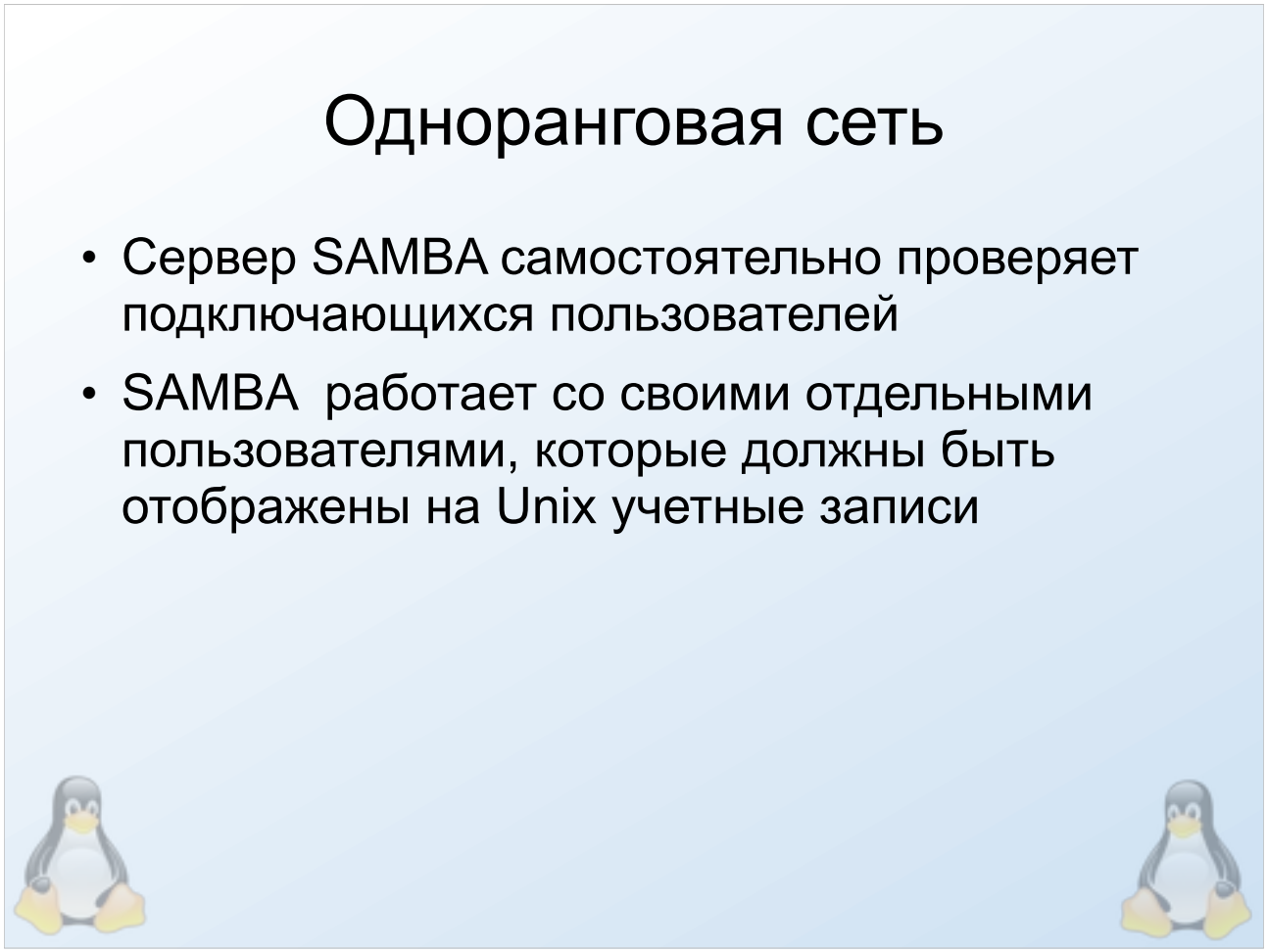
1. Настроив файл `smb.conf` следует проверить правильность его настроек с помощью программы `testparm`, позволяющей просмотреть полный список настроек.

Пример:

```
$ testparm -s | less
```

Примечание: Если среди настроек в файле `smb.conf` *имеется ошибка, то* `testparm` *выдаст сообщение об этом. Кроме того* `testparm` *выводит полный список опций конфигурации с их значениями. Опция* `-s` *требуется для того, чтобы перед выводом полного списка конфигурационных опций не надо было нажимать Enter.*

4.3. Настройка SAMBA сервера, как члена одноранговой сети.

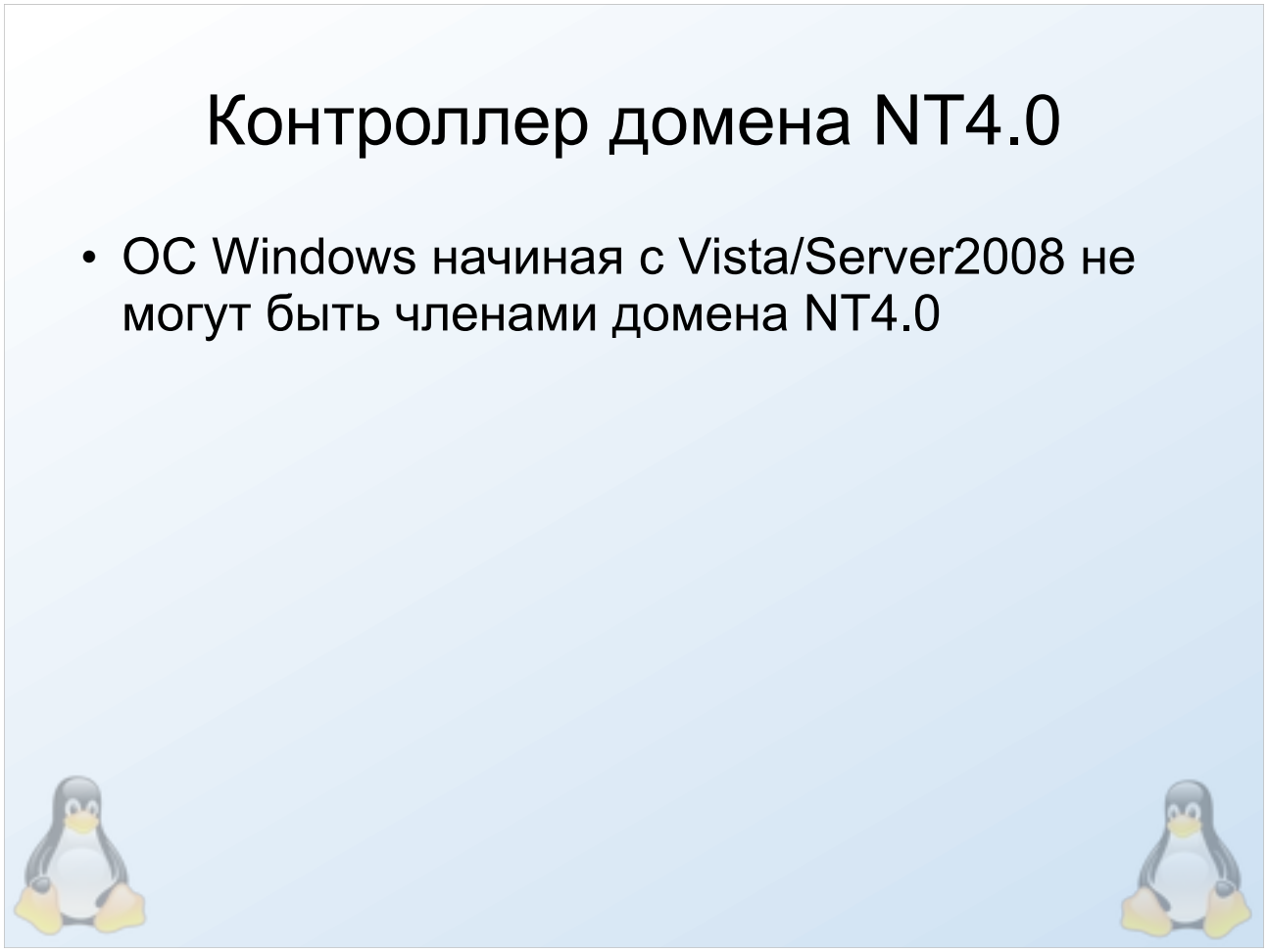


1. По умолчанию сервер SAMBA настроен в качестве члена одноранговой сети.
2. Для указания рабочей группы в глобальных параметрах следует указать параметр `workgroup=<имя_рабочей группы>`
3. Следует указать имя сервера и его описание.

Пример:

```
[global]
    workgroup = WORKGROUP
    netbios name = SAMBASRV1
    server string = Samba 2.2.6 on Linux
```

4.4. Настройка SAMBA сервера, как PDC (NT4)



1. Сервер SAMBA может выполнять следующие доменные роли:
   1. PDC – основной контроллер домена NT 4.0;
   2. BDC – вторичный контроллер домена NT 4.0;
   3. Member server – член домена NT/2000/2003;
2. При настройке SAMBA в качестве контроллеров домена следует помнить, что он не может обеспечить полную совместимость с контроллерами домена Windows NT 4.0, но может реализовать большинство его функций.
3. Для настройки PDC необходимо указать следующие параметры в глобальной секции:
   1. `netbios name` – имя компьютера **(обязательный)**;
   2. `workgroup` – имя домена **(обязательный)**;
   3. `passdb backend` – способ хранения информации о пользователях и группах. Может иметь следующие значения:
      1. `smbpasswd`,
      2. `tdbsam`,
      3. `nisplussam`,

4. `mysql`,
5. `ldapsam`.

Примечание: При использовании ldapsam можно создавать BDC под управлением SAMBA;

4. `os level` – вес сервера в выборах Master browser'a, должен быть больше 32.
5. `preferred master = yes` – вызвать перевыборы Master browser'a при включении демона nmbd;
6. `domain master = yes` – сделать доменным Master browser'ом **(обязательный)**;
7. `local master = yes` - сделать Master browser'ом для локальной сети;
8. `security = user` – задает способ аутентификации пользователей **(обязательный)**;
9. `domain logons = yes` – непосредственно включает режим PDC **(обязательный)**.

Пример:

```
[global]
netbios name = SAMBASRV1
workgroup = SAMBADOM
passdb backend = smbpasswd
os level = 33
preferred master = yes
domain master = yes
local master = yes
security = user
domain logons = yes
```

4. Помимо описания параметров сервера, необходимо создать общий ресурс `netlogon`. Эта папка необходима для выполнения сценариев входа пользователей в домен.

Пример:

```
[netlogon]
path = /var/lib/samba/netlogon
read only = yes
write list = ntadmin
```

5. Следующий этап в создании домена – регистрация компьютеров в качестве членов домена.
6. Для регистрации члена домена необходимо создать компьютерную учетную запись.

Пример: Регистрация компьютера с именем `sbmclient`

```
#/usr/sbin/useradd -d /dev/null -g 100 -s /bin/false -M smbclient$
#smbpasswd -a -m smbclient
```

Примечание: первая команда создает Unix учетную запись. Обратите внимание на знак $ в конце имени. Вторая команда регистрирует ее как клиента SAMBA.

7. Вместо ручной регистрации можно воспользоваться глобальной опцией `add machine script` для регистрации NT/2000/XP/2003 компьютеров.

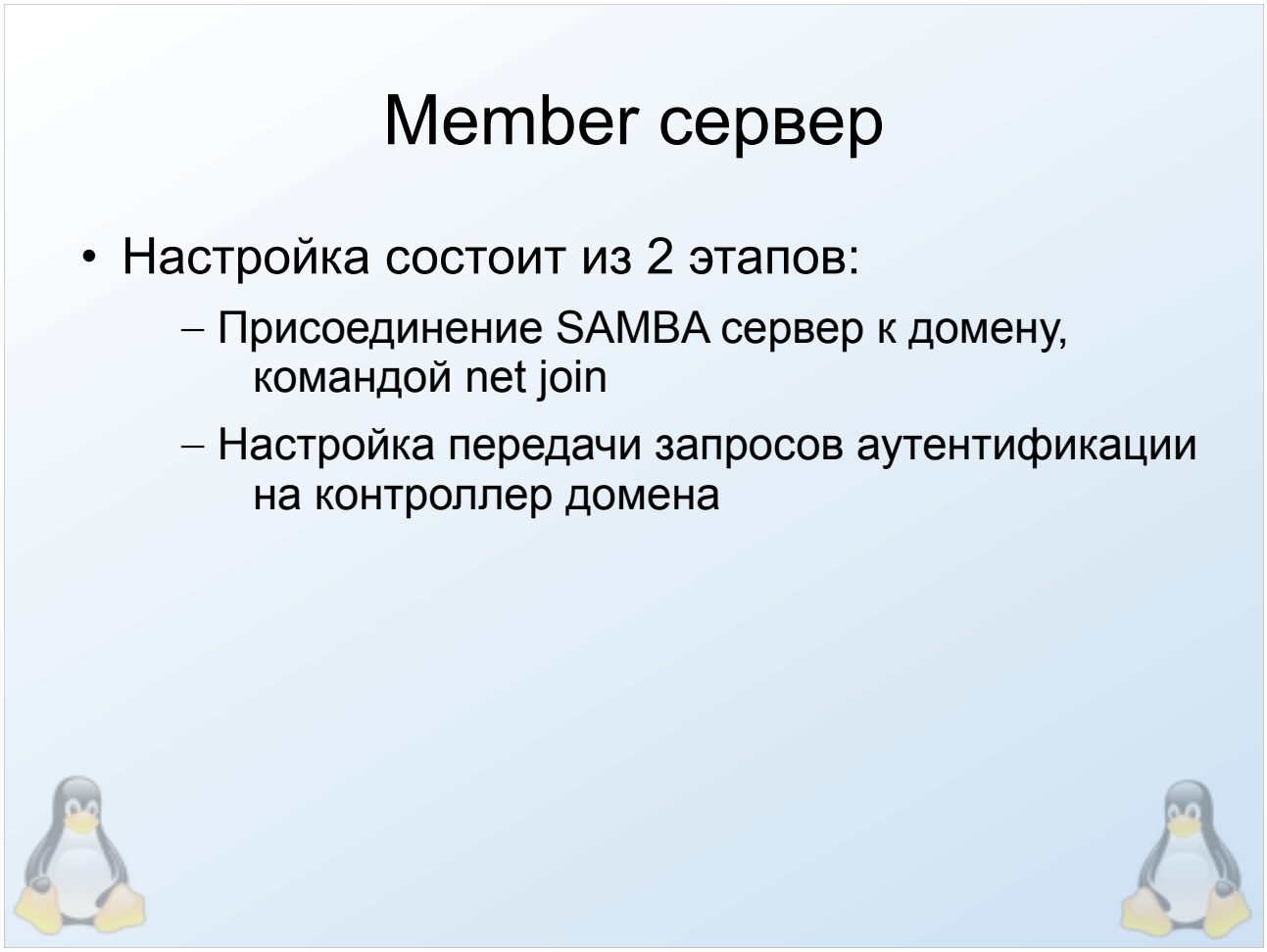
Пример:

```
add machine script = /usr/sbin/useradd -d /dev/null -g 100 -s /bin/false -M %u
```

Примечание: проверьте работает ли эта команда в вашей системе.

7. Необходимо также зарегистрировать пользователей Unix с помощью команды `smbpasswd`.

4.5. Настройка сервера SAMBA, как члена домена NT 4.0



1. Для настройки сервера SAMBA в качестве члена домена нужно проделать следующие:
   1. создать для сервера SAMBA компьютерную учетную запись на контроллере домена;
   2. настроить SAMBA передавать запросы аутентификации контроллерам домена.
2. Создать учетную запись на контроллере домена можно командой `net`.

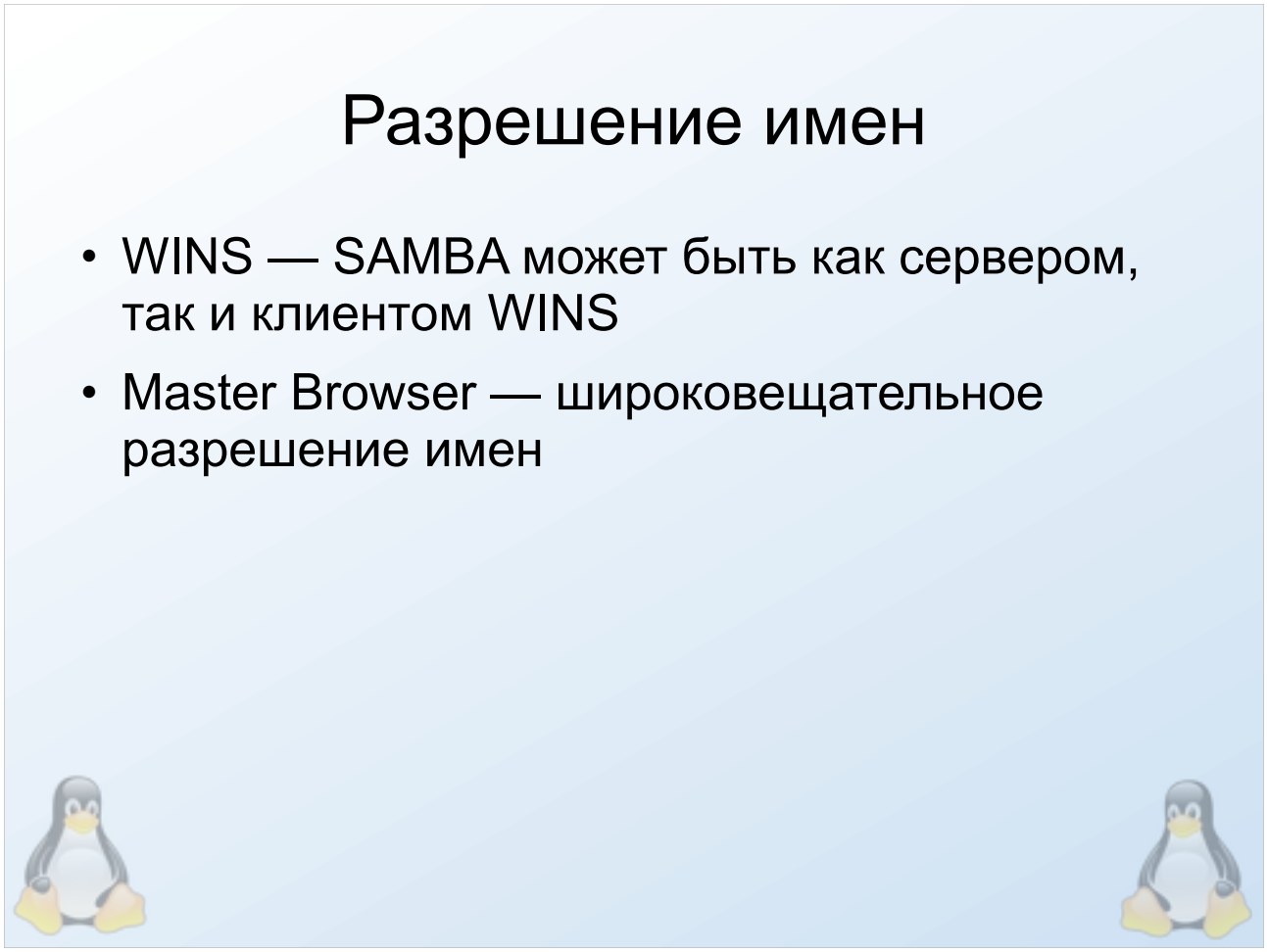
Пример: присоединение сервера к домену DOMAIN1 от имени пользователя root с паролем lin123. PDC этого домена называется SAMBAPDC.

```
# net join -S SAMBAPDC -Uroot%lin123
Joined domain DOMAIN1
```

3. Для настройки аутентификации на контроллере домена необходимо указать в глобальной секции `smb.conf` следующие параметры:
   1. `workgroup = <имя домена>` - указывает, в каком домене мы состоим;
   2. `security = domain` – директива передавать имя и пароль на проверку контроллерам домена;
   3. `password server = *` - имя контроллера домена.

*Примечание: параметр password server = * означает передать первому найденному контроллеру домена. Можно указать явно имена контроллеров домена, например: password server = DC1 DC2.*

4.6. Настройка SAMBA сервера, как WINS сервера



1. SAMBA сервер может выступать в роли:
   1. WINS (Windows Internet Name Server) сервера – регистрирует, разрешает и освобождает имена в IP адрес;
   2. WINS клиента;
   3. WINS proxy – перехватывает широковещательные запросы от не WINS клиентов и передает их WINS серверу.
2. Для настройки WINS сервера в глобальных параметрах `smb.conf` следует указать `wins support = yes`.
3. Если вы включили на SAMBA WINS сервер, то проверьте, чтобы других WINS серверов в сети **не было**.
4. В настоящее время SAMBA WINS не может реплицировать свою базу данных на другие WINS серверы.
5. Когда SAMBA работает как WINS сервер, он может вызывать внешнюю программу при изменениях в базе данных WINS. Параметр `wins hook`. Обычно применяется для динамической регистрации в DNS.
6. Настройка WINS клиента осуществляется настройкой `wins server = {IP адрес}`.

7. Можно указывать несколько WINS серверов, но клиент работает только с одним из них.
8. Клиент опрашивает WINS сервера по очереди, пока один из них не ответит. После ответа одного из серверов к другим обращений не производится, независимо от результата ответа.
9. Адрес каждого сервера можно предварять тегом.

Пример:

```
wins server = main:192.168.1.100 secondary:192.168.2.200
```

10. WINS proxy позволяет передавать широковещательные запросы не WINS клиентов WINS серверу. Параметр `wins proxy = {yes|no}`.

4.7. Настройка службы браузера

1. Для регистрации, разрешения и освобождения NetBIOS имен в сетях с клиентами, работающими в режиме B-node, используется служба браузера.
2. В локальном сегменте сети выбирается компьютер (Master Browser), который производит регистрацию, разрешение и освобождение имен, а также обновление списков зарегистрированных компьютеров на остальных узлах в сети.
3. Выборы master browser'a происходят, когда в сети появляется компьютер с большим приоритетом, чем у существующего master browser'a.
4. В глобальной секции smb.conf имеются несколько параметров влияющих на поведение службы браузера в SAMBA.
    1. `local master` – позволить или запретить демону nmbd становиться мастер браузером в локальном сегменте.
    2. `domain master` – сервер с этой опцией копирует списки с локальных мастер браузеров, формирует из них общий список, а, затем, распространяет общий список узлов обратно локальным мастер браузерам.
    3. `preferred master` – инициирует перевыборы при включении службы nmbd.
    4. `os level` – вес компьютера в выборах мастер браузера.

4.8. Обслуживание SAMBA нескольких IP сетей. Трансляция NetBIOS имен.

1. При работе в нескольких IP сетях, необходимо соблюсти три правила:
    1. В каждой подсети должен быть один Local Master Browser.
    2. В объединенной сети должен быть один Domain Master Browser.
    3. В объединенной сети должен быть WINS сервер.
2. Local Master Browser создается глобальным параметром `local master = yes`.
3. Если в сети нет Domain Master Browser и/или WINS сервера и вы не хотите их настраивать, то можно поступить следующим образом:
    1. Передавать широковещательные уведомления в удаленные подсети. Глобальный параметр `remote announce`

Пример: передача списка узлов рабочей группы WORKGROUP в подсети 192.168.2.0 и 192.168.3.0.

```
remote announce = 192.168.2.255/WORKGROUP 192.168.3.255/WORKGROUP
```

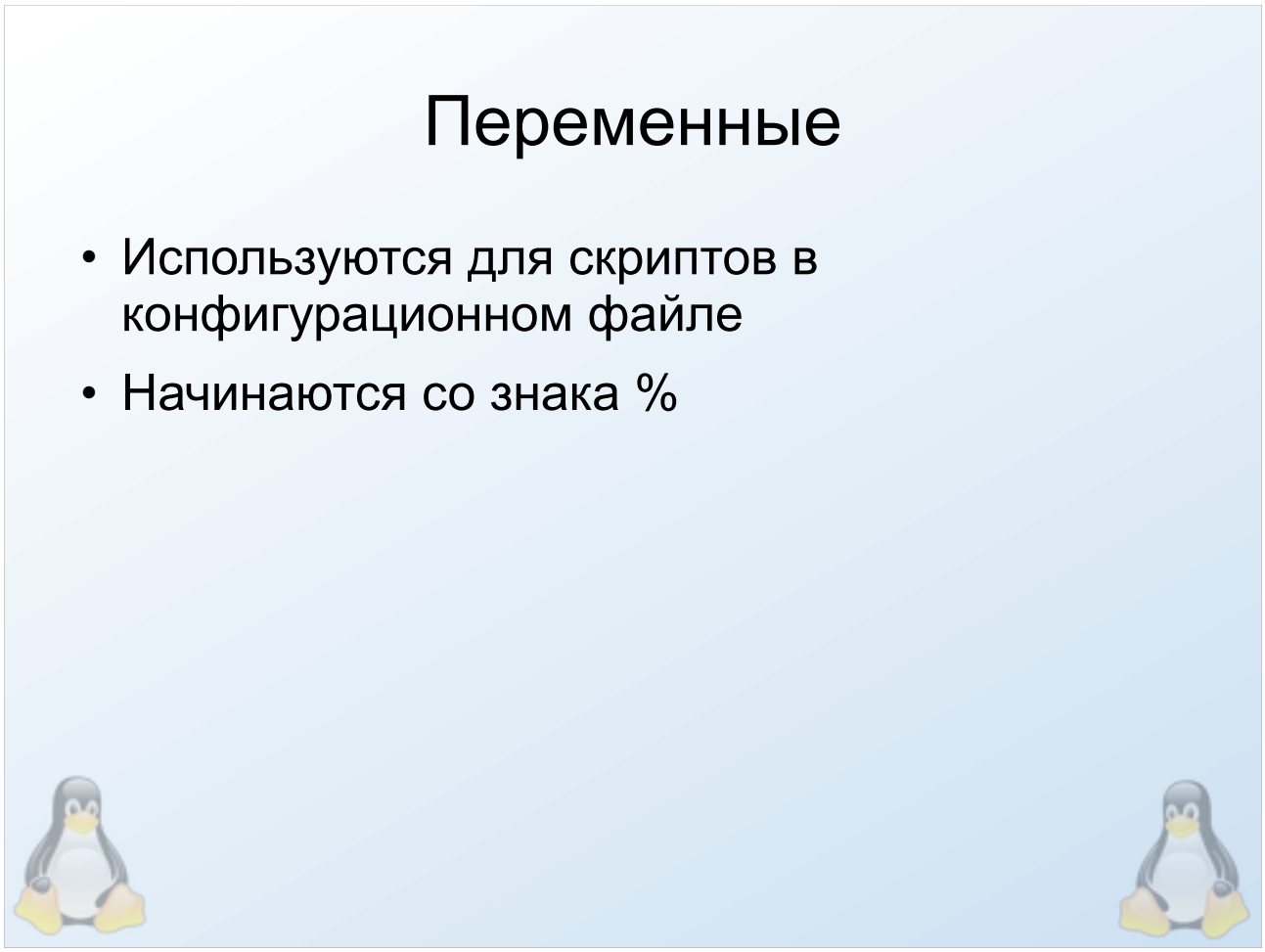
Примечание: не всегда возможно передавать широковещательные пакеты между подсетями.

    2. Второй способ позволяет напрямую уведомлять Local Master Browser о изменениях в списках. Параметр `remote browse sync`

Пример: передача списка узлов в подсети 192.168.2.0 и 192.168.3.0 непосредственно Local Master Browser.

```
remote browse sync = 192.168.2.130 192.168.3.120
```

4.9. Переменные SAMBA

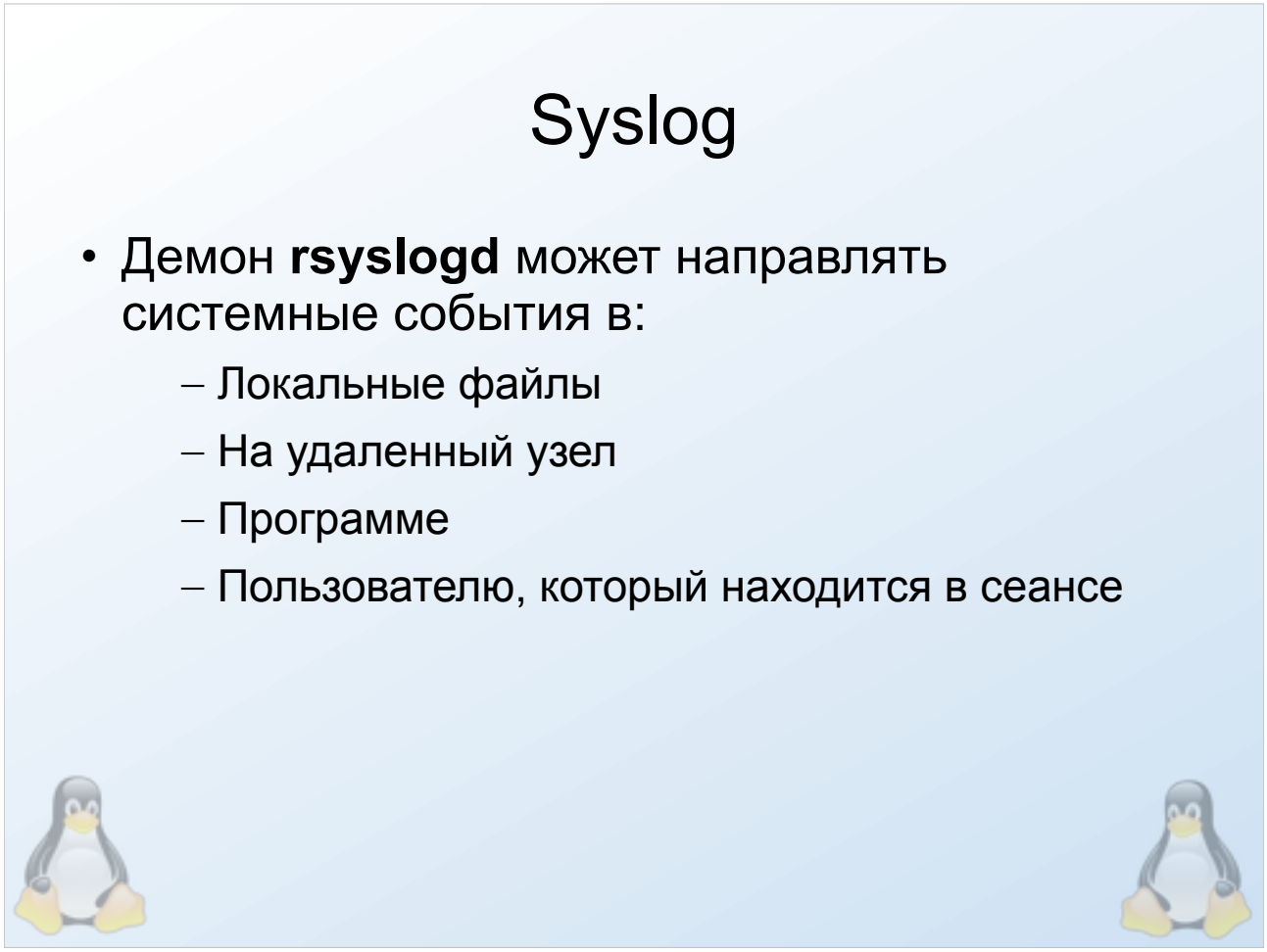


1. В конфигурационном файле SAMBA используются переменные для выполнения сценариев.
2. Переменные начинаются со знака %.
3. Наиболее важные переменные:
    1. `%U` – имя пользователя, запросившего службу
    2. `%u` – имя пользователя установившего сессию, работает только при установленном соединении, поэтому работает не во всех опциях.
    3. `%I` – IP адрес клиента.
    4. `%m` – NetBIOS имя клиента.
    5. `%L` – NetBIOS имя сервера.
    6. `%h` – имя узла сервера SAMBA.
    7. `%M` – имя узла клиента.
    8. `%G` – основная группа пользователя `%U`.
    9. `%g` – основная группа пользователя `%u`

10. `%D` – имя домена или рабочей группы.
11. `%T` – текущая дата и время.

Глава 5. Поддержка системных журналов.

5.1. Настройка rsyslogd для локального журналирования и в качестве клиента.



1. В GNU/Linux принято сохранять информацию разной степени детализации о процессе работы программ в специальных текстовых файлах, называемых журналами.
2. Стандартное место расположения журналов - это каталог `/var/log` .
3. Все журнальные файлы принадлежат к одной из двух категорий: системные журналы и журналы прикладных программ.
4. Служба rsyslog предназначена для обеспечения сохранения информации, поступающей от различных системных служб.
5. Служба rsyslog представлена демоном `rsyslogd` , запускаемым при старте операционной системы автоматически.

Пример:

```
$ ps -C syslogd
  PID TTY          TIME CMD
```

```
 1246 ?        00:00:00 syslogd
```

6. Задачей демона `syslogd` является сбор сообщений от системных служб и сохранение этих сообщений в заранее известных файлах журналов.
7. Конфигурационным файлом демона `syslogd` является `/etc/syslog.conf` .
8. Строки этого файла, начинающиеся с решетки # являются комментариями.
9. Структура строк, каждая из которых направляет некоторый поток сообщений в заданный файл (или на удаленный компьютер - сервер ведения журналов), представлена двумя полями:
   1. Определение сообщения (selector) - поле, в котором указывается от каких классов программ должны собираться сообщения в данный поток. И какие именно сообщения.
   2. Поле действия (action), указывающее куда должен быть записан поток сообщений. Чаще всего - это имя файла журнала в `/var/log` .

Пример:

```
#SELECTOR                       ACTION

*.crit;lpr,cron,mail.none       /var/log/critical
daemon.info;daemon.!err         -/var/log/daemons
daemon.=err                     /var/log/daemons.err
authpriv.*                      /var/log/messages
kern.*;kern.!=info              /var/log/kernel
cron.info                       -/var/log/cron
lpr.info                        -/var/log/lpr
mail.warning                    -/var/log/mail/mail
```

Примечание: Из приведенного выше примера заметно, что поле определения сообщения состоит из двух частей, разделенных точкой. Первая часть - источник сообщения (facility), а вторая - уровень важности (priority) сообщения. Эти две части уникально определяют все возможные сообщения, обрабатываемые syslog .

10. Служба rsyslog работает со следующими источниками сообщений (`facility`):
    1. `auth` – сообщения служб авторизации и безопасности (этот источник не рекомендуется использовать, вместо него необходимо использовать `authpriv`);
    2. `authpriv` – сообщения служб авторизации и безопасности;
    3. `cron` – сообщения служб `at` и `cron` ;
    4. `daemon` – сообщения различных демонов;
    5. `kern` – сообщения ядра;
    6. `lpr` – сообщения службы печати;
    7. `mail` – сообщения, поступающие от служб электронной почты;
    8. `mark` – источник, зарезервированный для внутреннего использования (в syslog), его не следует использовать в приложениях;

9. `news` – сообщения службы новостей;
10. `security` – то же, что и `auth` ;
11. `syslog` – собственные сообщения syslog;
12. `user` – источник сообщений, зарезервированный для пользовательских программ;
13. `uucp` – сообщения службы uucp;
14. `local0 ... local7` – источники сообщений, доступные для использования на локальной системе, но не являющиеся стандартными.

11. Все сообщения разделены также по следующим уровням важности (`priority`), приведенным в порядке убывания важности:
    1. `emerg` – система не работоспособна;
    2. `panic` – то же, что и `emerg` (не рекомендуется использовать);
    3. `alert` – требуется немедленное вмешательство;
    4. `crit` – критическое событие;
    5. `err` – ошибка;
    6. `error` – то же, что и `err` (не рекомендуется использовать);
    7. `warning` – предупреждение;
    8. `warn` - то же, что и `warning` (не рекомендуется использовать);
    9. `notice` – нормальное, но значимое событие;
    10. `info` – информационное сообщение;
    11. `debug` – отладочная информация.
12. Знак звездочка (*) является метасимволом, обозначающим либо все источники, если он указан перед точкой – разделителем, либо все уровни важности, если этот метасимвол установлен после точки.
13. При указании источника сообщения и уровня важности, разделенных точкой, определяется, что сообщения, поступающие от этого источника и имеющие указанный и вышележащие уровни важности, будут записаны в данный канал.

Пример:

```
lpr.info                        -/var/log/lpr
```

Примечание: Здесь установлено, что все сообщения службы печати с уровнями важности `info` *и выше, будут записаны в файл* `/var/log/lpr` *. Знак тире перед именем файла обозначает, что синхронизация буфера с диском при записи в этот файл журнала не обязательна.*

14. При необходимости запретить запись в какой-либо канал сообщений с заданным уровнем важности и выше, можно использовать знак восклицания (!) перед уровнем важности.

Пример:

```
daemon.info;daemon.!err         -/var/log/daemons
```

Примечание: В этом случае все сообщения от демонов с уровнями важности от `info` *до*

`warning` *будут записываться в журнал* `/var/log/daemons` *, а сообщения с уровнями важности, начиная с* `err` *, записаны туда не будут.*

15. Если же необходимо записывать в журнал сообщения только с определенным уровнем важности и ни с какими другими более, то перед требуемым уровнем важности следует поставить знак равно (=).

Пример:

```
daemon.=err                    /var/log/daemons.err
```

Примечание: В файл `/var/log/daemons.err` *будут записываться только сообщения об ошибках в работе демонов.*

16. Для исключения из потока сообщений те из них, которые имеют заданный уровень важности используют восклицательный знак и знак равенства (!=).

Пример:

```
kern.*;kern.!=info             /var/log/kernel
```

Примечание: Здесь в журнал будут записываться все сообщения от ядра, кроме информационных.

17. Если в канал не должны быть записаны любые сообщения от каких-либо источников, то удобно использовать директиву `none` :

Пример:

```
*.crit;lpr,cron,mail.none      /var/log/critical
```

Примечание: При этом в файл `/var/log/critical` *будут записываться сообщения от всех источников о критических и более важных событиях, кроме любых сообщений от служб печати, почты,* `at` *и* `cron` *.*

18. При записи события в файл журнала syslog производит синхронизацию буферов с диском.

19. Такая процедура позволяет получить некоторую уверенность, что все события будут записанны в журнал, но это также может создать проблемы связанные с производительностью компьютера.

20. Чтобы избежать таких проблем, для некритичных событий можно отключить использование системного вызова sync после регистрации каждого события. Для этого перед именем файла, в который записываются события, необходимо поставить знак минуса (-).

Пример:

```
cron.info                      -/var/log/cron
lpr.info                       -/var/log/lpr
mail.warning                   -/var/log/mail/mail
```

Примечание: Для критичных событий такую возможность лучше не использовать.

21. Для обеспечения повышенной защищенности сети все сообщения можно хранить не на локальном компьютере, а передавать по сети на специально выделенный для этого сервер.

22. Для передачи сообщений по умолчанию используется протокол UDP и порт 514.

23. Чтобы указать сервер, на который будут посылаться сообщения, в качестве цели необходимо указать имя компьютера, которому посылаются сообщения, и перед именем поставить знак @.

Пример:

```
*.*  @logserver
```

Примечание: в данном примере все сообщения посылаются на сервер с именем logserver.

24. Имя сервера протоколирования желательно указывать в файле `/etc/hosts`, поскольку демон `syslogd` может стартовать раньше, чем начинает работать разрешение имен.

25. Вместо имени узла, регистрирующего события, можно указать IP-адрес.

26. `syslogd` позволяет передавать сообщения именованным каналам: `|файл_FIFO`.

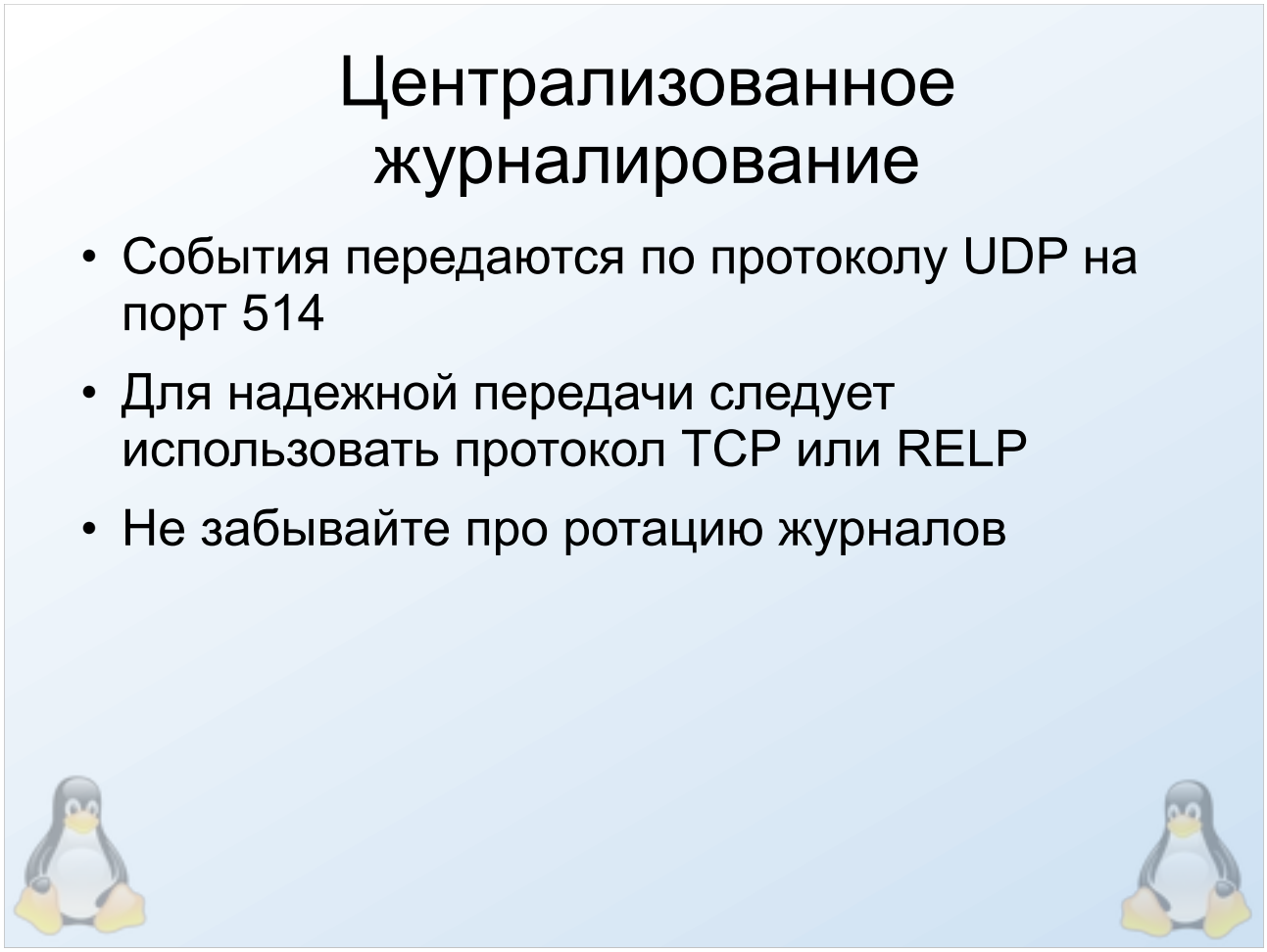
27. Передача сообщений зарегистрированным пользователям осуществляется указанием имени пользователя в качестве цели.

Пример:

```
*.emerg   root,sysadm,user
```

28. Цель * означает записать данное событие повсюду, т.е. в каждый журнал, именованный канал, всем пользователям и хостам указанным в файле конфигурации.

5.2. Запуск rsyslog в качестве централизованного сервера журналирования



1. Для получения сообщений из сети в конфигурации rsyslog нужно сделать следующее:
    1. Включить input модуль и настроить его параметры (обязательно).
    2. Настроить отдельные правила для сообщений поступающих из сети (желательно). Если таких правил не будет, то сообщений поступающие из сети будут попадать в те же журналы, что и локальные события. Это, в свою очередь затруднит интерпретацию событий.

Пример: настройка приема сообщений из сети с простой сортировкой поступающих сообщений.

```
# cat /etc/rsyslog.d/inudp.conf
module(load="imudp") # needs to be done just once
input(type="imudp" port="514" ruleset="UDP514_IN")

template(name="RemHost" type="list") {
   constant(value="/var/log/network/")
   property(name="hostname")
   constant(value="_")
   property(name="fromhost-ip")
```

```
   constant(value="/")
   property(name="syslogfacility-text")
   constant(value=".log")
} #template end

# ls -R /var/log/network/
/var/log/network/:
debian11_172.27.2.144

/var/log/network/debian11_172.27.2.144:
daemon.log  syslog.log  user.log

ruleset(name="UDP514_IN"){
   action(type="omfile" dynaFile="RemHost")
} #ruleset end
```

2. Для проверки настроек сервера журналирования можно применять утилиту logger:

```
logger [options] [message. . .]
```

3. Сообщение (message) может быть передано в командной строке, иначе сообщением является информация, поступающая со стандартного ввода.
4. Опции `logger`:
   1. `-f file` - Прочитать содержимое сообщения из указанного файла.
   2. -i - Включить идентификатор процесса команды logger**.**
   3. `-p pri` - Указать приоритет сообщения (pri). По умолчанию это user.notice.
   4. `-t tag` - Пометить все строки сообщения в журнале специальным тегом (tag).
5. Помимо настройки непосредственно журналирования, важно так же настроить ротацию журналов.
6. Проверьте настройки ротации командой `logrotate` с опцией `-d`.

Пример: настройки ротации журналов

```
# cat /etc/logrotate.d/net_logs
/var/log/network/*/*.log{
      weekly
      rotate 8
      missingok
      create 0600 root root
      sharedscripts
      postrotate
       # postrotate script should always return 0
       /usr/bin/systemctl kill -s HUP rsyslog.service >/dev/null 2>&1 || true
```

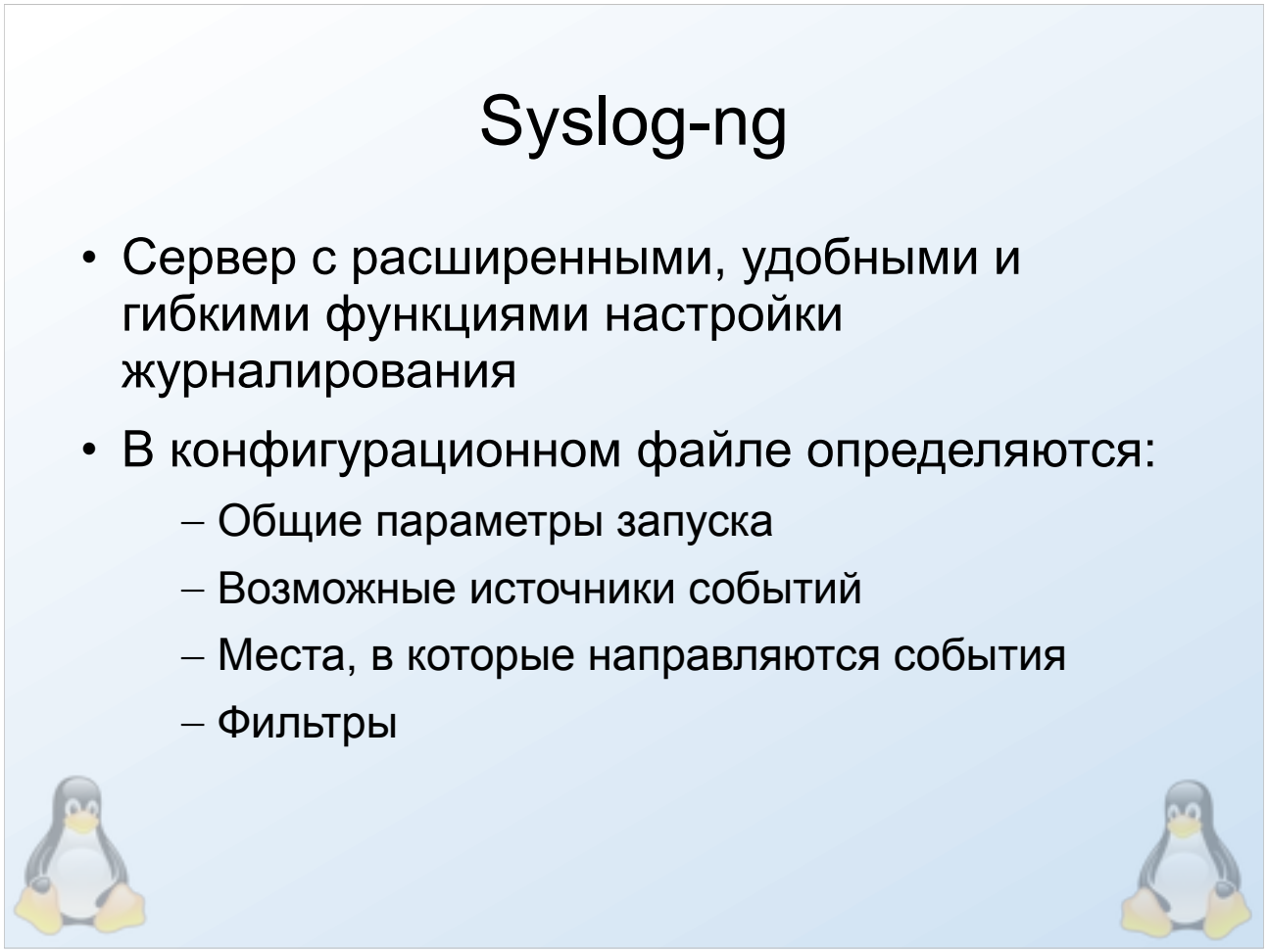
```
      endscript
}
# logrotate -d /etc/logrotate.d/net_logs
WARNING: logrotate in debug mode does nothing except printing debug messages!
Consider using verbose mode (-v) instead if this is not what you want.

reading config file /etc/logrotate.d/net_logs
Reading state from file: /var/lib/logrotate/logrotate.status
Allocating hash table for state file, size 64 entries
Creating new state
<...>

Handling 1 logs

rotating pattern: /var/log/network/*/*.log weekly (8 rotations)
empty log files are rotated, old logs are removed
considering log /var/log/network/debian11_172.27.2.144/authpriv.log
Creating new state
  Now: 2022-05-21 18:16
  Last rotated at 2022-05-21 18:00
  log does not need rotating (log has been already rotated)
considering log /var/log/network/debian11_172.27.2.144/cron.log
<...>
```

5.3. Пакет syslog-ng



1. Пакет syslog-ng (syslog new generation) это прямой потомок проекта syslog.
2. Пакет обладает расширенными функциями, обеспечивающими лучшую гибкость в управлении средствами журналирования.
3. Среди основных улучшений можно перечислить следущие:
   1. фильтрация сообщений;
   2. передача получателям;
   3. обеспечение безопасности передачи, посредством проверки целостности и шифрования сообщений;
   4. возможность работать по протоколам TCP и UDP.
4. Пакет изначально разрабатывался под руководством Базши Шейдлера (Bazsi Scheidler). *http://www.balabit.hu*.
5. Основные опции, с которыми работает `syslog-ng`:
   1. `-f файл` – путь к конфигурационному файлу;
   2. `-p файл` – путь к PID файлу.
6. Основной конфигурационный файл пакета `syslog-ng.conf`.

7. В файле конфигурации определяются следующие параметры:
   1. `options{}` - устанавливает глобальные опции `syslog-ng`. Основные опции:
      1. `chain_hostnames( yes | no )` - после записи имени узла, записывает все узлы через которые доставлялось сообщние.(`default=yes`).
      2. `keep_hostname( yes | no )` - доверять имени указанному в tcp/udp сообщении(`default=no`).
      3. `use_fqdn( yes | no )` - записывать полное имя tcp/udp отправителя (`default=no`).
      4. `use_dns( yes | no )` - разрешать IP адрес tcp/udp отправителя (`default=yes`).
      5. `use_time_recvd( yes | no )` - устанавливать временной штамп равный времени получения, а не отправления (`default=no`).
      6. `time_reopen( NUMBER )` - число секунд после разрыва tcp соединения и до его нового установления (`default=60`).
      7. `time_reap( NUMBER )` - число секунд перед закрытием неактивного файла(`default=60`).
      8. `log_fifo_size( NUMBER )` - максимальное число сообщений в очереди syslog-ng, если сообщений больше, то лишние отбрасываются (`default=100`).
      9. `sync( NUMBER )` - после записи указанного числа сообщений производится синхронизация диска. (`default=0`).
      10. `owner( string )` - владелец файлов. (`default=root`).
      11. `group( string )` - группа владельцев. (`default=root`).
      12. `perm( NUMBER )` - права доступа к журналам. (`default=0600`).
      13. `create_dirs( yes | no )` создавать/не создавать отсутствующие каталоги. (`default=no`).
      14. `dir_owner( string )` - владелец создаваемых каталогов. (`default=root`).
      15. `dir_group( string )` - группа владельцев каталогов. (`default=root`).
      16. `dir_perm( NUMBER )` - разрешения на доступ к каталогам. (`default=0700`).
   2. `source{}` - источники событий:
      1. `internal()` - сообщения от демона *syslog-ng*.
      2. `file("filename" [options])` – сообщения из файлов, например: */proc/kmsg*.
      3. `pipe("filename" )` - сообщения из именованных каналов.
      4. `unix_stream("filename" [options])` – сообщения от Unix сокетов (*/dev/log*) ориентированных на соединение (работает в ядрах до 2.4). В опциях может быть указано максимальное количество конкурируемых соединений. (default=100).
      5. `unix_dgram("filename" [options])` - сообщения от Unix сокетов (*/dev/log*) не ориентированных на соединение (работает в ядрах после 2.4).
      6. `tcp([ip(address)] [port(#)][max-connections(#)] )` - сообщения полученные по протоколу tcp на указанный порт (`default=514`) и указанный сетевой интерфейс (`default=all`); последний параметр количество одновременных соединений (`default=10`).

7. udp([ip(address)] [port(#)]) - сообщения полученные по протоколу udp на указанный порт (default=514) и указанный сетевой интерфейс (default=all);

3. destination{} - место регистрации событий:
   1. file("filename[$MACROS]") - стандартный ASCII текстовый лог файл. $MACROS может быть использован для автоматического именования файлов.
   2. tcp("address" [port(#);]) - передать сообщения на указанный адрес и порт TCP (default=514)
   3. udp("address" [port(#);]) - передать сообщения на указанный адрес и порт UDP (default=514).
   4. pipe("pipename") – передача в именованный канал.
   5. unix_stream("filename" [options]) – передача с установкой соединения на Unix сокет.
   6. unix_dgram("filename" [options]) – передача без установки соединения на Unix сокет.
   7. usertty(username) - направить на консоль указанного пользователя.
   8. program("/path/to/program") – передать на стандартный вход (stdin) программы.
4. filter{} - фильтры:
   1. facility(facility-name) - источник события;
   2. priority(priority-name), или priority(priority-name1,priority-name2, etc.), или priority(priority-name1..priority-name2), или level(priority-name) - фильтрация по приоритету, или списку приоритетов, или диапазону приоритетов.
   3. program(program-name) - по программе сгенерировавшей событие.
   4. host(hostname) - по компьютеру от которого получено сообщение.
   5. match(regular-expression) - по регулярному выражению в теле сообщения.
   6. filter(filter-name) - другой фильтр.
5. log{} - “собирает” параметры source, destination и filter, чтобы сформировать журнал, в котором регистрируются события.

Пример: конфигурационного файла syslog-ng.

```
# cat /etc/syslog-ng/conf.d/inudp.conf
source s_udp10514 { udp(ip(0.0.0.0) port(10514)); };

destination d_syslog-ng { file("/var/log/syslog-ng/${HOST}_${HOST_FROM
}/${FACILITY}.log"); };
```

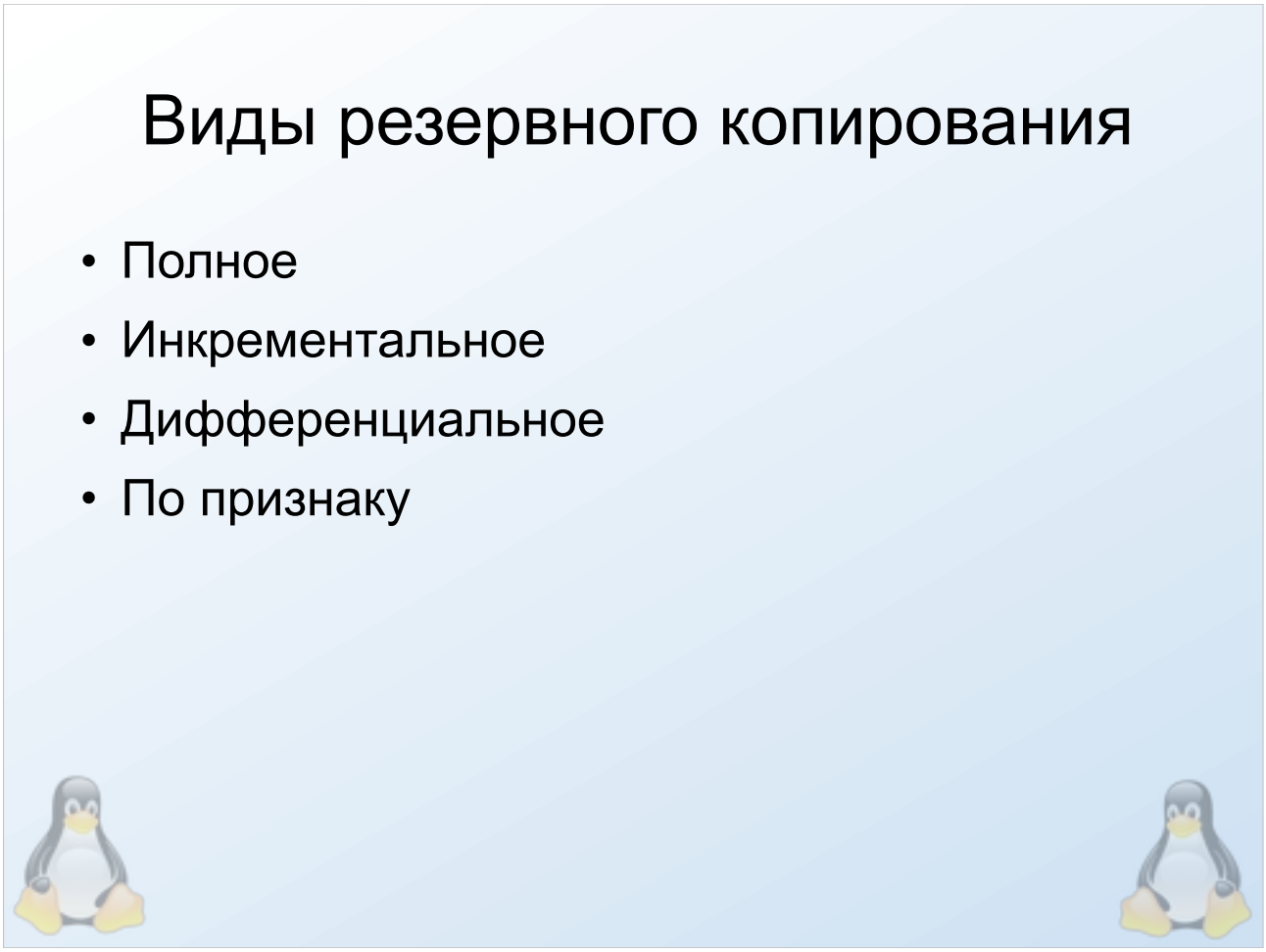
```
log { source(s_udp10514); destination(d_syslog-ng); };

# ls -R /var/log/syslog-ng/
/var/log/syslog-ng/:
debian11_172.27.2.144

/var/log/syslog-ng/debian11_172.27.2.144:
user.log
```

Глава 6. Резервное копирование.

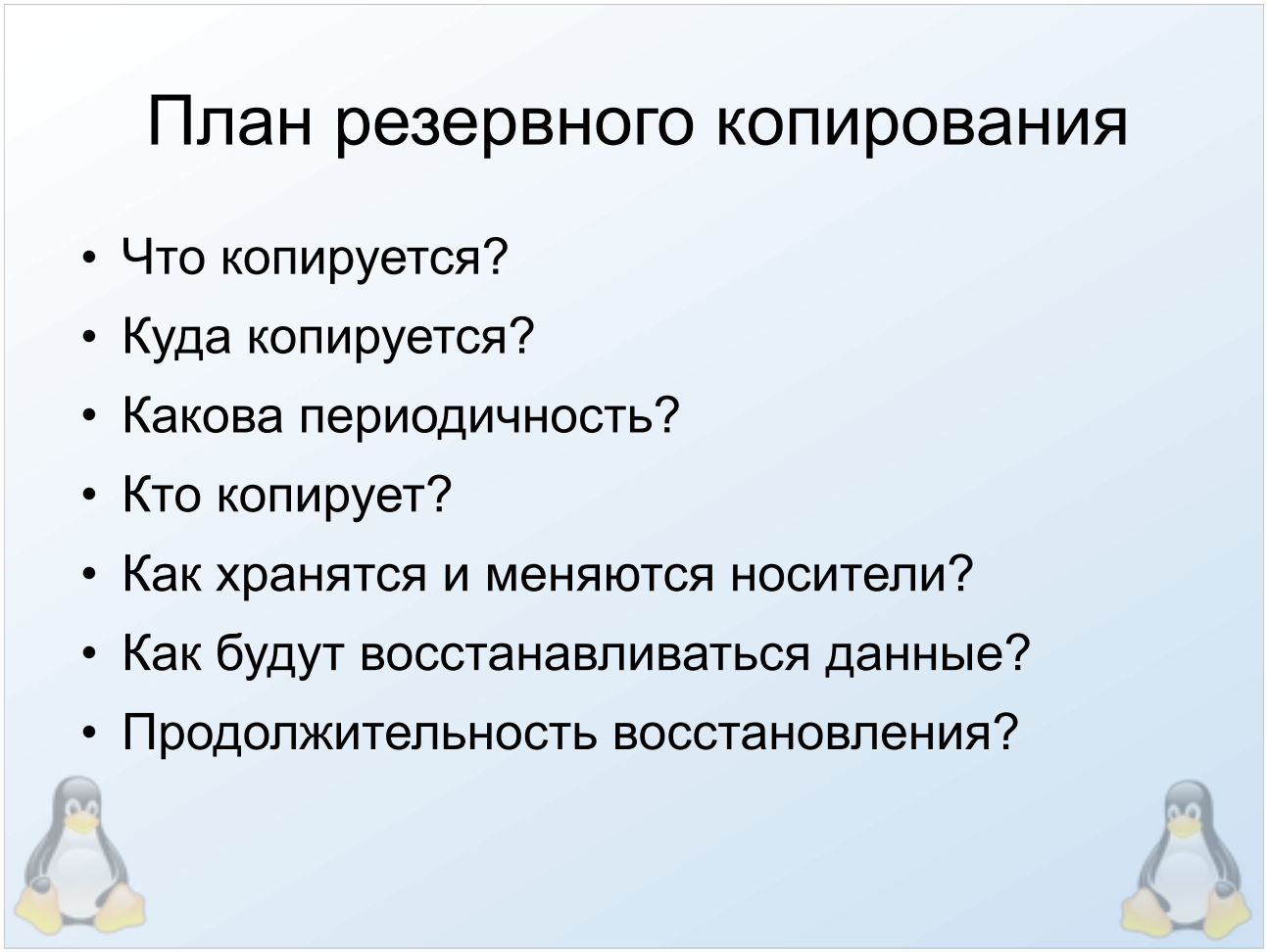
6.1. Виды резервного копирования.



1. Резервное копирование - это единственная операция, которая может гарантировать сохранность данных, находящихся в вычислительной системе.
2. Процесс архивирования данных можно разделить на следующие виды:
    1. Полное (обычное, full, normal)
    2. Инкрементальное (добавочное, incremental)
    3. Разностное (differential)
    4. Архивирование по признаку (data-specific).
3. При полном копировании архивируется все, что вы указали программе архивации. При этом программа архивации не анализирует данные, а просто помещает их в архивную копию.
4. Для инкрементального и разностного архивирования необходимо предварительно создать полную архивную копию данных.

5. В инкрементальном копировании в архивную копию попадают данные, которые изменились с момента последней архивации.
6. В разностном копировании в архивную копию попадают данные, изменившиеся с момента полной архивации.
7. Резервное копирование данных должно производиться тогда, когда эти данные не изменяются. В противном случае нельзя гарантировать сохранности этих данных.
8. Желательно регулярно выполнять резервное копирование в часы минимальной нагрузки на систему.
9. Всегда проверяйте ваши архивные копии, на предмет возможности восстановления данных.
10. Проводите пробные восстановления.
11. Не храните архивные копии в том же месте, где находится система, которую вы архивируете.

6.2. Разработка плана инкрементального архивирования.



1. При планировании резервного копирования следует определить следующие моменты:
    1. какие данные должны быть скопированы и где они находятся (в каких каталогах или файловых системах, а также, возможно, на каких компьютерах);
    2. какой тип носителей данных будет использован для резервного копирования;
    3. с какой периодичностью будет производиться копирование данных;
    4. каким образом и с какой частотой будет осуществляться ротация резервных носителей данных;
    5. кто будет уполномочен производить резервное копирование;
    6. будет ли требоваться человеческий контроль за ходом копирования или же копирование будет производиться полностью автоматически;
    7. в какие моменты времени должно осуществляться резервное копирование;
    8. где и как будет производиться хранение носителей резервных копий;
    9. каков будет порядок восстановления утраченных данных из резервных копий;
    10. максимальная допустимая продолжительность периода времени, необходимого для восстановления данных.

2. При разработке плана резервного копирования важно оценить два важнейших параметра:
    1. **RPO** (recovery point objective) – допустимая потеря данных. Любая информационная система должна обеспечивать (внутренними ли средствами, или сторонними) защиту своих данных от потери выше приемлемого уровня.
    2. **RTO** (recovery time objective) – допустимое время восстановления данных Любая информационная система должна обеспечивать (внутренними ли средствами, или сторонними) возможность восстановления своей работы в приемлемый срок.
3. Суммарное время на восстановление данных состоит как минимум из:
    1. времени, необходимого для поиска и доставки носителя резервной копии требуемых данных;
    2. времени на поиск требуемой информации на носителе (если поиск допускается);
    3. времени копирования информации с резервного носителя.
4. Разработка плана архивного копирования копирования и неукоснительное соблюдение этого плана помогает (но не гарантирует) сократить время простоев при аварийных сбоях, связанных с поиском и восстановлением данных.
5. При планировании инкрементального архивирования обычно составляют расписание, в соответствии, с которым, проводят архивацию.
6. В расписании необходимо указать:
    1. периодичность полного архивирования;
    2. периодичность и уровень инкрементального архивирования.

Пример: плана резервного копирования. Предположим имеются данные, с которыми ежедневно работают пользователи. Эти данные важны для нормального функционирования компании. Предполагается, что небольшая часть пользователей работает в выходные. При копировании требуется обеспечить целостность данных. Здесь возможны следующие варианты архивирования.

План 1.

Воскресенье	***Понедельник***	***Вторник***	***Среда***	***Четверг***	***Пятница***
Полный	Инкремент.	Инкремент.	Инкремент.	Инкремент.	Инкремент.

В этом плане в воскресенье, когда пользователи не работают в системе производится полное архивирование. В буднии дни инкрементальное копирование. Такой план позволяет снизить затраты времени и ресурсов на проведение архивирования, но требует более внимательного восстановление данных. Например, для восстановления всех данных за среду требуется сначала восстановить данные воскресного архивирования, затем последовательно восстанавливать данные из копий понедельника, вторника и среды.

План 2.

Воскресенье	***Понедельник***	***Вторник***	***Среда***	***Четверг***	***Пятница***
Полный	Разностный	Разностный	Разностный	Разностный	Разностный

В этом плане в воскресенье, когда пользователи не работают в системе производится полное архивирование. В будние дни разностное копирование. Такой план позволяет снизить

затраты времени на проведение восстановления, но требует более длительного копирования данных. В этом плане требуется большее количество пространства на резервных носителях, чем в первом плане. Например, для восстановления всех данных за среду требуется сначала восстановить данные воскресного архивирования, затем данные из копии среды.

6.3. Полное и инкрементальное архивирование с помощью tar.

1. Один из наиболее часто используемых инструментов резервного копирования - команда `tar` (tape archive).

1. С помощью этой команды можно создавать архивы файлов и каталогов, заданных ей в качестве аргументов.
2. Опция `f` команды `tar` указывает файл, в который будет помещен архив.
3. Опция `c` команды предназначена для создания полного архива (create).
4. Опция `v` заставляет команду `tar` выводить информацию об обрабатываемых файлах.

Пример: приведенная ниже команда поместит архив, содержащий всю информацию в каталоге `/home` , на магнитную ленту.

```
# tar cvf /dev/st0 /home
```

Примечание: После выполнения этой команды на магнитной ленте будет создан полный архив файлов в каталоге (со всеми подкаталогами) `/home` *. Для работы с магнитными лентами предназначены символьные файлы устройств, поэтому информация на них записывается в виде потока байт.*

5. Помимо создания файла архива в `tar` имеются опции по добавлению файлов в архив или

слиянию архивов:

1. `-A` - добавление файлов `tar` архива в существующий архив (слияние);
2. `-r` - добавление файлов в конец архива;
3. `-u` - обновление архива версиями файлов, более новыми, чем в архиве;

6. Инкрементальное копирование можно организовать с помощью опции `--listed-incremental (-g).`
7. При первом вызове с опцией `-g tar` создает файл с информацией о времени архивирования.
8. При последующих вызовах с этой опцией обрабатываются только те файлы, которые изменились с момента последней архивации.

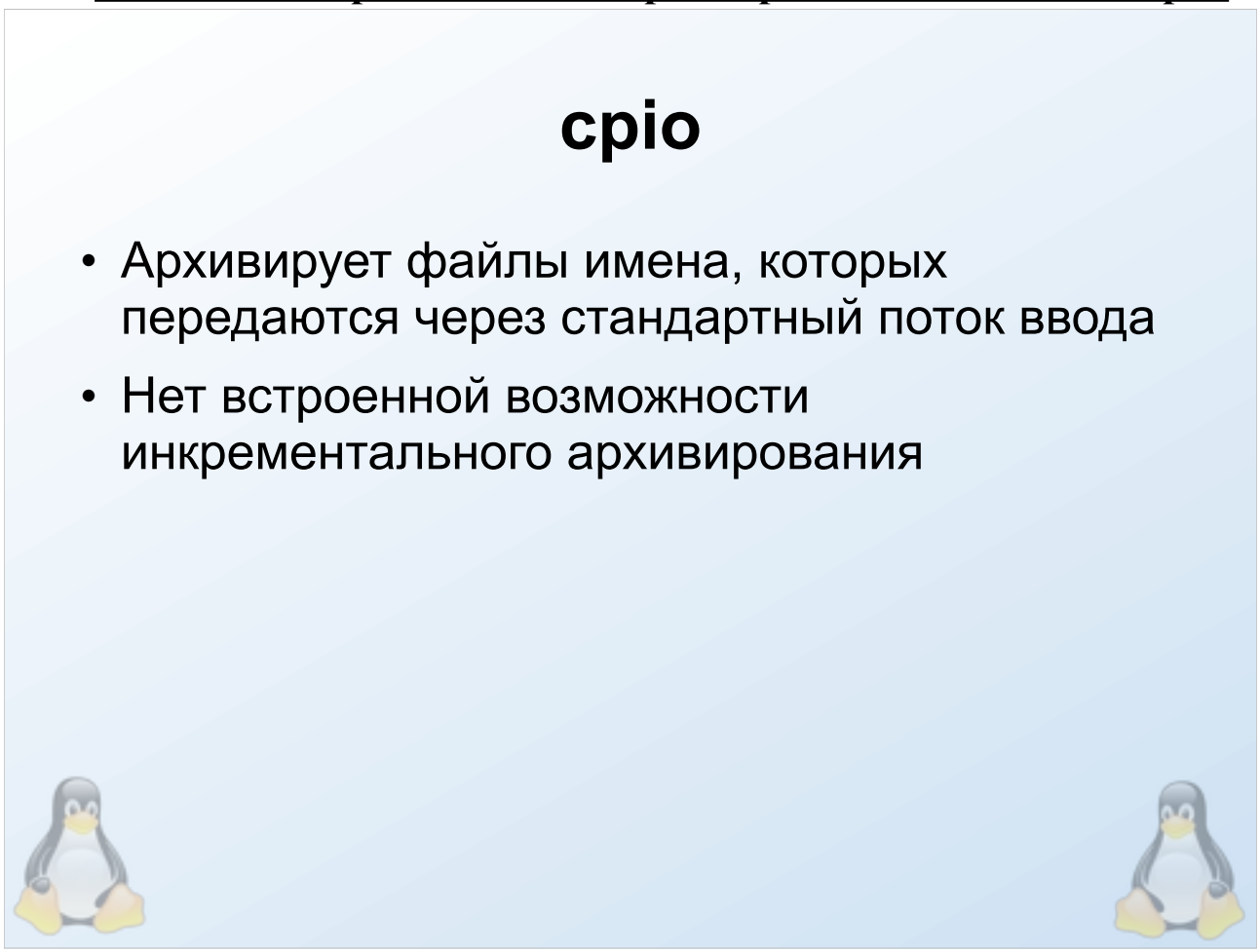
Пример:

```
tar cvf d1.1.tar --listed-incremental d1.inc d1/
```

Примечание: в данном примере создается архив с именем d1.1.tar и инкрементальный файл с именем d1.inc. Архивируется каталог d1

9. Для выполнения разностного архивирования команде `tar` необходимо подставлять всегда начальный инкрементальный файл, т.е. после полного архивирования сохранить инкрементальный файл в альтернативном месте, а перед каждым архивированием восстанавливать его.

6.4. Полное и инкрементальное архивирование с помощью cpio.



1. Команда `cpio` является вторым по значимости инструментом архивирования после `tar` , обладая при этом более гибкими возможностями по выбору файлов для архивирования, чем `tar` .

Примечание: Это связано с тем, что команда `cpio` *позволяет передавать через стандартный поток ввода имена файлов как для архивирования, так и имена архивов для извлечения из них файлов.*

2. Команда `cpio` способна работать в трех режимах, определяемых опциями:
    1. `-o` - для копирования в архив (copy-out), в котором архивируемые файлы помещаются в архив, а сам поток байт архива копируется в выходной (output) файл;
    2. `-i` - для копирования из архива (copy-in), в котором файлы извлекаются из архива, который передается команде на вход (input);
    3. `-p` - проходном (pass-through), при использовании которого файлы копируются из одного каталога в другой без реального создания архива.
3. Чаще всего список файлов, которые должны быть помещены в архив, передается на стандартный поток ввода команды `cpio` с помощью команды `find` .
4. В `cpio` нет опций для инкрементального копирования, но организовать такое копирование можно самому. Для этого

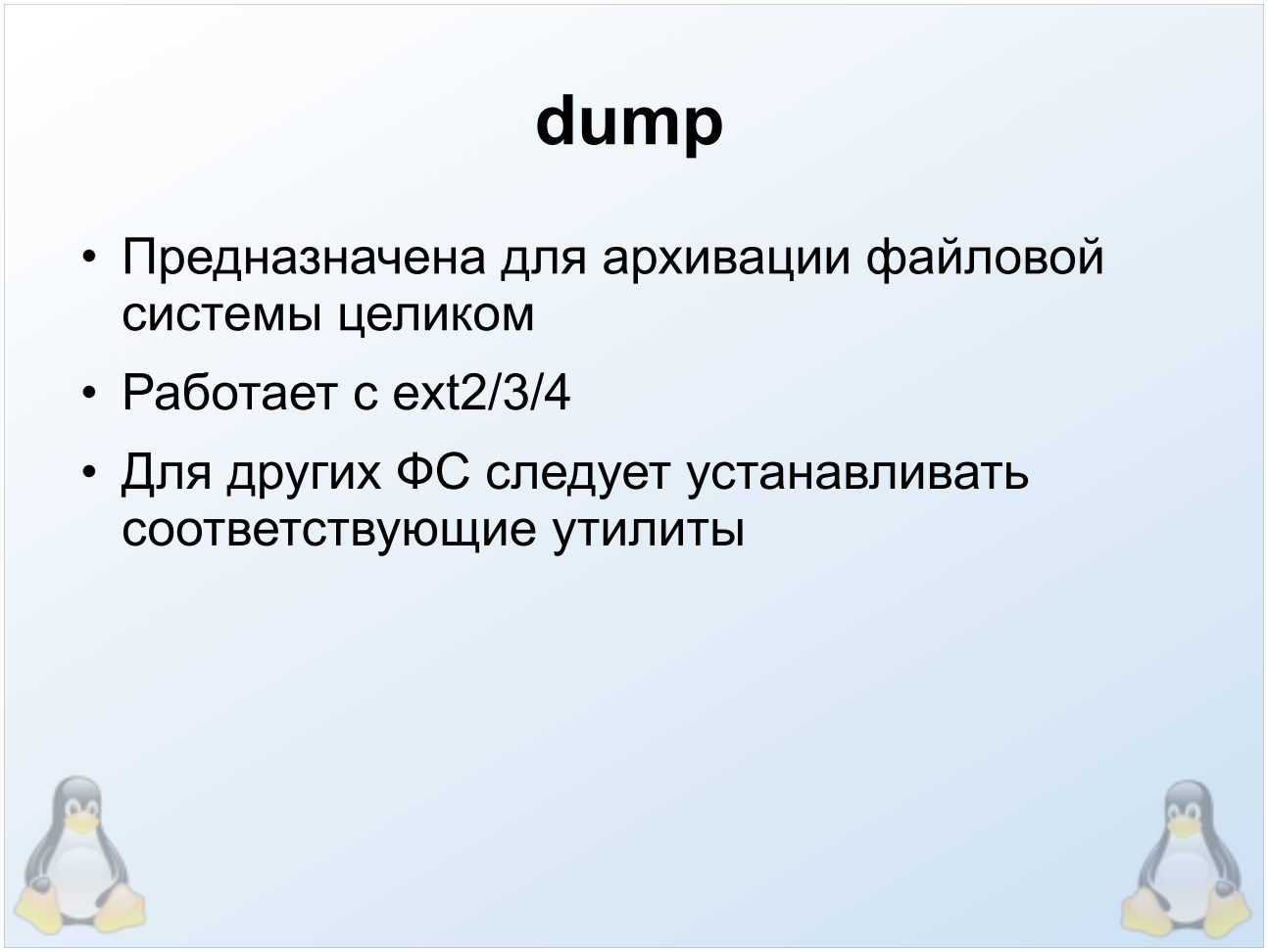
1. Поместить все файлы в архив, для полного архивирования;
2. Создать файл, содержащий время архивирования;

Пример:

```
find directory -depth|cpio -oF dir.0.cpio && date +%y%m%d%k%M.%S > 
cpio.inc
```

5. При последующих вызовах `cpio` необходимо произвести следующие действия:
   1. Извлечь время архивирования из файла, и сравнить время модификации файлов с этим временем;
   2. Поместить найденные файлы в инкркментальный архив.

6.5. Инкрементальное архивирование с помощью dump.



1. Утилита `dump` архивирует файлы из файловых систем ext.
2. Восстановление данных производится утилитой `restore`.

3. В `dump` имеется понятие уровней инкрементального архивирования.
4. Всего имеется 10 уровней:
   1. 0 уровень – полное архивирование файловой системы;
   2. 1-9 уровень – архивирование данных, изменившихся после архивации на предыдущих уровнях;
5. Инкрементальное архивирование применяется ко всей файловой системе целиком.
6. Для отслеживания уровней и дат архивирования используется файл `/etc/dumpdates`.
7. Опция `-u` заставляет обновлять `/etc/dumpdates`.

Пример: инкрементального архивирования.

Понедельник	*Вторник*	*Среда*	*Четверг*	*Пятница*
0	1	2	3	4

8. Опция `-f` позволяет указать, в какой файл производится архивирование. Файл это локальный файл, либо файл на удаленной машине.

Пример:

```
#dump 0uf backupsrv:/dev/st0 /
```

Примечание: в этом примере архивирование производится на компьютер с именем backupsrv.

9. Команда `restore` работает в нескольких режимах задаваемых опциями:
    1. `-C` – режим сравнения файлов из архива с файлами на диске.
    2. `-i` – переход в интерактивный режим восстановления. Для просмотра доступных команд в этом режиме используйте команду help.
    3. `-P` – создает файл Quick File Access без реального восстановления.

Примечание: Quick File Access это файл, в котором сохраняется логическая позиция на ленте.

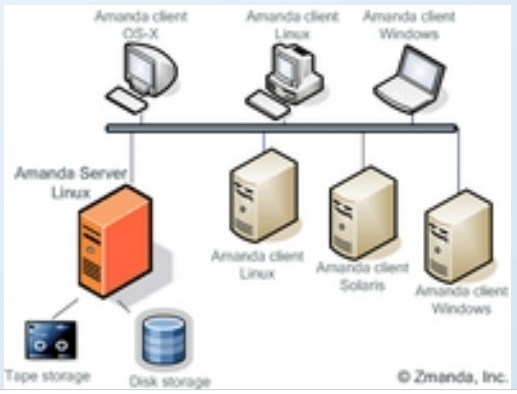
    4. `-R` – восстановление файловой системы с запросом дополнительных томов.
    5. `-r` – восстановление файловой системы целиком.
    6. `-t` – просмотр содержимого архива.
    7. `-x` – восстанавливать указанные после этой опции файлы.

6.6. Система AMANDA.

6.6.1. Настройка клиента AMANDA.

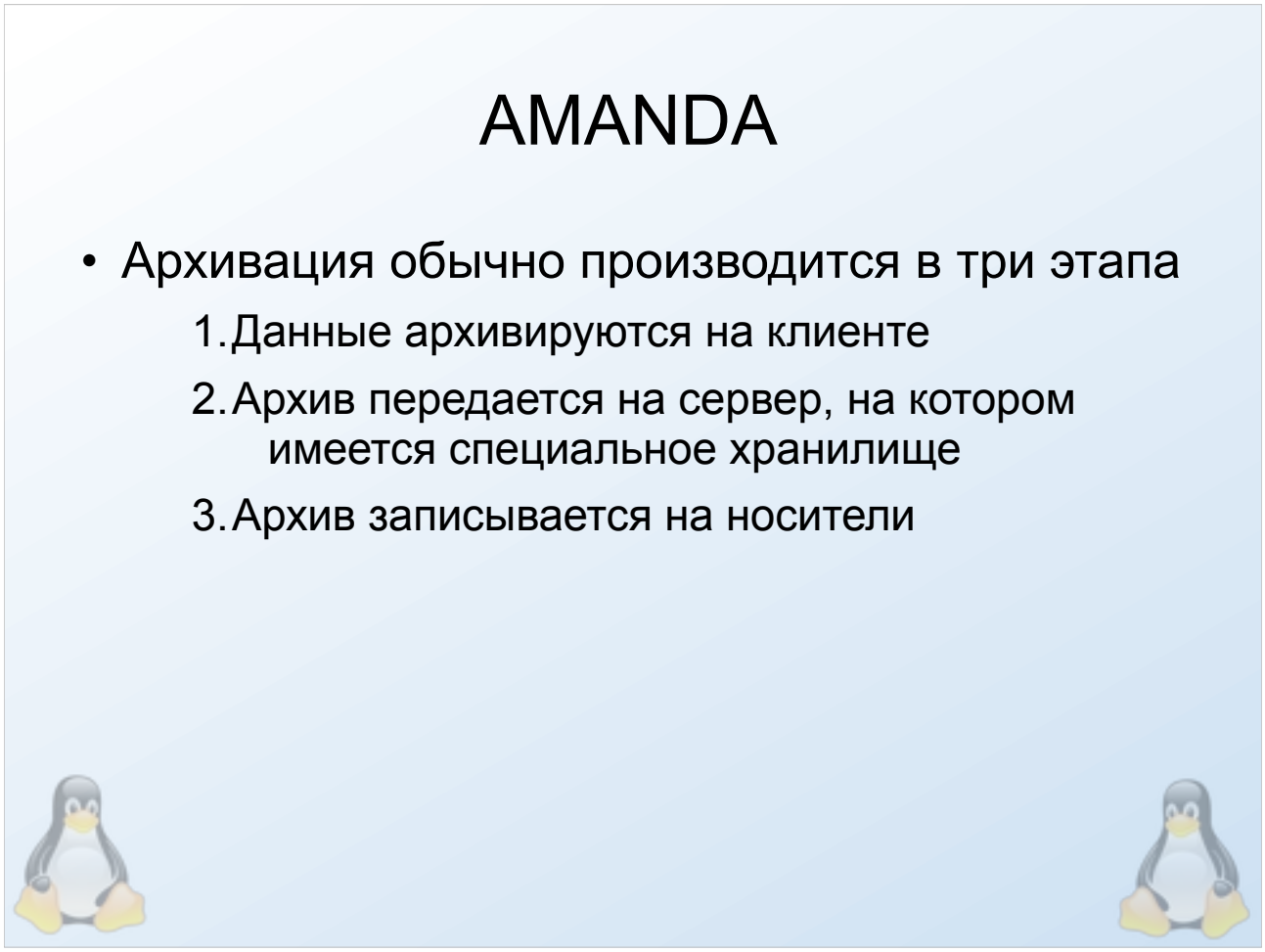
AMANDA

- AMANDA — инструмент для автоматизации архивации
  - Централизованное архивирование
  - Встроенный планировщик
  - Использует свои протоколы для передачи данных по сети



1. Система AMANDA (Advanced Maryland Automatic Network Disk Archiver) позволяет настроить централизованное архивирование данных в сети.
2. AMANDA сама по себе не является архиватором. Это оболочка для управления программами резервного копирования. По умолчанию это обычно dump/restore.
3. Данный пакет объединяет различные инструменты, предназначенные для выполнения резервного копирования в сети, и ориентирован в основном на работу в сетях небольшого и среднего размера, но может применяться и в больших сетях.
4. Для взаимодействия между сервером и клиентом применяются собственные протоколы. NFS, rshd и другие подобные протоколы при работе AMANDA не используются.

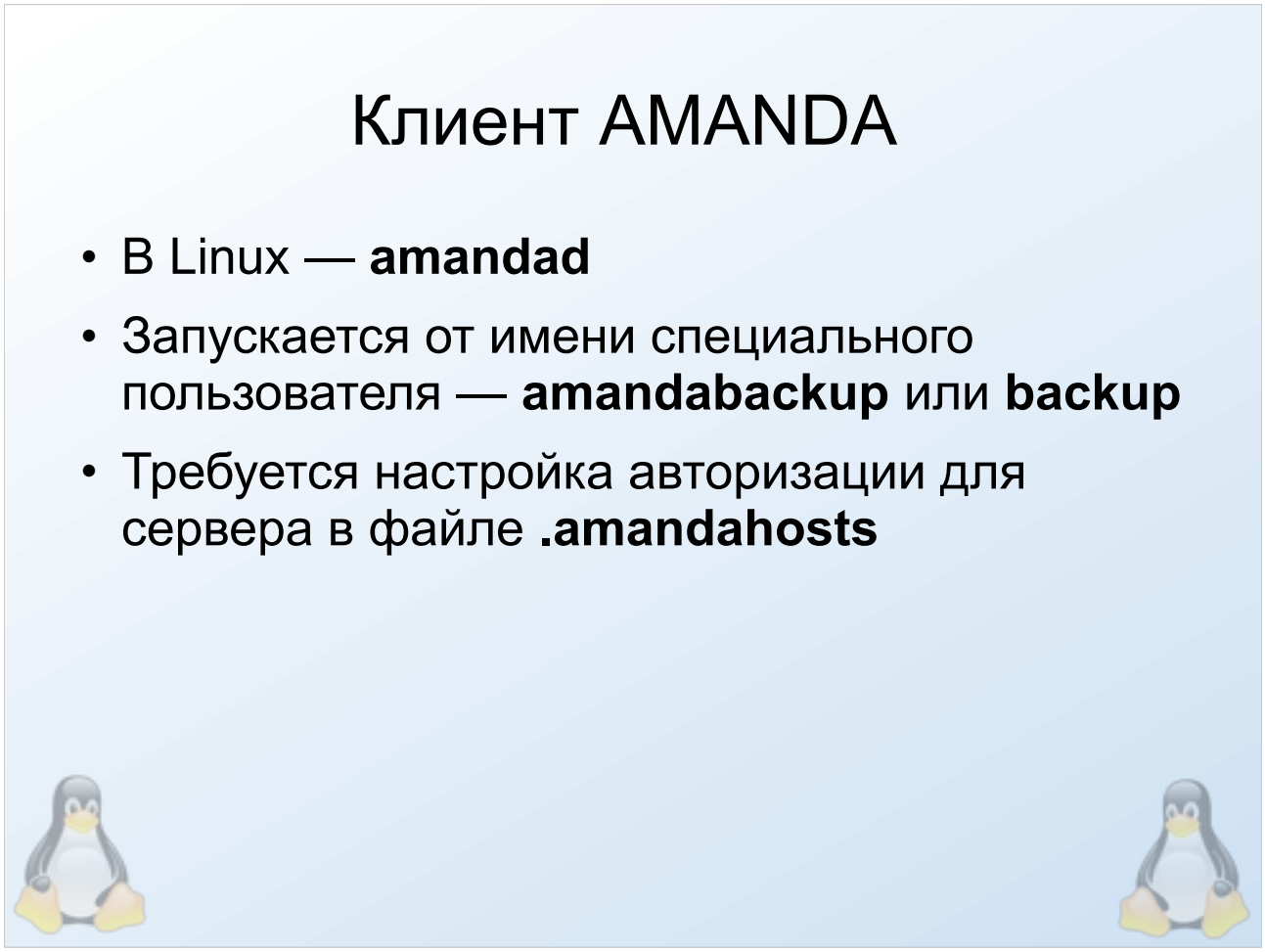
Примечание: При выполнении резервного копирования данных, расположенных на компьютерах под управлением Windows, AMANDA использует программу smbclient и стандартный сервер SMB/CIFS.)



5. AMANDA не только поддерживает сетевые соединения, но и выступает в роли инструмента планирования. Эта возможность помогает осуществлять резервное копирование, а для сети большого размера планирование является неотъемлемой частью работ по администрированию систем.
6. AMANDA позволяет указать, какой компьютер должен участвовать в резервном копировании в конкретный момент времени и должно ли выполняться полное или инкрементное копирование данных.
7. AMANDA в обычных условиях сначала копирует данные с клиентской машины на жесткий диск сервера резервного копирования, а затем записывает эти файлы на ленту.

Примечание: При необходимости вы можете настроить AMANDA так, что данные будут непосредственно записываться на ленту, но такая конфигурация снизит производительность программ.

8. AMANDA осуществляет резервное копирование, инициируемое сервером, поэтому на компьютере, выступающем в роли клиента, должна выполняться программа-сервер.
9. Данная программа, предназначенная для работы в системах Linux и UNIX, поставляется в составе пакета AMANDA и называется `amandad`.
10. Обычно эта программа-сервер обычно запускается с помощью суперсервера.



11. Соответствующая запись в конфигурационном файле `/etc/inetd.conf` имеет следующий вид:

```
amanda dgram udp wait amanda amandad amandad
```

Примечание: Данная запись запускает сервер amandad от имени пользователя amanda. Учетная запись, соответствующая этому пользователю, должна присутствовать в системе. При необходимости вы можете изменить данную запись в соответствии с особенностями системы, например, вам, возможно, придется указать полный путь к исполняемому файлу amandad.

12. Если в вашей системе используется суперсервер `xinetd`, вы должны будете создать запись в его конфигурационном файле.

13. Для запуска сервера AMANDA с помощью суперсервера в файл `/etc/services` необходимо включить специальную запись.

Пример: Она может выглядеть следующим образом:

```
amanda 10080/udp
```

Примечание: В результате клиент резервного копирования AMANDA станет доступен для сервера резервного копирования.

14. Остальные действия по настройке выполняются на компьютере, выступающем в роли сервера резервного копирования.

15. При выполнении резервного копирования данных, содержащихся на узлах сети, необходимо также создать резервную копию сервера резервного копирования. Для этого на сервере обычно устанавливается программное обеспечение клиента резервного

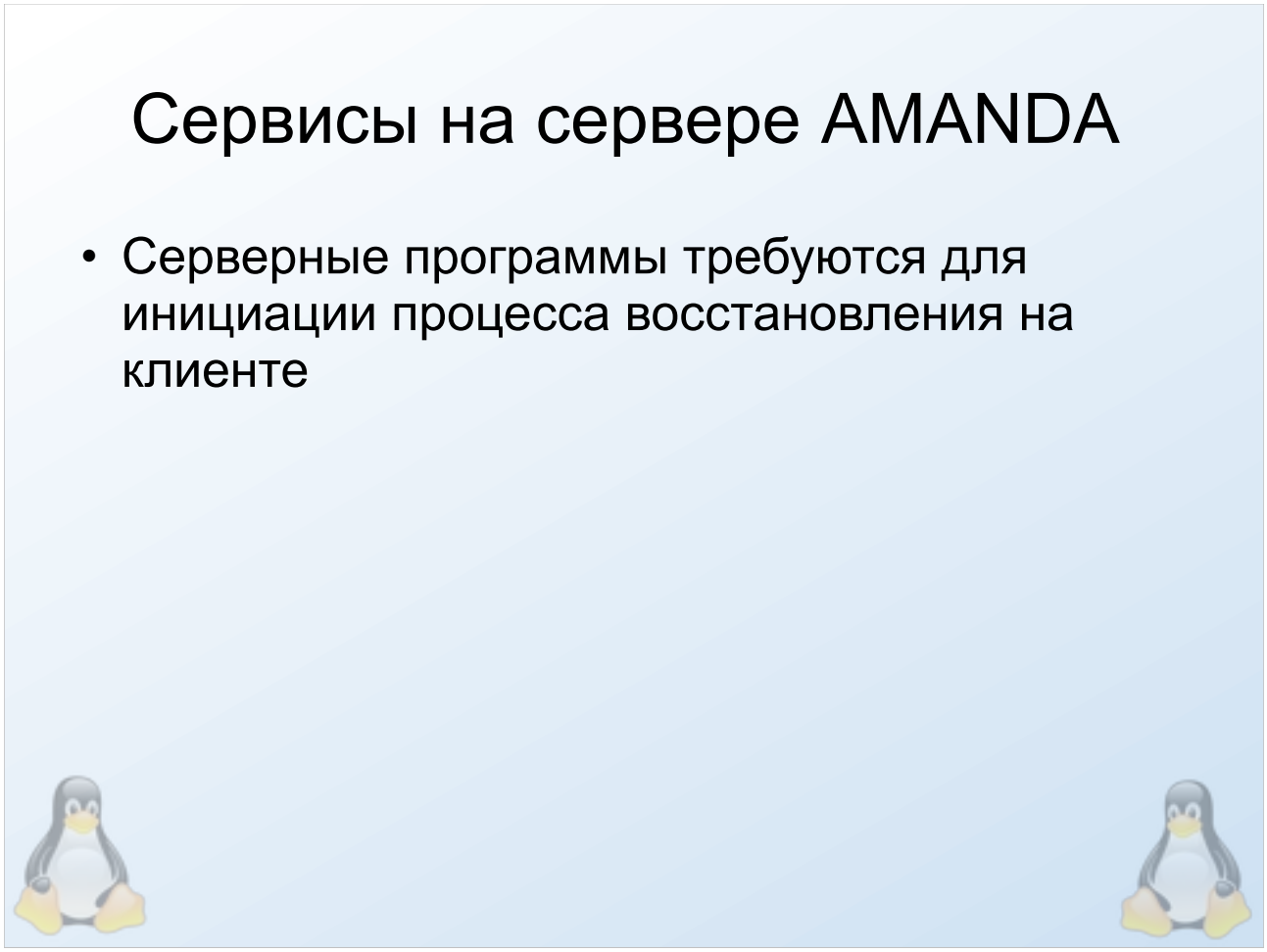
копирования.

16. Для работы клиента резервного копирования необходим файл авторизации с именем `.amandahosts`, находящийся в рабочем каталоге пользователя, который запускает AMANDA. В этом файле должны быть указаны полностью определенное доменное имя (FQDN) сервера резервного копирования и имя пользователя, разделенные пробелом или знаком табуляции.

Пример, приведенная ниже запись позволяет пользователю amanda на сервере buserver.domain.ru создавать резервные копии данных.

```
buserver.domain.ru amanda
```

6.6.2. Настройка сервера резервного копирования AMANDA



1. Поскольку с точки зрения сетевого взаимодействия сервер резервного копирования действует как клиент, на этом компьютере не нужно программное обеспечение сервера.

Примечание: Тем не менее следует заметить, что AMANDA поддерживает восстановление данных по инициативе клиента, поэтому в пакете AMANDA содержатся две серверные программы, предназначенные для выполнения на сервере резервного копирования. В результате пользователь, работающий на компьютере, выполняющем функции клиента резервного копирования, может просматривать имеющиеся данные и инициировать процесс восстановления информации.

2. Серверные программы на сервере резервного копирования обычно запускаются посредством суперсервера. Соответствующие записи в файле `/etc/inetd.conf` имеют следующий вид:

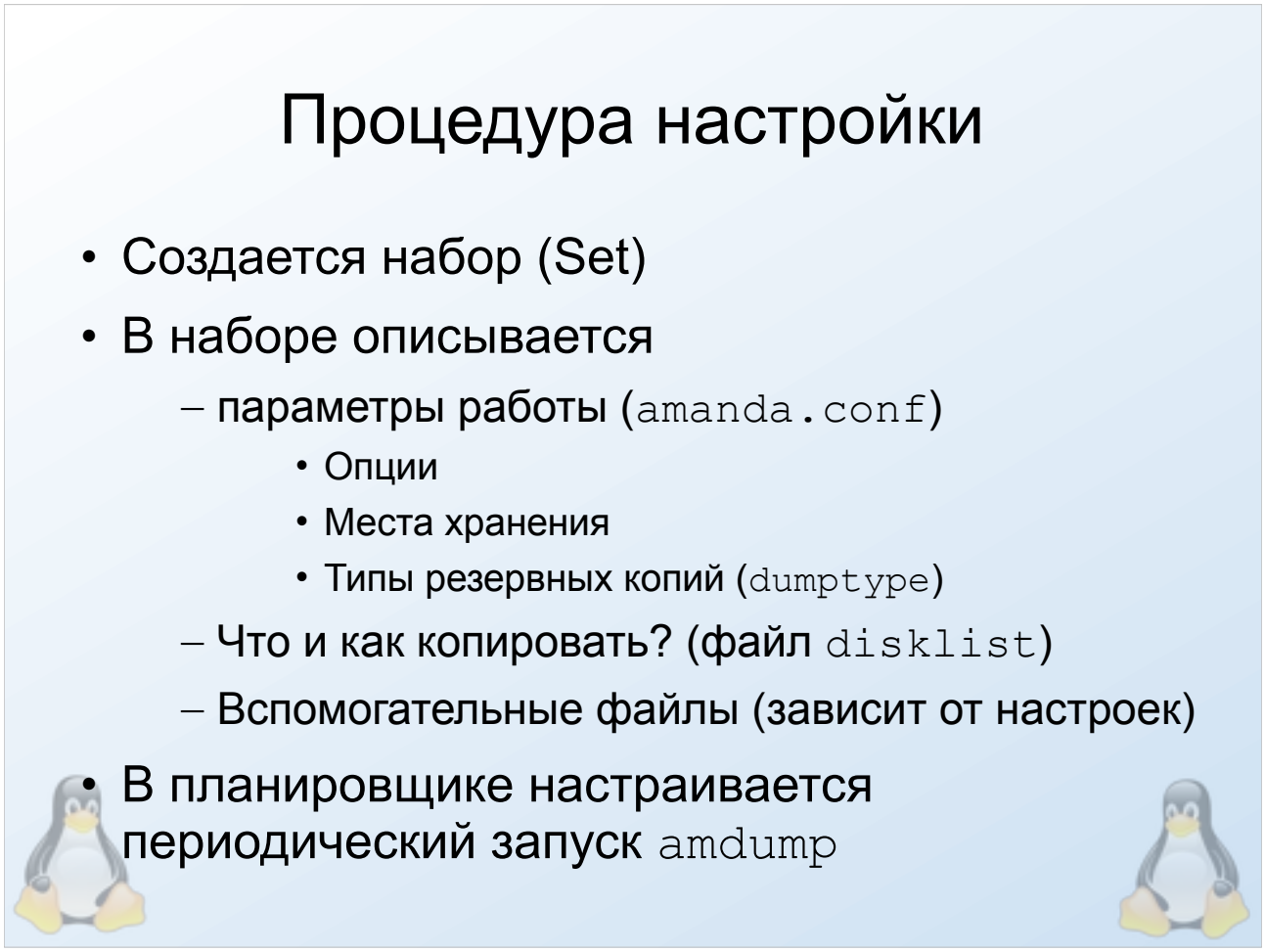
```
amandaidx stream tcp nowait amanda amindexd amindexd
amidxtape stream tcp nowait amanda amidxtaped amidxtaped
```

Примечание: Как и для программ, выполняющихся на клиенте резервного копирования, вам, возможно, придется указать полный путь к исполняемым файлам.

3. Чтобы указанные программы могли запускаться посредством суперсервера, надо включить в файл `/etc/services` следующие строки:

```
amandaidx 10082/tcp
amidxtape 10083/tcp
```

4. Пользователь, запускающий программы AMANDA, должен иметь право читать и записывать информацию на магнитную ленту. В противном случае AMANDA не будет иметь доступа к устройству и не сможет создавать резервные копии данных.



5. AMANDA может иметь несколько независимых наборов конфигураций (Set).
6. Особенности работы набора определяются содержимым конфигурационных файлов `amanda.conf`, которые обычно находятся в подкаталогах каталога `/etc/amanda`.

Пример: конфигурационный файл, описывающий правила ежедневного копирования, может размещаться в каталоге `/etc/amanda/Daily`, а файл, определяющий создание архивов веб серверов, — в каталоге `/etc/amanda/WebArchive`.

Примечание: Если вы использовали для инсталляции AMANDA исходные коды, в вашем распоряжении имеется образец конфигурационного файла. Он находится в каталоге example инсталляционного пакета или в /usr/share/doc/amanda-server/examples/ если вы устанавливали из пакета.

7. В состав файла `amanda.conf` входит набор строк, и каждая из них начинается с ключевого слова, за которым следует одно или несколько значений.

Пример: запись, указывающая, как долго длится процесс полного копирования содержимого всей сети, выглядит следующим образом: `dumpcycle 4 weeks`

8. Некоторые из записей могут занимать несколько строк, тогда они помещаются в фигурные скобки и определяют набор связанных между собой опций.
9. Для большинства опций можно принять значения, установленные по умолчанию.
10. Опции, которые возможно, необходимо изменить, описаны ниже.
    1. `org` – с помощью данной опции задается название организации, которое затем

указывается в отчетах, генерируемых AMANDA. Значение опции org никак не влияет на процесс создания резервной копии.

2. `mailto` – о выполненных действиях AMANDA сообщает, посылая письмо по адресу, указанному в качестве значения этой опции. При необходимости вы можете задать несколько адресов, разделив их пробелами.
3. `dumpuser` – посредством данной опции указывается имя пользователя, инициирующего процедуру резервного копирования. При инсталляции AMANDA это имя задается с помощью опции –`with-user`.
4. `dumpcycle` – в качестве значения этой опции указывается число дней, составляющих цикл резервного копирования.
5. `runspercycle` – периодичность запусков AMANDA.

Пример: Предположим, что цикл резервного копирования, указанный посредством опции `dumpcycle`, составляет четыре недели. Установив значение `runspercycle`, равное 20, вы сообщаете, что данные для копирования должны выбираться исходя из того, что AMANDA будет запускаться один раз в день в рабочие дни. Значение 4 говорит о том, что AMANDA будет запускаться один раз в неделю. (Заметьте, что реальный запуск AMANDA осуществляется с помощью инструмента `cron`. Значение опции `runspercycle` не влияет на периодичность запуска, оно лишь позволяет AMANDA планировать, какие данные должны быть скопированы на резервный носитель.)

6. `tapecycle` – значение данной опции определяет число магнитных лент, используемых в цикле резервного копирования. Учитывая, что некоторые носители могут быть повреждены, значение `tapecycle` должно быть несколько больше, чем значение опции `runspercycle`.
7. `tapetype` - если вы сообщите о том, какой тип накопителя вы используете, AMANDA сможет определить, как долго будет длиться запись информации.

Примечание: В конфигурационном файле, поставляемом в составе конфигурационного пакета, указано несколько типов накопителей. Если вы не найдете тип, соответствующий вашему устройству, вам придется выяснить эти сведения самостоятельно.

Сделать это можно с помощью утилиты `tapetype`, которая поставляется в составе пакета AMANDA, но по умолчанию не устанавливается. Перейдите в каталог `tape-src` инсталляционного пакета и задайте команду

```
make tapetype.
```

Затем установите в устройстве ненужную вам ленту и введите

```
./tapetype -f /dev/устройство
```

(где под устройством понимается файл, соответствующий накопителю на ленте). В результате выполнения данной команды вы получите набор значений для вашего устройства. Для завершения данной процедуры потребуется несколько часов, и все данные на ленте будут уничтожены.

Если в вашем накопителе реализована встроенная функция сжатия данных, вы можете увеличить значение длины ленты в полтора-два раза. При этом соблюдайте осторожность: если вы будете копировать данные, плохо поддающиеся сжатию, места на ленте может не хватить.

8. `tapedev` – данная опция задает файл устройства Linux, представляющий интерфейс *без перемотки* к накопителю на ленте. В большинстве случаев в качестве значения этой опции указывается имя `/dev/nst0` или `/dev/nht0`.

9. `netusage` – с помощью этой опции указывается максимальная пропускная способность линий, на которую может рассчитывать AMANDA при выполнении резервного копирования.
10. `labelstr` - значением этой опции является регулярное выражение, которое AMANDA использует для назначения имен лентам с резервными копиями данных. Имена лент учитываются при их подготовке.

Примечание: Особенности подготовки лент рассматриваются далее.

11. `tpchanger, changerfile и changerdev` – если в вашем накопителе предусмотрена автоматическая смена ленты, вы можете управлять устройством с помощью специальных файлов.

Примечание: Примеры этих файлов находятся в каталоге example инсталляционного пакета.

12. `infofile, logdir и indexdir` - расположение файлов журналов AMANDA задается с помощью опций `infofile` и `logdir`. Кроме того, опция `indexdir` определяет индексный файл, содержащий список скопированных файлов. Значения этих опций можно оставить без изменения.
13. `holdingdisk` – кроме установки значений приведенных выше опций, вам также надо описать область для хранения данных. Для этого используется опция `holdingdisk`, значение которой занимает несколько строк и состоит из ряда подопций. К ним относятся:
    1. `directory` - каталог для хранения файлов;
    2. `use` - пространство на диске, которое может быть использовано для хранения данных;
    3. `chunksize` – если на диске сервера резервного копирования недостаточно места, вы можете задать отрицательное значение, при этом файл, размер которого превышает абсолютное значение `chunksize`, непосредственно записывается на ленту, минуя область хранения. Положительные значения chunksize указывают на то, что большие файлы должны быть разбиты на части для размещения в области хранения. Такой подход удобно применять в тех случаях, когда файловая система на диске или особенности ядра не позволяют обрабатывать большие файлы.

Пример: конфигурационных файлов amanda.conf и . Предполагается, что архивация производится на диск.

```
# cat /etc/amanda/MySet1/amanda.conf
# Глобальные опции
org "My Org"
mailto "backup"
# Частота полной архивации
dumpcycle 1 weeks
# Количество лент
tapecycle 20
# Количество запусков за dumpcycle
runspercycle 5
# Формула dumpcycle*(tapecycle/runspercycle) определяет возраст самой
# старой резервной копии.
```

```
# tapecycle должно быть больше runspercycle !!!
runtapes 1
tpchanger "amandastore"
logdir   "/amandastore/MySet1/log"
indexdir "/etc/amanda/MySet1/index"
infofile "/etc/amanda/MySet1/curinfo"
labelstr "^MySet1-[0-9][0-9]*$"
tapetype HARD-DISK
tapedev "file:/amandastore/MySet1/vtapes"

# Определение ченджера
define changer amandastore {
  tpchanger "chg-disk:/amandastore/MySet1/vtapes"
  property "num-slot" "20"
  property "auto-create-slot" "yes"
}

# Определение ленты
define tapetype HARD-DISK {
  comment "Dump onto hard disk"
  length 20480 mbytes # specified in mbytes to get the exact size of 20GB
}

# Определения типов копирования
define dumptype global {
   comment "Global definitions"
   index yes
   record yes
}

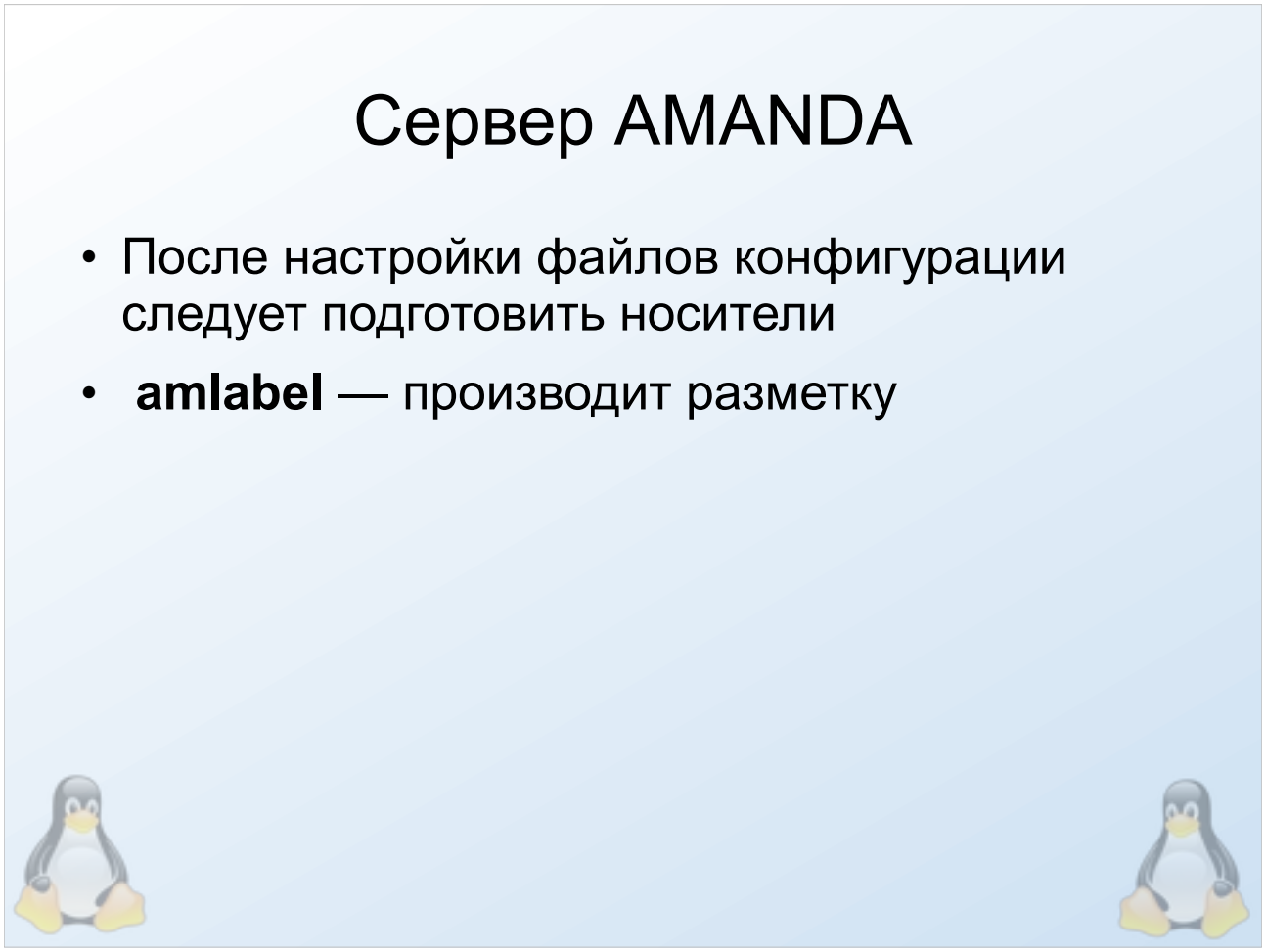
define dumptype hard-disk-dump {
   global
   comment "Back up to hard disk instead of tape - using dump"
   holdingdisk no
   index yes
   priority high
}

define dumptype hard-disk-tar {
```

```
   hard-disk-dump
   comment "Back up to hard disk instead of tape - using tar"
   program "GNUTAR"
}

# Определения интерфейсов
define interface local {
   comment "a local disk"
   use 1000 kbps
}

define interface enp0s3 {
   comment "1000 Mbps ethernet"
   use 100 Mbps
}
# cat /etc/amanda/MySet1/disklist
localhost /etc hard-disk-tar
```



11.После подготовки конфигурационных файлов необходимо подготовить хранилище данных.

Пример: подготовка диска к хранению данных

```
# mkdir -p /amandastore/MySet1/vtapes
# mkdir -p /amandastore/MySet1/log
# chown -R backup:backup /amandastore/MySet1
```

12.Подготовка лент осуществляется с помощью утилиты `amlabel`. Эта утилита должна запускаться от имени пользователя, который выполняет резервное копирование.

Пример: Вызов `amlabel` может выглядеть следующим образом:

```
# sudo -u backup bash -c 'for i in {0..20} ; do amlabel MySet1 MySet1-$i slot $i; done'
Reading label...
Found an empty tape.
Writing label 'MySet1-0'...
Checking label...
Success!
Reading label...
Found an empty tape.
```

```
Writing label 'MySet1-1'...
<...>
```

13. В большинстве случаев AMANDA копирует данные из сети на несколько лент. Чтобы вы могли различать ленты, вам необходимо разработать схему их именования.
14. Определение типов резервных копий в файле `amanda.conf`, задается записями `dumptype`. Они определяют, как должно выполняться резервное копирование содержимого клиентского компьютера или отдельной файловой системы.
15. В составе записи `dumptype` могут указываться следующие опции:
    1. `compress [client | server] [best | fast | none]` – тип сжатия, выполняемого клиентом или сервером копирования. Тип сжатия выбирается в зависимости от используемого процессора и сетевых ресурсов.

Примечание: Значение `best` указывает на то, что необходимо выполнить наиболее эффективное сжатие, для которого требуется большое количество ресурсов процессора. Значение `fast` задает быстрое, но менее эффективное сжатие. Значение `none` запрещает сжатие данных.

    2. `exclude [list] "строка"` – в зависимости от того, указано ли значение `list`, AMANDA включает заданную строку в качестве значения опции `--exclude` или `--exclude-from` утилиты `tar`.
    3. `holdingdisk` *логическое_значение* – задавая логическое значение, вы можете указать AMANDA, следует ли использовать область диска для хранения данных.
    4. `index` *логическое_значение* – задавая логическое значение равным `yes` или `no`, вы сообщите AMANDA о том, следует ли создавать перечень файлов, скопированных на резервный носитель.
    5. `kencrypt` *логическое_значение* – данная опция позволяет указать, следует ли кодировать данные, передаваемые по сети, с помощью протоколов Kerberos. Значение, равное `yes`, можно указывать только в том случае, когда сеть сконфигурирована для работы с Kerberos.
    6. `program "строка"` – AMANDA может использовать либо утилиту `tar`, либо программу `dump`, ориентированную на работу с конкретной операционной системой. Данная опция позволяет указать, какую из этих программ следует использовать. По умолчанию AMANDA работает с программой `dump` (значение DUMP данной опции). Чтобы задать использование `tar`, необходимо, чтобы значение опции было равно GNUTAR. (При работе с Samba по умолчанию применяется утилита `tar`.)
    7. `skip-incr` *логическое_значение* – если значение данной опции равно `true`, то при инкрементном копировании файловая система, для которой указан этот тип резервной копии, не учитывается.
16. Большинство определений типов резервных копий начинается с имени другого определения. Это означает, что новое определение создано на основе существующего и для него справедливы значения опций базового определения.

Пример: в конфигурационном файле `amanda.conf`, поставляемом в составе инсталляционного пакета, присутствует определение с именем `global`, включаемое в другие определения. Заметьте, что при выполнении одной операции резервного копирования могут быть созданы копии различных типов. Например, вы можете скопировать важные

системные файлы, не используя сжатие, но сжимать при копировании менее важную информацию. Вы также можете использовать утилиту `dump` для копирования разделов ext2fs, а утилиту `tar` — для копирования разделов ReiserFS.

17. Определение данных для копирования указывается в файле `disklist`, который находится в том же каталоге, что и `amanda.conf`.

Примечание: в файле amanda.conf содержатся важные опции, управляющие процессом копирования, но отсутствуют сведения о клиентах резервного копирования или каталогах, содержимое которых необходимо записать на резервный носитель.

В составе инсталляционного пакета AMANDA содержится пример файла disklist. Очевидно, что рабочий вариант этого файла, созданный с учетом конфигурации вашей системы, будет существенно отличаться от образца.

18. Содержимое файла `disklist` представляет собой набор записей, каждая из которых содержится в отдельной строке и состоит из трех полей. В этих полях указывается

    1. имя компьютера, выступающего в роли клиента резервного копирования,
    2. область для копирования;
    3. тип резервной копии для этой области.

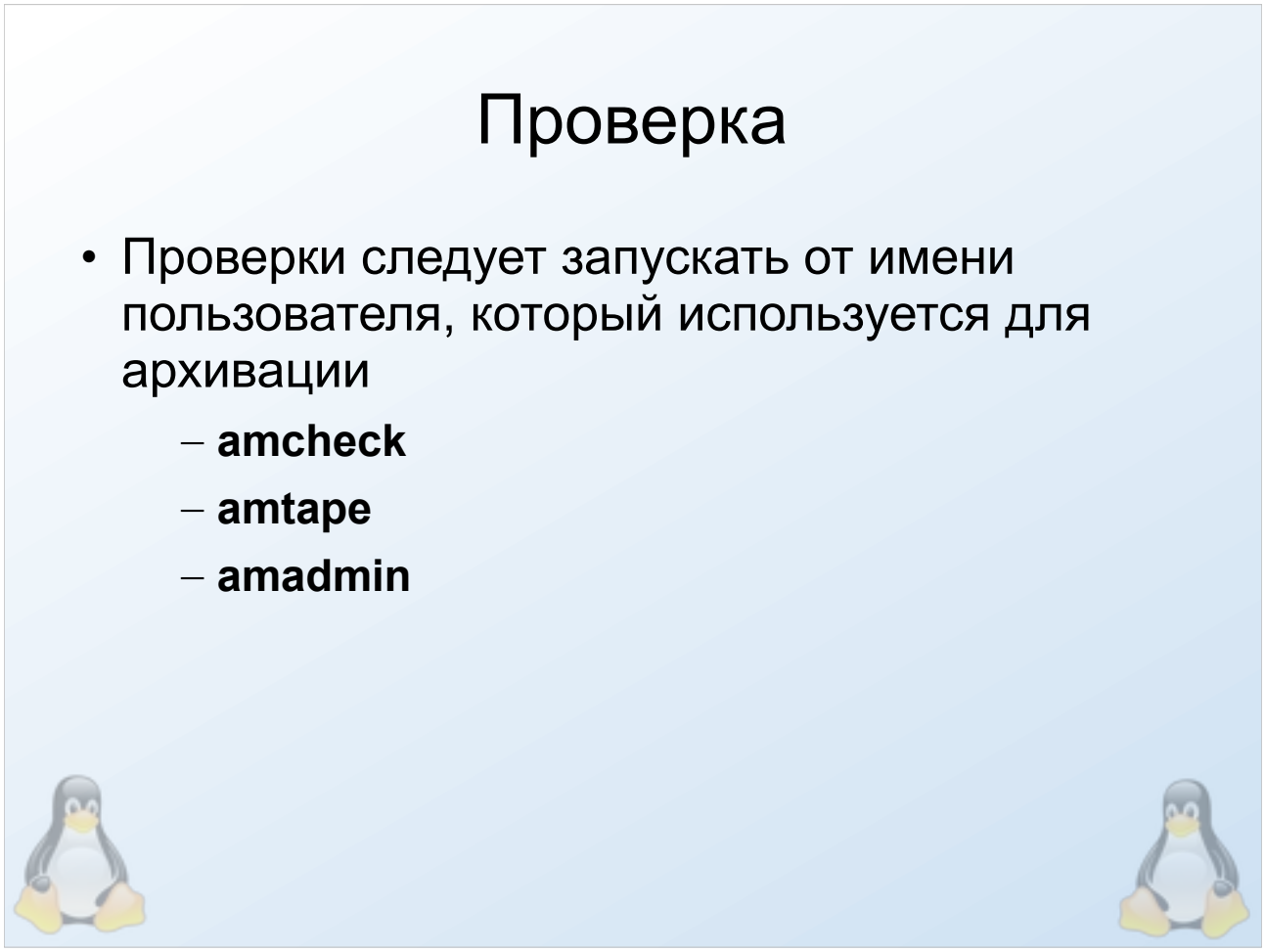
Пример: В качестве области для копирования может быть указано имя устройства (например, `/dev/hda2` или `hda2`) либо точка монтирования файловой системы (например, `/home`).

19. Строки, начинающиеся с символа #, содержат комментарии.

Пример файла `disklist`

```
# Создание резервной копии сервера резервного копирования
buserver.domain.ru / root-tar
buserver.domain.ru /var user-tar
buserver.domain.ru /hold holding-disk
# Создание резервной копии клиента Linux или UNIX
buclient.domain.ru / root-tar
buclient.domain.ru /home user-tar
# Создание резервной копии клиента Windows
buserver.domain.ru //WINPC/DRIVEC user-tar
```

Примечание: Раздел `/hold` компьютера `buserver.domain.ru` содержит область для хранения данных. В определении соответствующего типа резервной копии указано значение `no` опции `holdingdisk`, что предотвращает использование этой области для временного хранения своего же содержимого. Если в данном разделе не содержится ничего, кроме области хранения данных, вы можете исключить его из процесса копирования. Для копирования данных клиента Windows указывается имя компьютера Linux или UNIX, на котором установлен и выполняется сервер Samba, а также NetBIOS-имя клиента Windows (WINPC) и имя разделяемого объекта (DRIVEC). В листинге в качестве компьютера, на котором установлен продукт Samba, указан сам сервер резервного копирования, но при необходимости вы можете задать имя другого компьютера. Чтобы создать резервную копию устройства на компьютере под управлением Windows, его не нужно монтировать; AMANDA обращается к системе Samba для использования `smbclient`. Samba использует `smbclient` так же, как обычный клиент резервного копирования утилиту `tar` или `dump`. В файловой системе компьютера, на котором выполняется Samba, необходимо создать файл `/etc/amandapass`. В этом файле следует указать имя разделяемого объекта и пароль. В качестве пользовательского имени AMANDA передает имя SAMBA, поэтому при работе с Windows NT, 2000 или XP этот пользователь должен присутствовать в системе. Для того чтобы изменить имя пользователя, применяемое по умолчанию, вам надо при конфигурировании пакета AMANDA задать это имя в качестве значения опции `—with-samba-user`.



20.После настройки выполните проверку ваших настроек.

1. Команда `amcheck` — проверка сервера, хостов, дисков.
2. Команда `amtape` — управление лентами.
3. Команда `amadmin` — управление AMANDA.

Пример: проверка на наличие ошибок и инвентаризация «лент» (слотов).

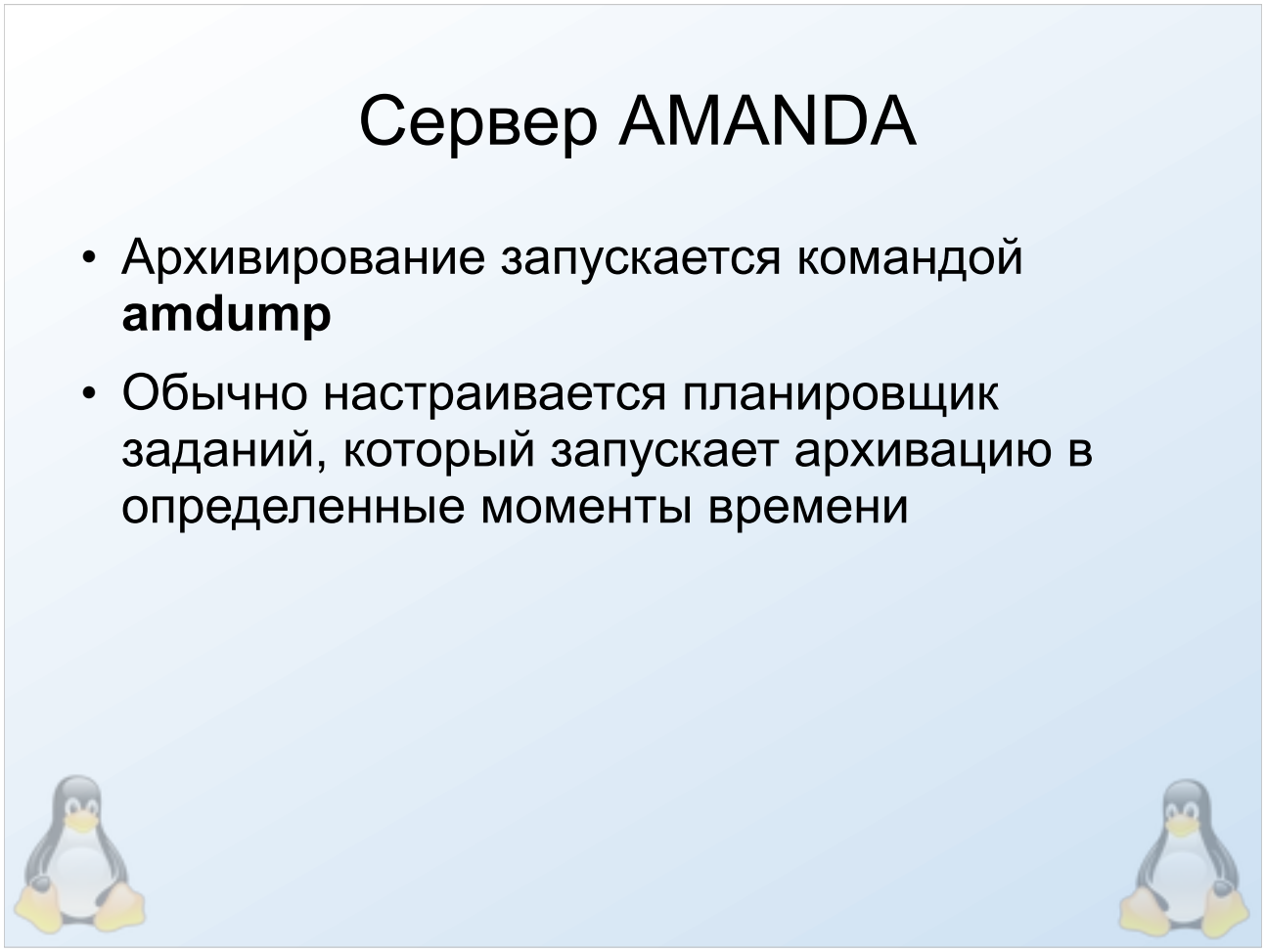
```
# sudo -u backup amcheck MySet1
Amanda Tape Server Host Check
-----------------------------
slot 0: volume 'MySet1-0'
Will write to volume 'MySet1-0' in slot 0.
NOTE: skipping tape-writable test
NOTE: storage 'MySet1': retentions-tapes (19) <= runspercycle (20)
NOTE: conf info dir '/etc/amanda/MySet1/curinfo' does not exist
      it will be created on the next run
NOTE: index dir '/etc/amanda/MySet1/index' does not exist
      it will be created on the next run
Server check took 0.421 seconds
```

```
Amanda Backup Client Hosts Check
--------------------------------
Client check: 1 host checked in 2.329 seconds.  0 problems found.
(brought to you by Amanda 3.5.1)

# sudo -u backup amtape MySet1 inventory
slot 0: label MySet1-0 (current) [retention-no]
slot 1: label MySet1-1 [retention-no]
slot 2: label MySet1-2 [retention-no]
slot 3: label MySet1-3 [retention-no]
slot 4: label MySet1-4 [retention-no]
slot 5: label MySet1-5 [retention-no]
slot 6: label MySet1-6 [retention-no]
slot 7: label MySet1-7 [retention-no]
slot 8: label MySet1-8 [retention-no]
slot 9: label MySet1-9 [retention-no]
slot 10: label MySet1-10 [retention-no]
slot 11: label MySet1-11 [retention-no]
```

```
DUMPUSER              "backup"
PRINTER               ""
MAILER                "/usr/bin/mail"
TAPEDEV               "file:/amandastore/MySet1/vtapes"
TPCHANGER             "amandastore"
CHANGERDEV            ""
# sudo -u backup amadmin MySet1 info

Current info for localhost /etc:
  Stats: dump rates (kps), Full:   -1.0,  -1.0,  -1.0
                     Incremental:   -1.0,  -1.0,  -1.0
           compressed size, Full: -100.0%,-100.0%,-100.0%
                     Incremental: -100.0%,-100.0%,-100.0%
  Dumps: lev datestmp  tape             file   origK   compK secs
```

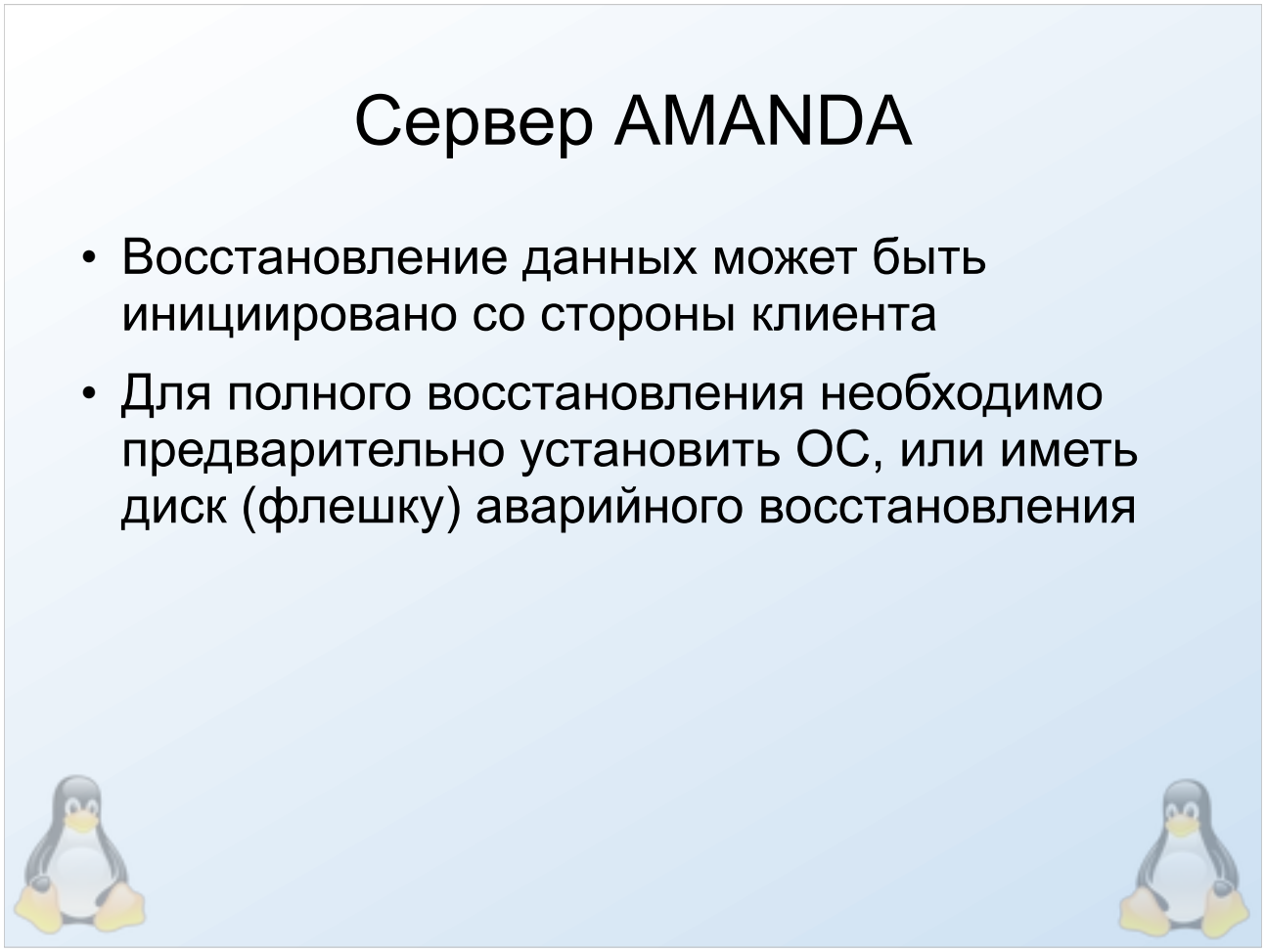


21. Для того чтобы инициировать процесс создания резервной копии с помощью AMANDA, необходимо запустить на сервере резервного копирования программу `amdump`. Лучше всего запускать ее с помощью инструмента `cron` в то время, когда нагрузка на сеть минимальна, например ночью.

Пример, команда может выглядеть так:

```
amdump Daily.
```

Примечание: Перед запуском `amdump` вам необходимо установить на устройстве ленту, подготовленную так, как было описано выше. В результате выполнения данной команды AMANDA обнаружит области, предназначенные для копирования, обработает их содержимое и запишет на магнитную ленту. Процедура резервного копирования не обязательно будет охватывать все компьютеры в сети и даже все разделы на одном компьютере. В зависимости от значения опции dumpcycle и других опций подобного назначения, указанных в конфигурационном файле, а также от типа устройства резервного копирования, AMANDA может принять решение скопировать данные лишь с нескольких компьютеров или вместо полного копирования выполнить инкрементное копирование. Если продолжительность цикла резервного копирования не слишком мала, в течение этого цикла AMANDA выполнит резервное копирование всех компьютеров в сети.



22. Действия по восстановлению данных можно условно разделить на две описанные ниже категории.

1. Частичное восстановление. При выполнении частичного восстановления необходимо извлечь с резервного носителя лишь некоторые файлы.

Пример, пользователю могут понадобиться файлы, которые были по ошибке удалены на прошлой неделе, или у вас может возникнуть необходимость просмотреть старые файлы протоколов. Для этого надо выполнить действия, обратные созданию резервной копии. Например, если для создания копии вы использовали опцию —create утилиты tar, то для восстановления файлов вам надо задать опцию —extract и указать имена файлов или каталогов, которые вы собираетесь восстановить.

Примечание: Если вы смонтировали некоторую файловую систему и выполняли резервное копирование по инициативе сервера, вам следует убедиться, что программа-сервер, которая выполняется на компьютере, выступающем в роли клиента резервного копирования, предоставляет серверу резервного копирования право записи данных. Это требование не выдвигалось при создании резервной копии.

2. Полное восстановление. В этом случае предполагается восстановление всех файлов на диске или по крайней мере тех данных, которые нужны для загрузки компьютера. Необходимость в полном восстановлении может возникнуть при повреждении жесткого диска или содержащихся на нем программ.

Пример: полное восстановление придется производить после того, как от имени пользователя root была выполнена команда `rm -rf /`.

ВНИМАНИЕ Полное восстановление может понадобиться для того, чтобы воспроизвести состояние системы, которое было до вмешательства злоумышленника в ее работу. Однако в этом случае необходимо действовать очень осторожно, так как в результате восстановления данных потеряется информация о характере атаки. Перед тем как восстанавливать состояние системы, вам необходимо проанализировать недостатки системы, которыми смог воспользоваться хакер для ее взлома.

23. Полное восстановление данных представляет собой сложную задачу, так как вам необходимо найти способ записи данных на компьютер, на котором нет работоспособных программ.

24. Чтобы решить эту проблему, многие системные администраторы создают минимальную конфигурацию системы на дискете, компакт-диске или другом носителе. Для работы в сети этот вариант системы должен содержать средства поддержки сетевого взаимодействия, а также программы, обеспечивающие работу клиента резервного копирования.

Примечание: Если в вашей сети присутствует сервер резервного копирования под управлением Linux и большое число клиентов резервного копирования под управлением Windows 9x/Me, вы можете использовать для восстановления данных минимальную конфигурацию системы Linux. В этой системе должны присутствовать средства Samba, которые вы используете, чтобы запустить сервер SMB/CIFS. Этот сервер необходим для восстановления файлов с сервера резервного копирования Linux. После того как полное восстановление данных окончится, вам необходимо запустить программу FDISK системы DOS, чтобы пометить раздел как загружаемый, а затем с помощью программы SYS записать данные в загрузочный сектор раздела.

При работе с Windows NT, 2000 или XP процесс восстановления данных будет более сложным, особенно, если на компьютере используется файловая система NTFS. В этом случае вам надо первоначально установить систему в небольшой загрузочный раздел. Содержимое этого раздела можно сохранять и восстанавливать посредством утилиты dd, работающей в системе Linux, или с помощью одного из коммерческих инструментов. В некоторых случаях целесообразно использовать способ полного восстановления данных, отличный от того, который применялся для создания резервной копии.

25. В некоторых случаях, в особенности тогда, когда минимальная конфигурация системы, пригодная для восстановления данных, не была создана, обеспечить работу клиента резервного копирования можно, только тогда, когда повторно инсталлирована базовая операционная система. Эта система используется для восстановления остальных данных, причем в этом случае можно ограничиться частичным восстановлением.

26. AMANDA отличается от других средств, рассмотренных в данной главе, тем, что в состав этого пакета входят инструменты восстановления данных, предназначенные для выполнения на стороне клиента резервного копирования. Наиболее мощным из этих инструментов является `amrecover`, который вызывает в процессе работы другие программы, например `amrestore`.

27. Если вы запустите `amrecover` на клиенте резервного копирования от имени пользователя `root`, программа отобразит приглашение для ввода команды.

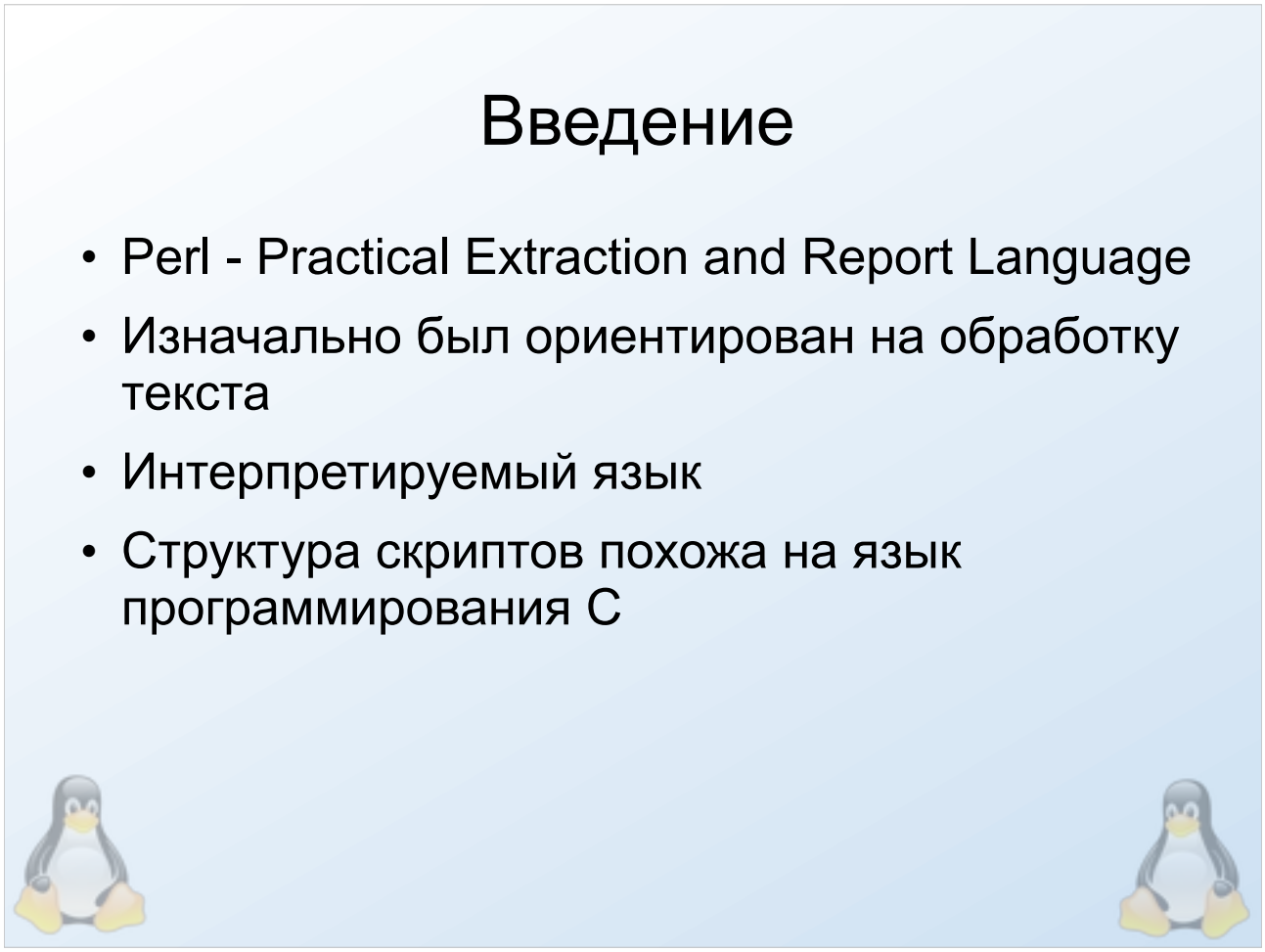
28. Допустимыми командами являются:

    1. `setdate` - определение даты резервной копии;
    2. `cd` - изменение текущего каталога;
    3. `add` - добавление файла к набору, предназначенному для восстановления;
    4. `extract` – восстановление файлов.

29. После указания команды `extract` программа `amrecover` предложит установить соответствующую ленту, содержащую резервную копию.
30. Независимо от того, какой метод вы используете для восстановления данных, этот метод необходимо опробовать до того, как его придется применять на практике. Желательно установить специальную тестовую систему и использовать ее для проверки процедуры создания резервной копии и восстановления данных.
31. Если в вашей сети имеется несколько различных операционных систем, проверьте, как выполняется восстановление данных в каждой из них.
32. Подробно опишите действия по восстановлению данных, а в случае изменения структуры сети убедитесь в том, что данная процедура по-прежнему дает ожидаемые результаты.
33. Если вы собираетесь использовать для полного восстановления данных минимальную конфигурацию системы, создайте несколько копий диска.

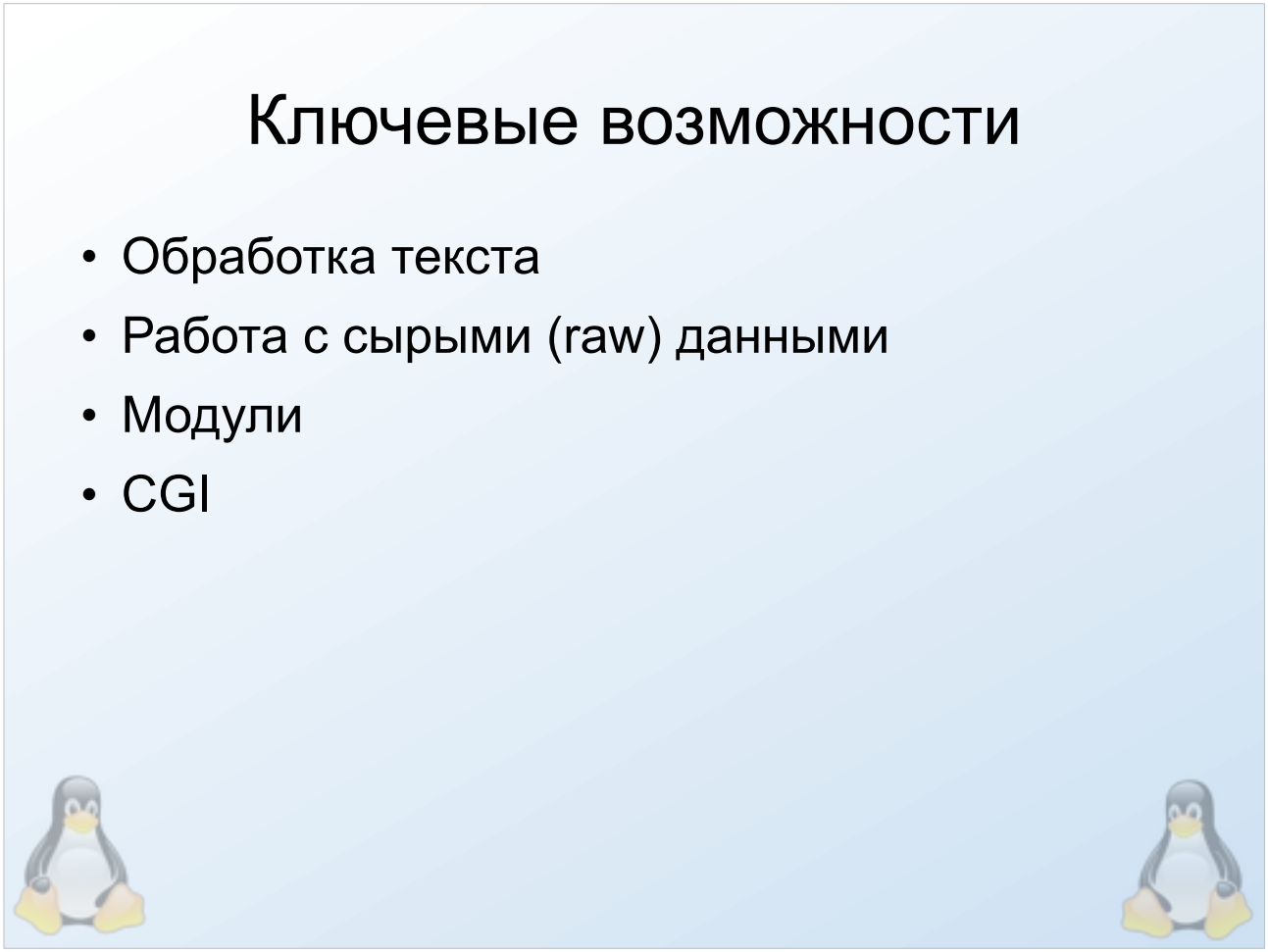
Глава 7. Язык сценариев Perl.

7.1. Введение



1. Perl (Practical Extraction and Report Language) является переносимым, интерпретируемым языком, идеально приспособленным для многочисленных приложений по обработке текста.
2. Отцом языка Perl является Ларри Вол (Larry Wall). Он разработан в 1986 году для создания отчетов о содержании многочисленных текстовых файлов в среде UNIX. Поскольку существующие средства не подходили для решения такой задачи, Вол (Wall) изобрел новое средство для ее решения. Название Perl обозначает практический язык для извлечения и составления отчетов (Practical Extraction a Report Language). Вол продолжал добавлять различные возможности к Perl и сделал его доступным для общего пользования.
3. Perl является интерпретируемым языком. Это означает, что программы, написанные на Perl, обычно исполняются путем вызова интерпретатора Perl и передачи ему списка команд, из которых состоит программа.
4. Структура Perl очень напоминает структуру языка программирования C и на первый взгляд выглядит так же, как программа C.
5. Все операторы C представлены в Perl, а управляющие структуры, такие как if или for, имеются на языке Perl в несколько измененном виде.

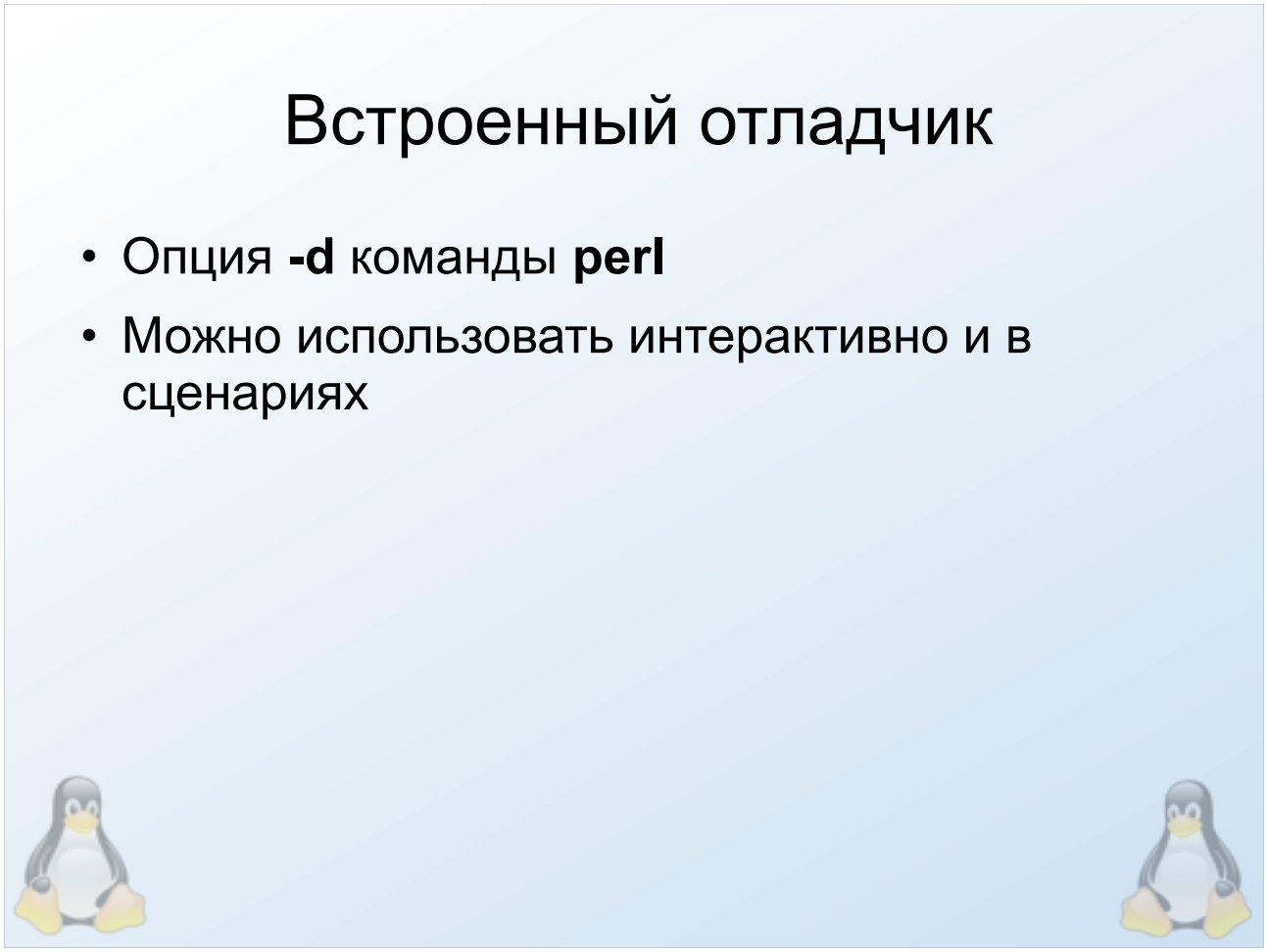
6. Perl предоставляет разработчику широкий спектр возможностей для создания кратких и эффективных программ. Ниже приведены некоторые черты Perl.



1. Ассоциативные массивы, которые индексируются программами с использованием нецелых ключей
2. Автоматическое преобразование типов между целыми числами, числами с плавающей точкой и строками
3. Автоматическое преобразование размера массивов
4. Функции для преобразования бинарных данных
5. Широкая поддержка регулярных выражений, которые программы используют для поиска, замены и других операций, связанных с разбором текста
6. Функции вывода/ввода файлов
7. Функции форматированного вывода, наподобие функции C, с добавлением к ним способности генерации отчетов на основе шаблонов (template)
8. Полный набор. операторов C, с добавлением также операций по сравнению строк
9. Функции для обработки списков, которые поддерживают стеки, очереди и другие данные списочных типов
10. Функции системного сервиса
11. Богатый набор операторов и структур управления, включая подпрограммы

12. Огромное количество модулей, которые позволяют решать самые разнообразные задачи.
13. Специальные конструкции языка и модули для создания CGI скриптов.

7.2. Использование отладчика Perl



1. Использование отладчика может оказаться исключительно полезным при изучении языка Perl.
2. Отладчик Perl встроен в сам Perl.
3. Отладчик, можно запустить, используя ключ `-d` в командной строке следующим образом:

Пример:

```
$ perl -d hello.pl
```

Примечание: В таком случае загрузит скрипт hello.pl и начнет отладку.

4. Если вы используете Perl-скрипт, то нужно поместить комментарий `#!/usr/bin/perl -d` в самом начале скрипта Perl.
5. Для загрузки отладчика без загрузки скрипта используйте следующую команду:

```
$ perl -de 0
```

Примечание: В этом случае аргумент командной строки -d указывает Perl на необходимость вызова отладчика, а аргумент -e 0 заставляет выполнить Perl скрипт, состоящий из 0 строк. Поскольку скрипт 0 не существует,то Perl просто запустит отладчик. Если Perl установлен на вашу систему корректно, то на экране вы увидите следующие данные, которые говорят, что вы находитесь в отладчике:

```
Loading DB routines from $RCSfile: perldb.pl,v $$Revision: 4.0.1.3
$$Date: 92/06/08 13:43:57 $
Emacs support available.
Enter h for help.
main '(p1000159:1):
DB<1>
```

6. В отладчике можно набрать любое выражение Perl, и он немедленно его исполнит. Дополнительно можно использовать следующие команды отладчика:
   1. `h` - распечатать в качестве подсказки список команд отладчика
   2. `n` - выполнять до следующего выражения
   3. <CR> - повторить последнюю команду `n` или `s`
   4. `p` выражение - сокращение для команды print выражение
   5. `q` - окончить работу
   6. `r` - исполнять до выхода из подпрограммы
   7. `s` - один шаг по скрипту (со входом в подпрограмму)

Пример: следующая команда отладчика использует функцию `print` для того, чтобы вывести на экран сообщение Hello World:

```
DB<1> print "Hello World\n"; <ENTER>
Hello World
DB<2>
```

7. Для упрощения набора в отладчике можно опускать точку с запятой в конце выражения.
8. Отладчик всегда переходит на новую строку для новой команды.

Пример: Следующий код иллюстрирует использование команды p: <ENTER>

```
DB<2> p "Hello World\n" <ENTER>
Hello World
DB<3>
```

Примечание: Как можно видеть, всякий раз, когда вы набираете команду отладчика, счетчик отладчика увеличивается на единицу, что находит отражение в его строке DB. Читая дальше эту главу, вам, вероятно,стоит запустить отладчик, чтобы иметь возможность набирать в нем примеры и работать с языком Perl интерактивно.

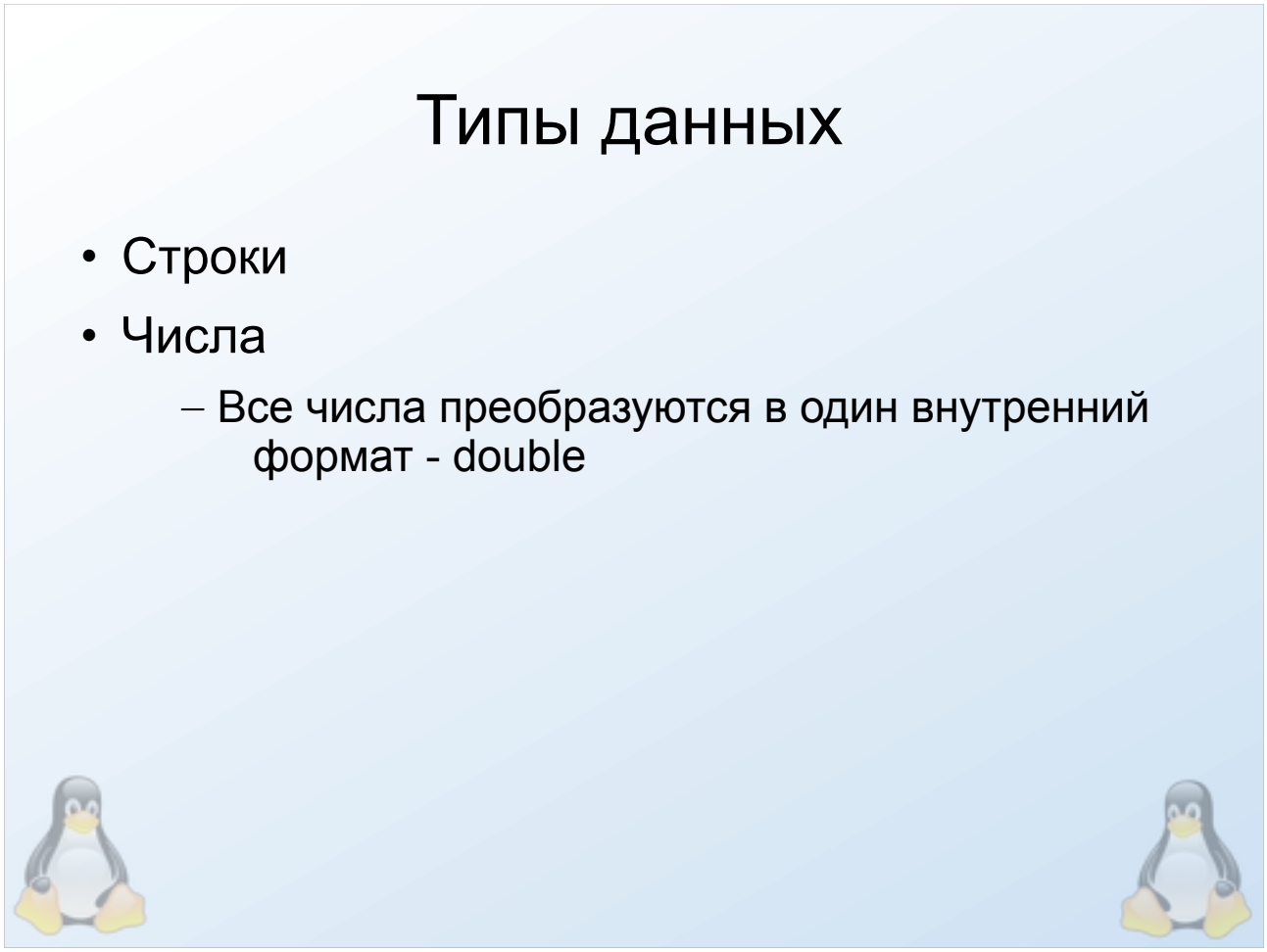
9. Для того, чтобы ввести выражение из многих строк в отладчик (debugger), необходимо использовать символ продолжения<\> конце каждой строки.. Иначе отладчик сообщит о синтаксической ошибке.

Пример: Следующая команда иллюстрирует использование символа продолжения при работе с отладчиком:

```
DB<3> for ($i = 0; $i < 10; $i++) { \ <ENTER>
cont: print $i; \ <ENTER>
cont: } <EIITER>
0123456789
DB<4>
```

Примечание: В этом случае для вывода чисел от 0 до 9 использовался цикл for, который будет подробно рассмотрен в этой главе.

7.3. Типы данных в Perl



1. Типы данных используются в программах при объявлении переменных.
2. Тип данных определяет то множество значений, которые может принимать переменная, а также набор операций,которые программа может выполнять с ней.
3. В языке Perl данные могут быть числом или строкой символов.
4. Одно значение называется скалярной величиной или просто скаляром.

Пример: скалярных значений, которые используются в языке Perl:
- Десятичные: 127 или 127.0 или 1.27E2
- Шестнадцатиричные: 0x7F или 0x7f
- Восьмеричные: 0177 (первый 0 указывает, что используется восьмеричное число)
- Строка: "Hello World\n" или 'Hello World'

Пример: следующая команда использует отладчик Perl для того,чтобы вывести число 0177 восьмеричной системы, соответствующее числу 127 десятичной:

```
DB<4> p 0177 <ENTER>
127
```

5. Perl переводит данные в свой внутренний формат. Когда Perl печатает восьмеричные или шестнадцатиричные значения, он сначала переводит их в десятичный формат, как было показано.

Примечание: Скрипт Perl позволяет использовать функцию printf для того, чтобы выводить, значения в вызываемом формате, в таком как восьмеричный или шестнадцатиричный.

6. В качестве внутреннего представления всех чисел используется формат с плавающей запятой двойной точности (double). Иными словами, среди внутренних форматов нет целочисленного.

Примечание: В большинстве случаев вы можно не обращать на это внимания, и Perl сделает все сам как надо. Например, если вы используете величины в контексте, где только целочисленные значения имеют смысл, Perl сам автоматически усечет число.

Примечание: Если вы программируете на C и использовали целочисленное деление с усечением целых чисел автоматически, то, программируя на языке Perl, надо не забыть выполнить усечение вручную, используя функцию int().

Пример: Следующая команда иллюстрирует, как Perl обрабатывает числа целого типа и с плавающей запятой:

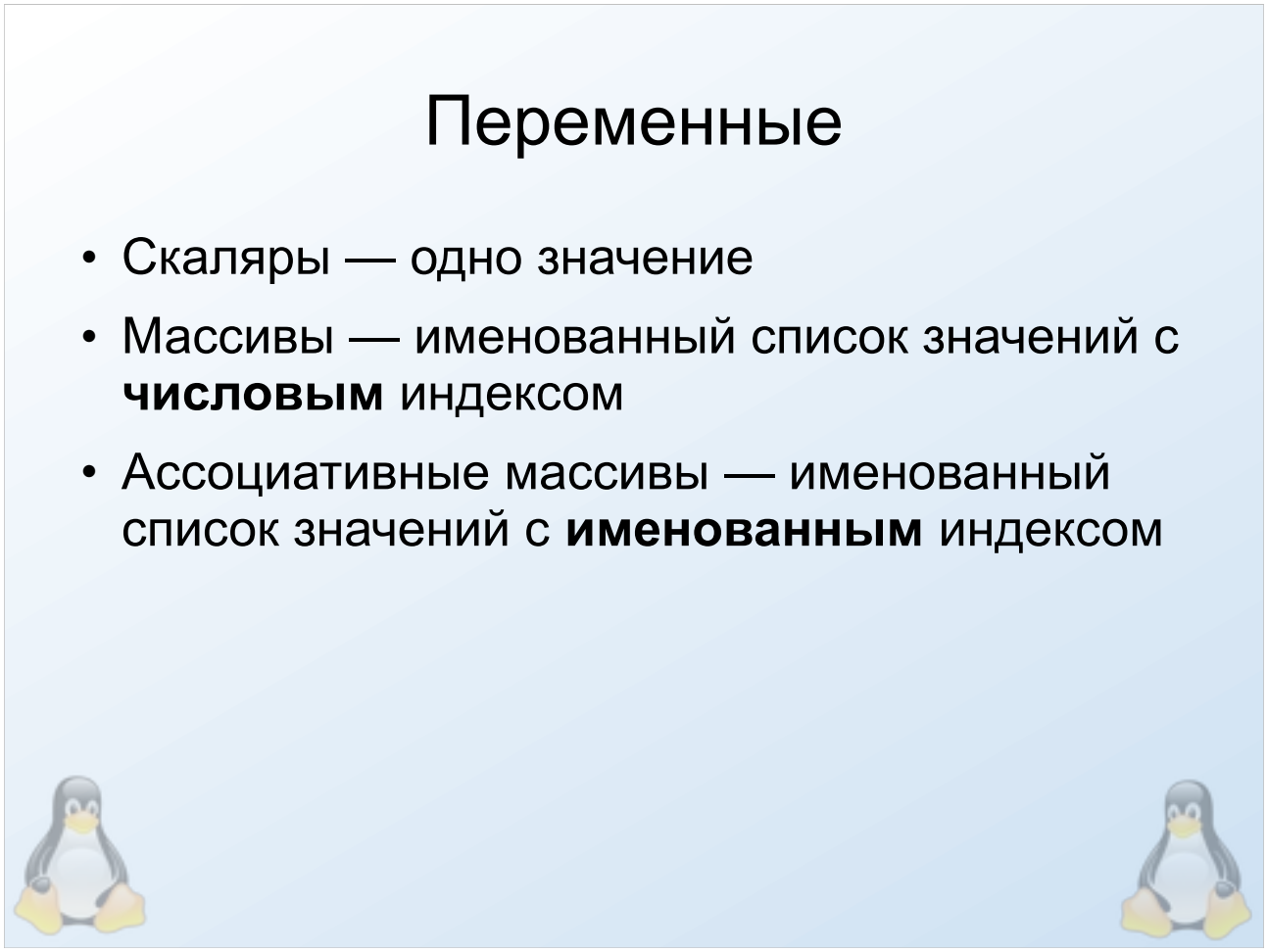
```
print 6 & 3;      # выведет 2
print 6.9 & 3.1    #
print 7 / 2      # выведет 2.3333    не целое
print int(7/3)     # выведет 2
```

7. Точно так же, как Perl преобразует числа с плавающей запятой в целые числа: когда скрипт использует целочисленные значения, он также преобразует числа в строки и наоборот, когда такое преобразование имеет смысл.

Пример: если скрипт использует числа в контексте, где только строки имеют смысл, например, при соединении строк, он конвертирует числа в строки. Аналогичным образом, если требуется использовать строки там, где только числа имеют смысл, то Perl конвертирует их в числа. Работая со скриптами Perl, обычно не надо беспокоиться о внутреннем представлении скалярных величин.

8. Perl поддерживает также концепцию булевых значений, но не имеет для их описания специального типа. Как и в C, численное значение рассматривается истинным, если оно не равно нулю. Дополнительно строковое значение рассматривается как истинное, если оно не равно '' или '0'. Некоторые булевы операторы, возвращают единицу в качестве значения<истинно> и нуль - в качестве <ложно>.
9. Тем самым, ваш скрипт должен просто рассматривать ненулевые величины как строчного типа,так и числового в качестве булева значения <истинно>.
10. Скрипты Perl могут группировать скалярные величины вместе и создавать список (list).
11. Если скрипт хранит список в какой-то переменной,то эта переменная становится массивом (array).

7.4. Переменные



1. Perl поддерживает три типа переменных:
   1. скаляры,
   2. массивы;
   3. ассоциативные массивы.
2. Как и в языке C, имена переменных пишутся с различением строчных и заглавных букв.

Пример: имена VAR, Var и var описывают различные переменные.

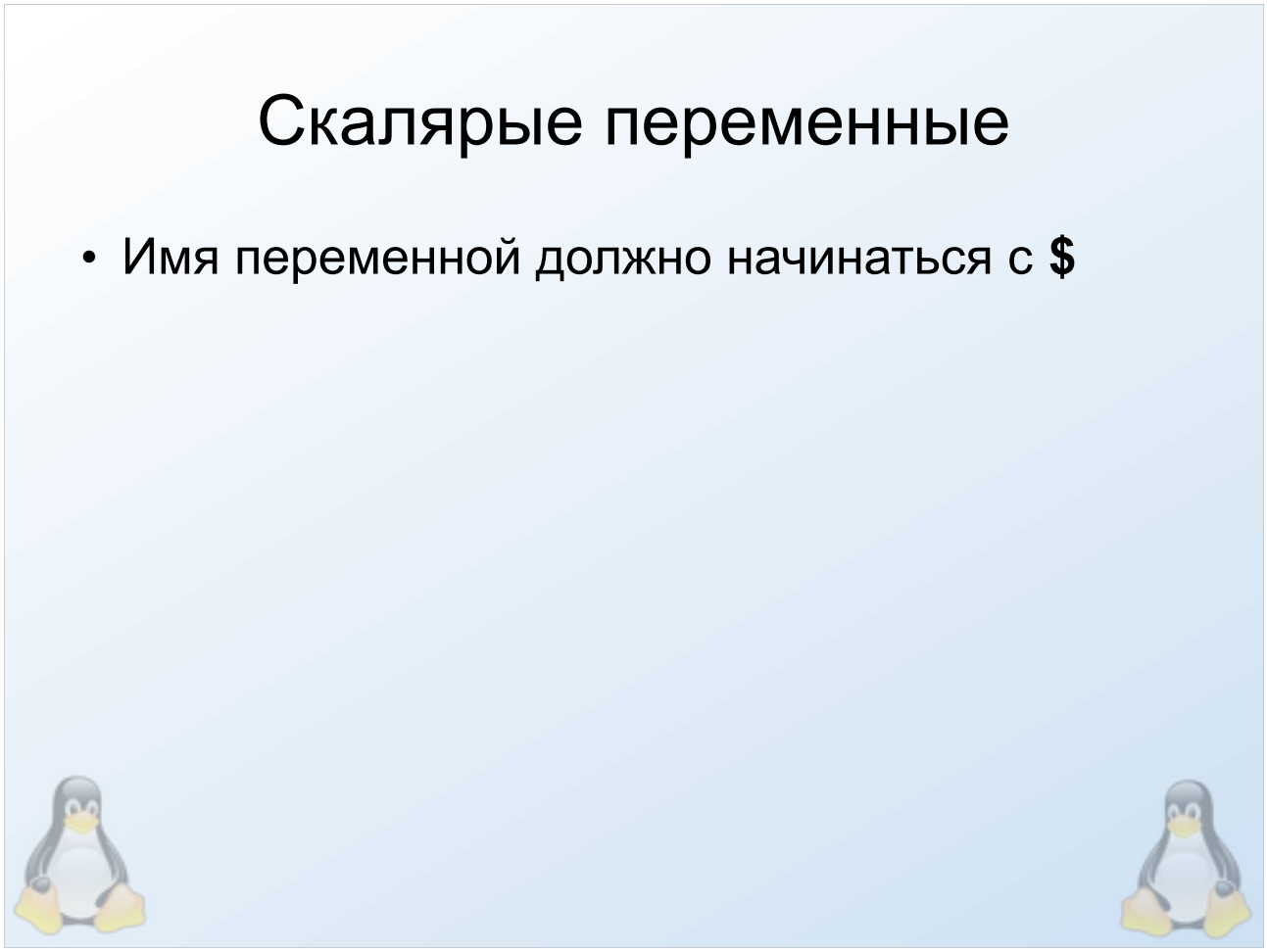
3. Скрипт может иметь скалярную переменную под именем var и переменную-массив, также названную var. Они будут различаться в языке Perl в соответствии с контекстом.

Примечание: Переменные Perl не типизированы, как это делается и в C.

Пример: скалярная переменная может содержать любой тип скаляра, и приведение типов осуществляется автоматически.

4. Переменные на языке Perl необязательно должны быть объявлены. Если переменная не объявлена, то Perl рассматривает ее как глобальную.

7.4.1. Скалярные переменные

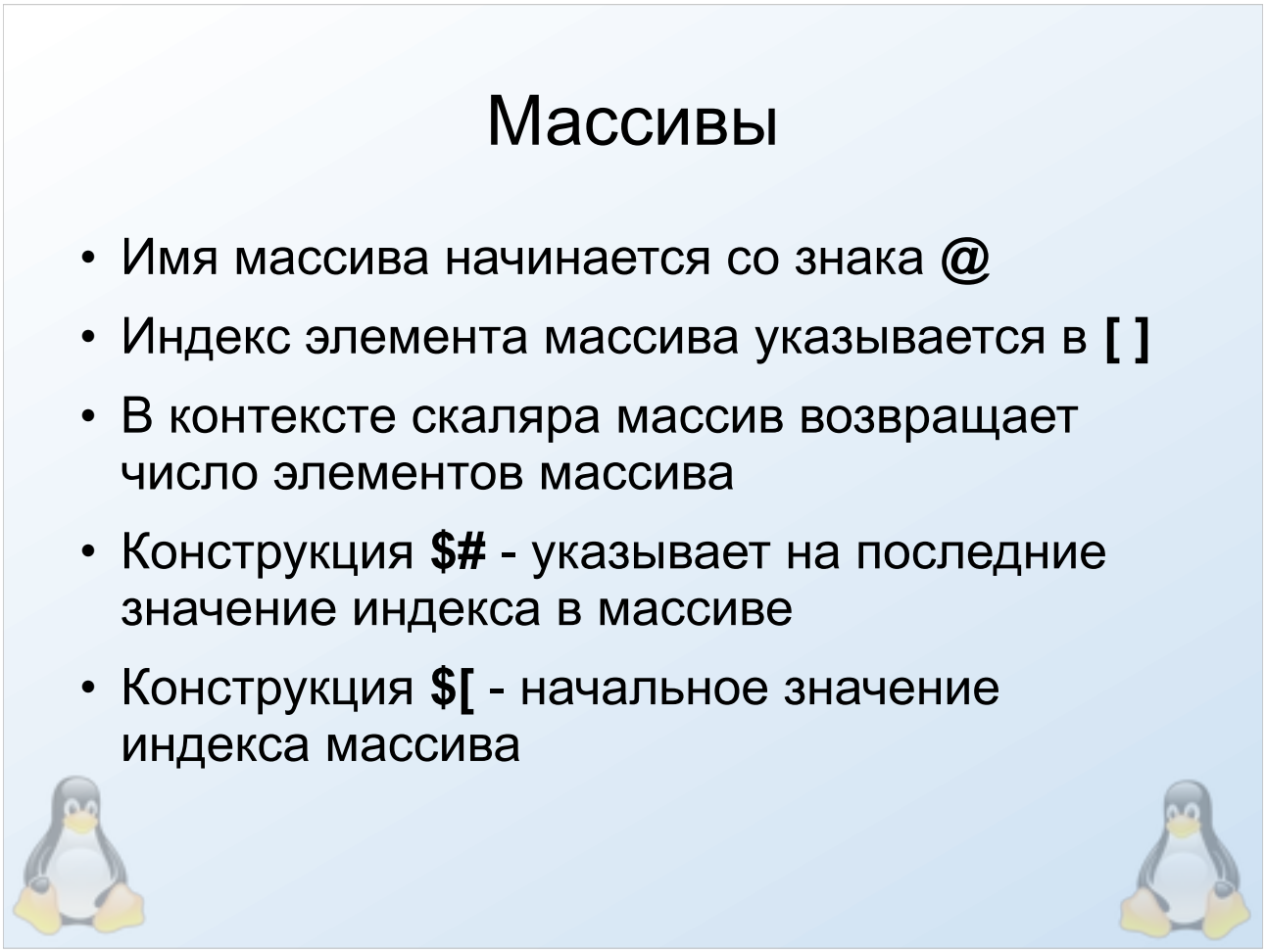


1. Скалярная переменная может содержать единственное значение.
2. В языке Perl имена скалярных переменных всегда начинаются со знака ($).

Пример: В следующем выражении скалярной переменной $age присваивается значение 35, а переменной $name строковое значение <Bob>. Затем используется функция print для вывода значения каждой из переменных:

```
$age = 35;
$name = 'Bob';
print ($name,'is',$age);
```

7.4.2. Массивы



1. Массивы представляют собой переменные, принимающие в качестве значения список из скалярных величин.

Пример: Следующий текст программы на языке Perl иллюстрирует объявление переменных типа массив и их инициализацию:

```
@days = ('Sun','Mon','Tue','Wed','Thu','Fri','Sat');
print(@days);              # выведет 'SunMonTueWedThuFriSat'
print($days[4]);           # выведет 'Thu'
@weekdays = @days[1..5];   # значение ('Mon','Tue','Wed','Thu','Fri')
@emptylist = ();           # постой список
```

2. Ссылка на переменные типа массив начинается со знака @ и сопровождается значениями в квадратных скобках [ ].
3. Индексами массивов для скриптов всегда являются переменные целого типа, которые начинаются с нулевого значения.
4. Если использовать знак ($) вместо знака (@), то скрипт будет ссылаться на скалярную величину.

Примечание: Это замечание является очень важным. Квадратные скобки указывают, что скрипт ссылается на массив.Знак $, в свою очередь, означает ссылку на скалярную величину.

5. Массиву могут присваиваться в качестве значений литералы, скрипт может также присваивать массивам значения переменных или даже других массивов, как показано ниже:

```
@stuff = ($age, $name)
@FriendsOfMine = ('Joe','Mary', @FriendsOfYours);
```

Пример: использование части массивов:

```
@weekend = @days[0,6] ;    # результат    ('Sun','Sat')
print (@days[1..5,0,6]);
# выведет 'MonTueWedThuFriSunSat'
```

6. Если скрипт использует переменную типа массив в контексте скаляра, то значением служит число элементов массива.
7. Скалярным контекстом является такой контекст, где только скалярные значения имеют смысл.

Пример: следующее выражение использует скалярный контекст для массива stuff для того, чтобы определить число элементов содержащееся в массив. Если число элементов, больше или равно 2, то скрипт выдает сообщение и заканчивает исполнение:

```
(@stuff >= 2) || die "Too much stuff! \n";
```

Примечание: Функция die служит директивой языку Perl закончить выполнение и выдать при этом указанное сообщение. Если сообщение не содержится, то функция просто заканчивает выполнение скрипта.

8. Perl также поддерживает специальную конструкцию переменная, которая возвращает последнее значение индекса в массиве.

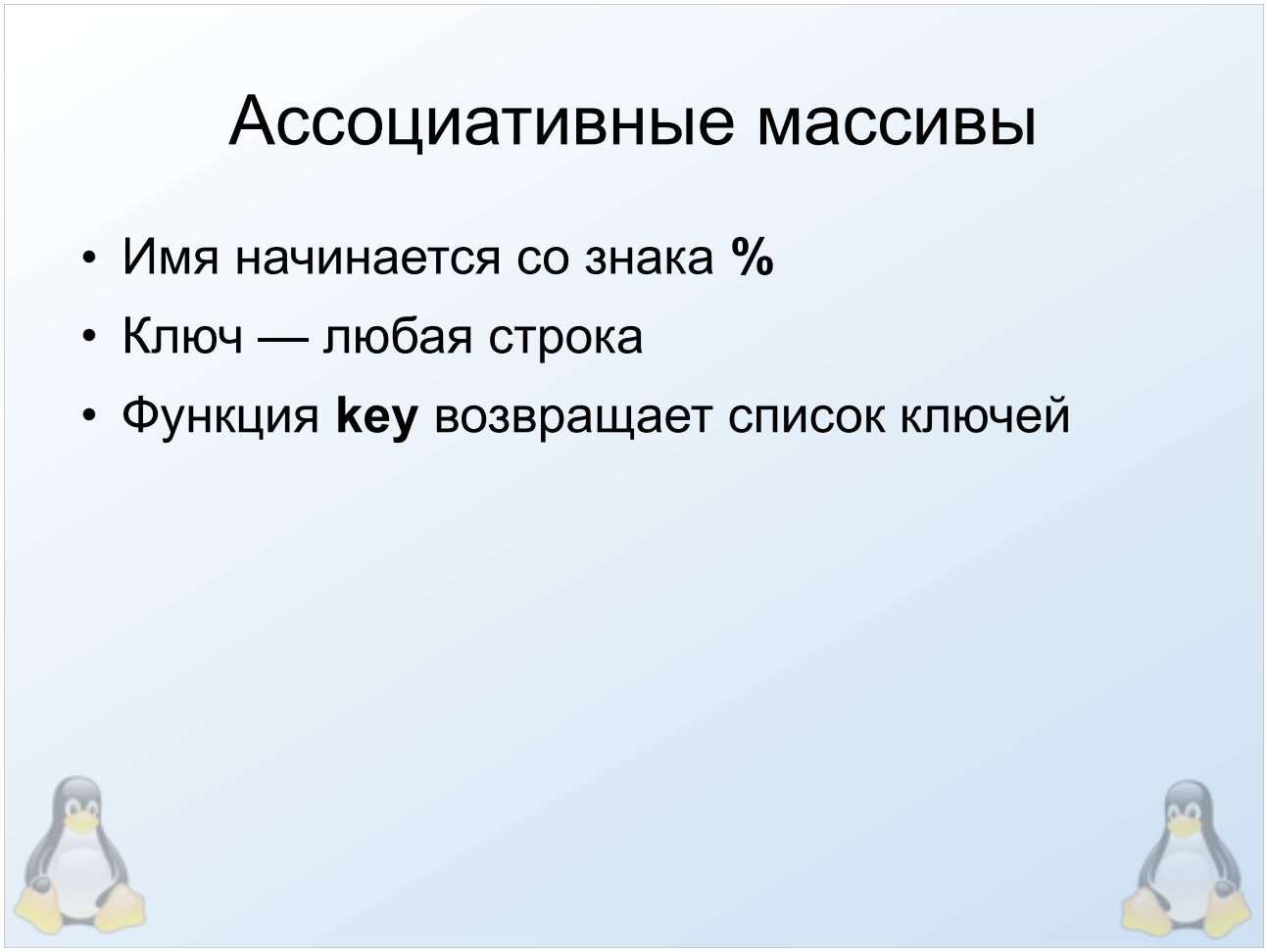
Пример: следующее выражение for использует $[для того, чтобы определить начальное значение индекса массива, и $# для определения последнего из элементов массива. При этом с помощью выражения for выводятся значения каждого из элементов:

```
for ($i =$[; $i <= $#stuff; $i++)
    {
        print $stuff[$i];
    }
```

Примечание: Записанный цикл for можно заменить следующим эквивалентным выражением:

```
Print @stuff;
```

7.4.3. Ассоциативные массивы



1. Ассоциативные массивы аналогичны обычным массивам в том отношении, что они представляют собой список скалярных переменных.
2. Различие заключается в том, что массив должен использовать целочисленные значения в качестве индексов при выборе элементов массива, тогда как ассоциативный массив может использовать величины любого типа для выбора элементов массива.
3. Индексные величины для ассоциативного массива называются ключами.

Пример:

```
$ages{'Bob'} = 35;
$ages{'Mary'} = 25;
$, = ' ';      # change output    separator for print operator
print @ages{'Bob','Mary'};   #    выведет '25 35'
print keys(%ages);           #    выведет 'Bob Mary'
for $name (keys(%ages))
    {
        print "$name is    $ages{$name}\n";
    }
```

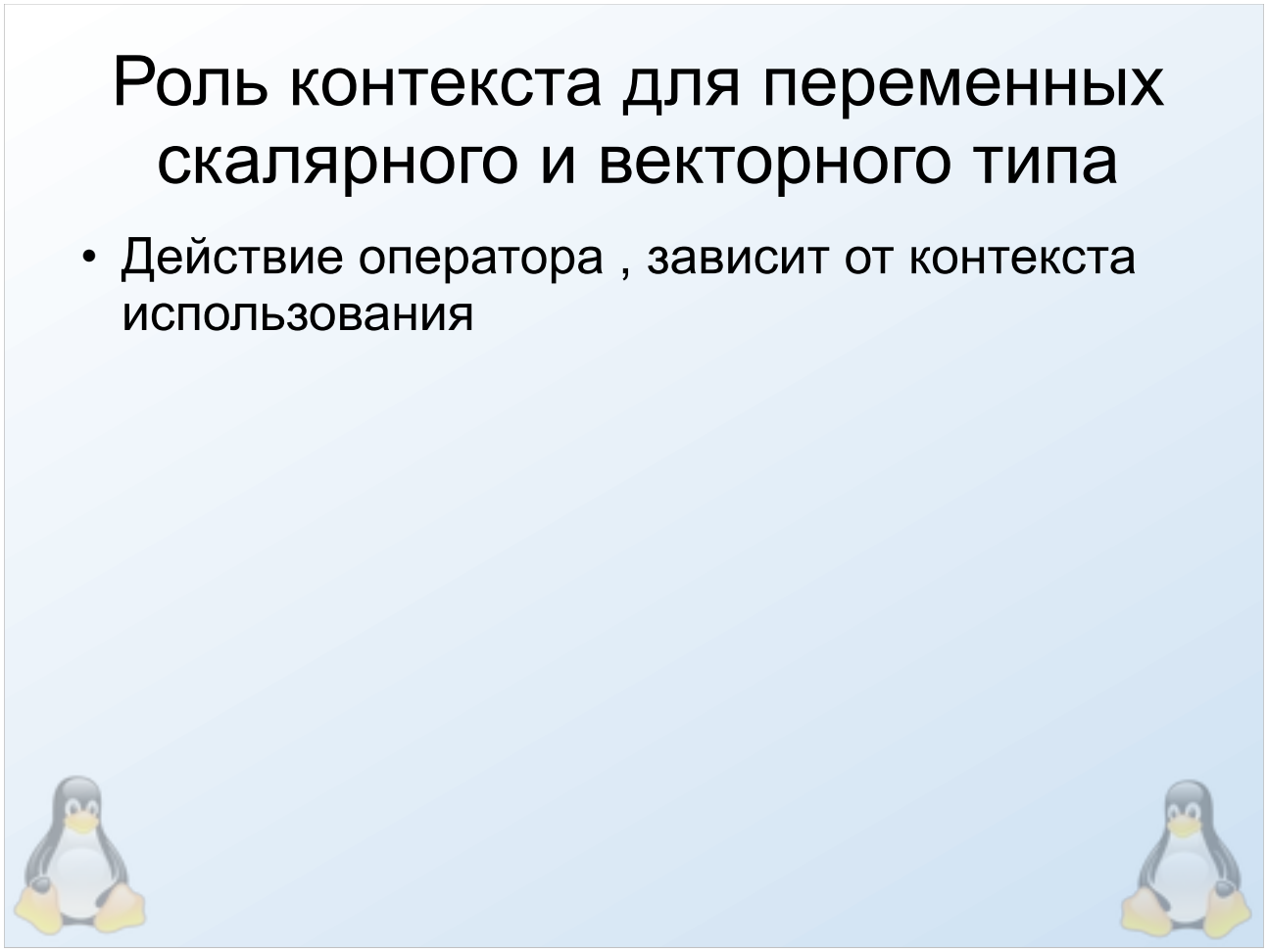
Примечание: Как можно видеть, программа присваивает значения переменной'$,' (скалярная переменная, именем которой является запятая). Скрипт использует это выражение для того, чтобы при дальнейшем использовании оператора print выходные данные не сливались между собой. Далее в этой главе обсуждаются специальные переменные, такие как '$,'. Ассоциативные массивы идентифицируются с помощью фигурных скобок. Так же как с массивами, при ссылках на ассоциативный массив целиком индексы не используются. Например, ссылка @ages{'Bob', 'Mary'} использует индексы в скобках, что указывает на ассоциативный массив. Префикс @ указывает на то, что речь идет о массиве. Аналогичное использование знака доллара перед массивом указывает, что используется скалярная величина.

Примечание: Если два ключа заданы, то вместе со знаком @ эта говорит о том, что речь идет о части ассоциативного массива и результат должен быть в виде списка. Такое выражение эквивалентно #ages{' Bob'}, #ages{'Mary'}. которое имеет своим значением величину (35, 25).

4. Оператор `keys` возвращает полный список ключей ассоциативного массива.
5. В приведенном выше примере цикл for ссылается на переменные,заключенные в двойные скобки. Perl, в свою очередь, заменит значения, на которые ссылаются переменные, в то время, когда будет анализировать строку. Этот называют процесс подстановкой переменной или интерполяцией.

Примечание: Perl не интерпретирует переменные, содержащие строки в одинарных кавычках.

7.4.4. Роль контекста для переменных скалярного и векторного типа



1. Отметим, что оператор построения списка (,) выглядит точно так же, как оператор (,) последовательного вычисления (sequential evaluation).
2. Какой из операторов используется, зависит от контекста, в котором он появляется, в частности, является ли переменная скаляром или массивом.
3. Perl использует конструирование списков в контексте массивов и последовательное вычисление для скаляров.

Пример: Рассмотрим следующие выражения:

```
@an_array = (1,2,3,4,5);
$a_scalar = (1,2,3,4,5);
```

Примечание: В первом выражении инициализируется массив, в то время как второе выражение устанавливает значение скалярной переменной $a_scalar равным 5, отбрасывая первые четыре величины.

Пример: Рассмотрим два следующих выражения:

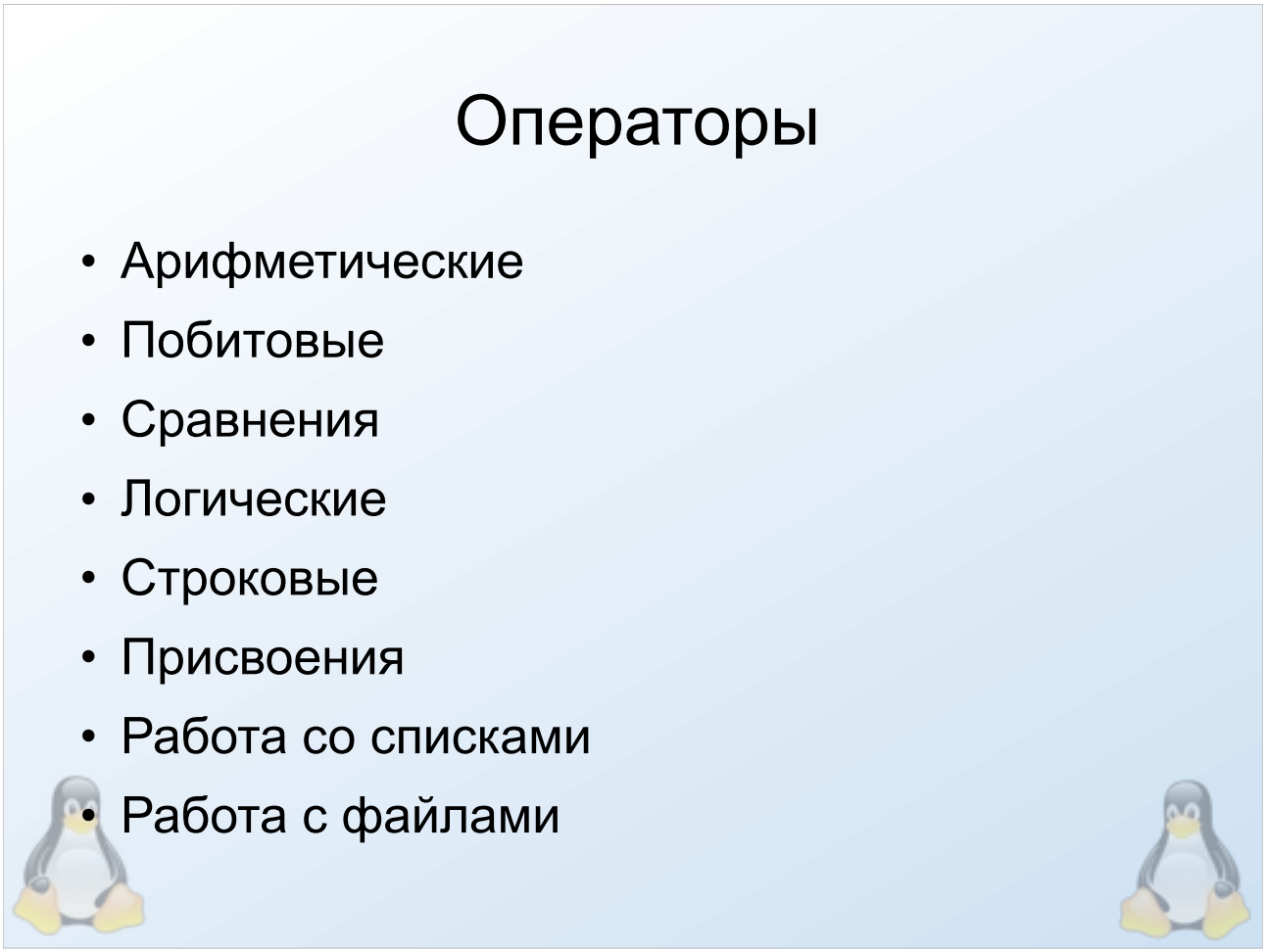
```
print $assoc{1,2};
print @assoc{1,2};
```

Примечание: В первом случае будет напечатано одно значение ассоциативного массива с двумя ключами, в то время как во втором будут напечатаны два значения ассоциативного массива с одним ключом.

Пример: Из двух следующих выражений первое копирует список, тогда как второе присваивает скалярной величине значение, равное размеру массива:

```
@x = @list;
$x = @list;
```

7.5. Операторы Perl



1. Все операторы C присутствуют в языке Perl, за исключением оператора приведения типов (type), оператора обращения к содержимому указателя *ptr и оператора выбора члена структуры var.member или var->member.
2. Кроме того, в языке Perl реализовано много новых операторов для использования в таких операциях как сравнение и обработка строк.
3. Имеются следующие виды операторов:
    1. Арифметические операторы
        1. + сложение
        2. - вычитание или изменение знака
        3. * умножение
        4. / деление (только для чисел с плавающей запятой)
        5. % взятие по модулю (только для целочисленных значений)
    2. Побитовые операторы
        1. | побитовое ИЛИ
        2. & побитовое И

3. ^ побитовое исключающее ИЛИ
4. ~ побитовая инверсия
5. << сдвиг влево
6. >> сдвиг вправо

3. Операторы сравнения

Примечание: В скобках указаны строковые операторы.

1. = = (eq) равно
2. != (ne) не равно
3. > (gt) больше чем
4. < (lt) меньше чем
5. >= (ge) больше или равно
6. <= (le) меньше или равно
7. <=> (cmp) не равно (результат со знаком)

Примечание: Результатом операции сравнения является единица, если сравнение истинно и нуль в противном случае. Однако последняя операция (<=> или cmp) может возвращать значения -1, 0 или 1 в зависимости от того, является ли значение первого операнда меньше, второго, равным ему или большим.

4. Логические операторы
   1. || логическое ИЛИ
   2. && логическое И
   3. ! логическое отрицание
   4. -: условная операция
   5. , последовательное выполнение
5. Строковые операторы
   1. . конкатенация строк
   2. x репликация
   3. =~ сопоставление переменной с образцом
   4. !~ то же, что и предыдущее, но с дополненным отрицанием результата
6. Операторы присваивания
   1. =
   2. +=
   3. -=
   4. *=
   5. /=
   6. %=
   7. |=
   8. &=

9. ^=
10. ~=
11. <<=
12. >>=
13. **=
14. .=
15. x=

7. Операции для работы со списками
   1. , конструктор списков
   2. .. оператор области
   3. x оператор репликации
8. Операторы для работы с файлами
   1. -d проверяет наличие каталога
   2. -e определяет наличие файла
   3. -s определяет размер файла
   4. -w определяет, можно ли писать в данный файл

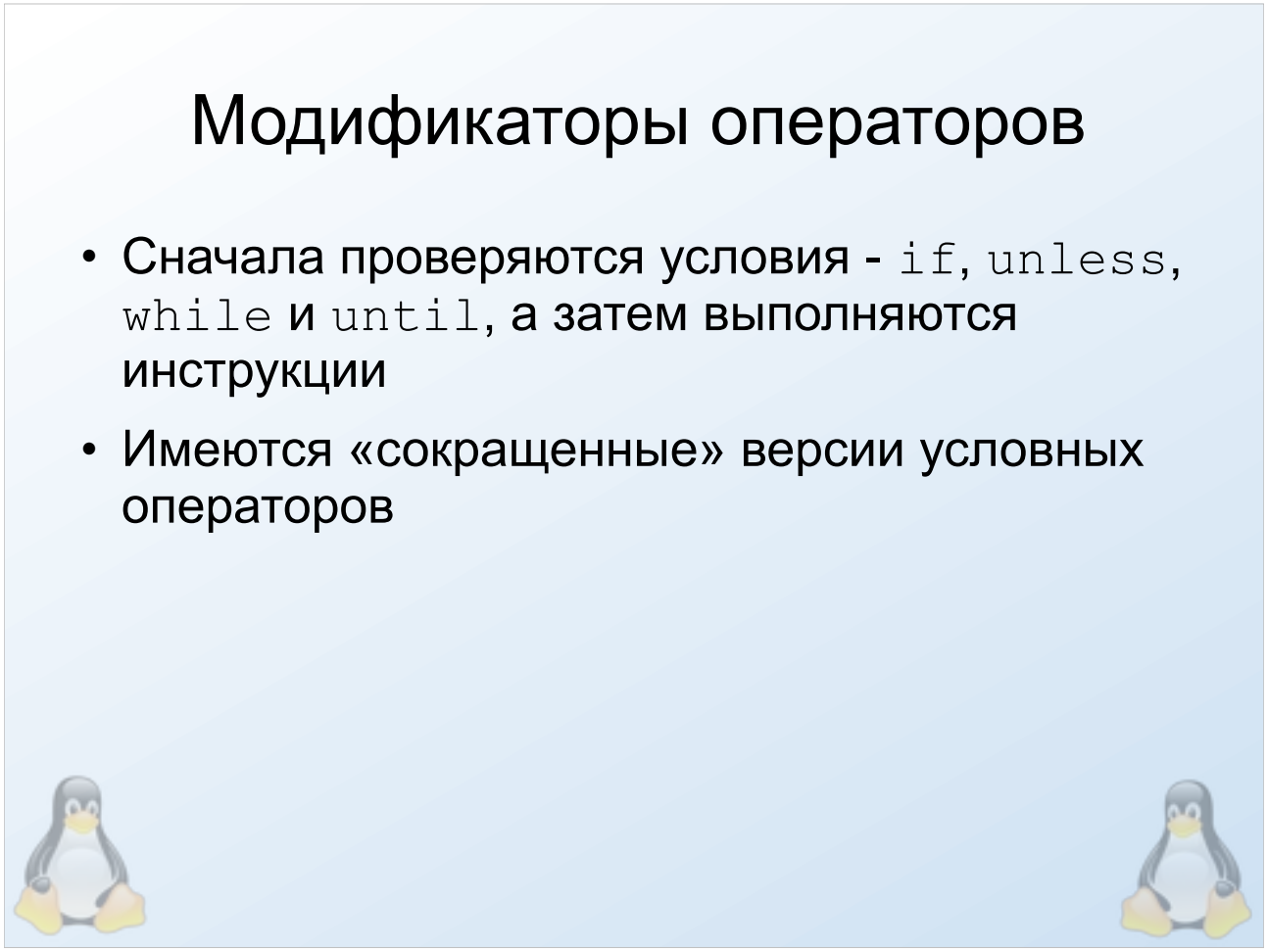
7.5.1. Приоритеты выполнения операторов

1. Как и всякий язык программирования, Perl определяет приоритеты выполнения операторов, с помощью которых упорядочивается последовательность их выполнения.
2. Далее перечислены приоритеты операторов начиная от высшего и следуя к низшему:

```
++
! ~ унарный минус
**
=~ !~
* / % x
+
<<>>
-d -e -s -w (и другие файловые операторы)
<> <= >= It gt le ge
= = != < => eq ne cmp
&
|^
&&
||
..
: = += -= *=
```

3. В скрипте можно изменять последовательность выполнения операторов с помощью скобок.

7.6. Модификаторы операторов



1. В языке Perl используются специальные формы конструкции `if`, `unless`, `while` и `until`, которые позволяют управлять ходом вычислений.
2. Perl сначала анализирует условие и только потом выполняет инструкцию.
3. В определенных случаях эти специальные конструкции могут сделать ваш код более ясным и легко читаемым. Для того, чтобы код было легче читать и его смысл был более очевиден, целесообразно выбирать подходящий формат для записи.

Пример: Рассмотрим следущее выражение, использующее функцию `die` для того, чтобы закончить выполнение скрипта, если значение переменной `$count` меньше чем 10:

```
if ($count < 10)
    {
        die;
    }
```

Примечание: Если расположить функцию die перед оператором if, как показано ниже, число строк кода уменьшится:

```
die if ($count < 10);
```

Примечание: Аналогичным образом, то же самое выражение может быть записано в следующем виде:

```
($count >= 10) || die;
```

Примечание: В данном случае, если переменная $count больше либо равна10, Perl прекращает дальнейшее вычисление в данной строке и функция die не выполняется. В противном случае, если переменная $count меньше 10, после вычисления первой части выражения код запускает функцию die и тем самым заканчивает выполнение скрипта.

4. Конструкция `unless` также позволяет уменьшить число строк до одной:

Пример:

```
die unless ($count >= 10);
```

Пример: Аналогичным образом, следующие циклы `while` являются идентичными:

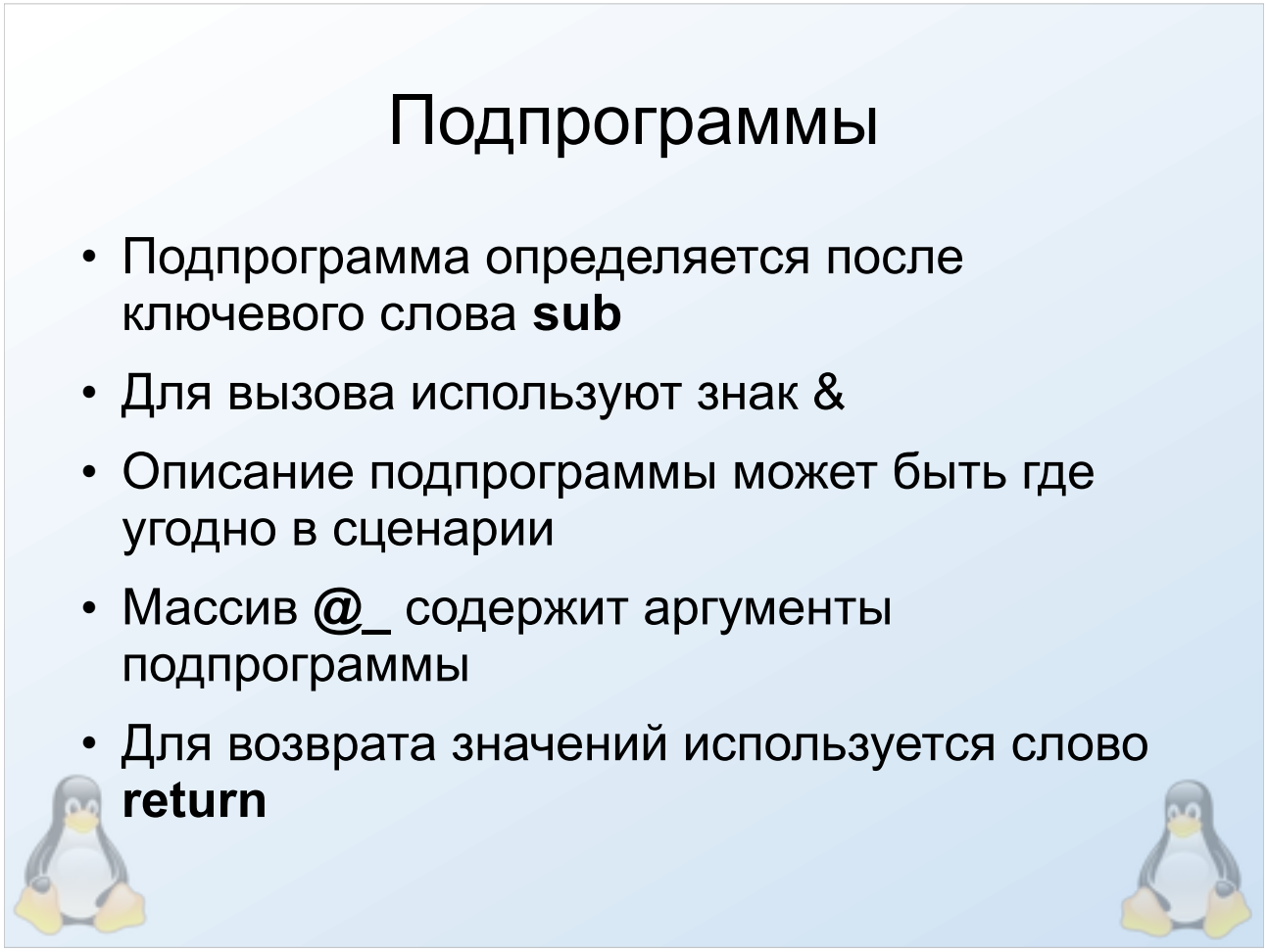
```
$i = 0;
while ($i < 10)
  {
    $i++;
   };
-----------------------------------------------------
$i = 0;
$i++ while ($i < 10);
```

Примечание: Как можно видеть, используя модификацию конструкции, скрипт позволяет сократить число строк в записи цикла до одной. Аналогичным образом, следующие циклы until эквивалентны:

```
$i = 10;
until ($i >= 10)
  {
    $i++;
   };
-----------------------------------------------------
$i = 10;
$i++ until ($i >=10);
```

Примечание: Во всех четырех случаях, даже если выражение для оценивания расположено после инструкции для выполнения, Perl сначала анализирует условие и только потом выполняет инструкцию.

7.7. Подпрограммы



1. Подпрограмма может быть определена с помощью ключевого слова `sub`, как показано ниже:

```
sub demo_sub
   {
    print "demo_sub called\n";
   }
```

Примечание: В данном случае инструкции образуют подпрограмму, названную demo_sub.

2. Для вызова подпрограммы перед именем помещают знак амперсанда (`&`).

Пример:

```
&demo_sub;
```

3. При вызове подпрограммы в языке Perl скобки могут быть опущены.
4. Вы можете разместить подпрограмму где угодно в пределах исходного кода скрипта, потому что Perl проанализирует весь исходный текст перед тем, как начать выполнение скрипта. Можно объявить подпрограмму в исходном коде сразу после первого использования подпрограммы (forward reference).
5. Подпрограммы могут иметь аргументы и возвращать значения.

Пример: Следующий фрагмент кода содержит подпрограмму с именем show_value, которая выводит значение, полученное подпрограммой в качестве параметра:

```
sub show_value
  {
    print 'The value    id ', $_[0];
  }
&show_value(1001);
```

6. Формально подпрограмма языка Perl не объявляет переменных для хранения аргументов. Вместо этого в подпрограмму передается переменная типа массив с именем `@_`, которая содержит значения параметров.
7. В свою очередь, подпрограмма получает доступ к значениям аргументов, используя следующие обозначения для элементов массива: `$_[0]`, `$_[1]`, и т. д.

Пример: Аналогично предыдущему примеру, следующая подпрограмма show_two_values выводит значения двух параметров:

```
sub show_two_values
    {
        print 'Первый параметр    ', $_[0], "\n";
        print 'Второй параметр    ', $_[1], "\n";
    }
  &show_two_values(1001, 2002);
```

Пример: Наконец, следующая функция show_all_values выводит значения всех параметров, которые она получает. Функция использует массив для определения числа параметром:

```
sub show_all_values
    {
        for ($i = 0; $i < @_; $i++)
          {
               print 'Parametr ', $i, ' is ', $_[$i], "\n";
          }
        }
& show_all_values(1001,2002,3003,4004);
```

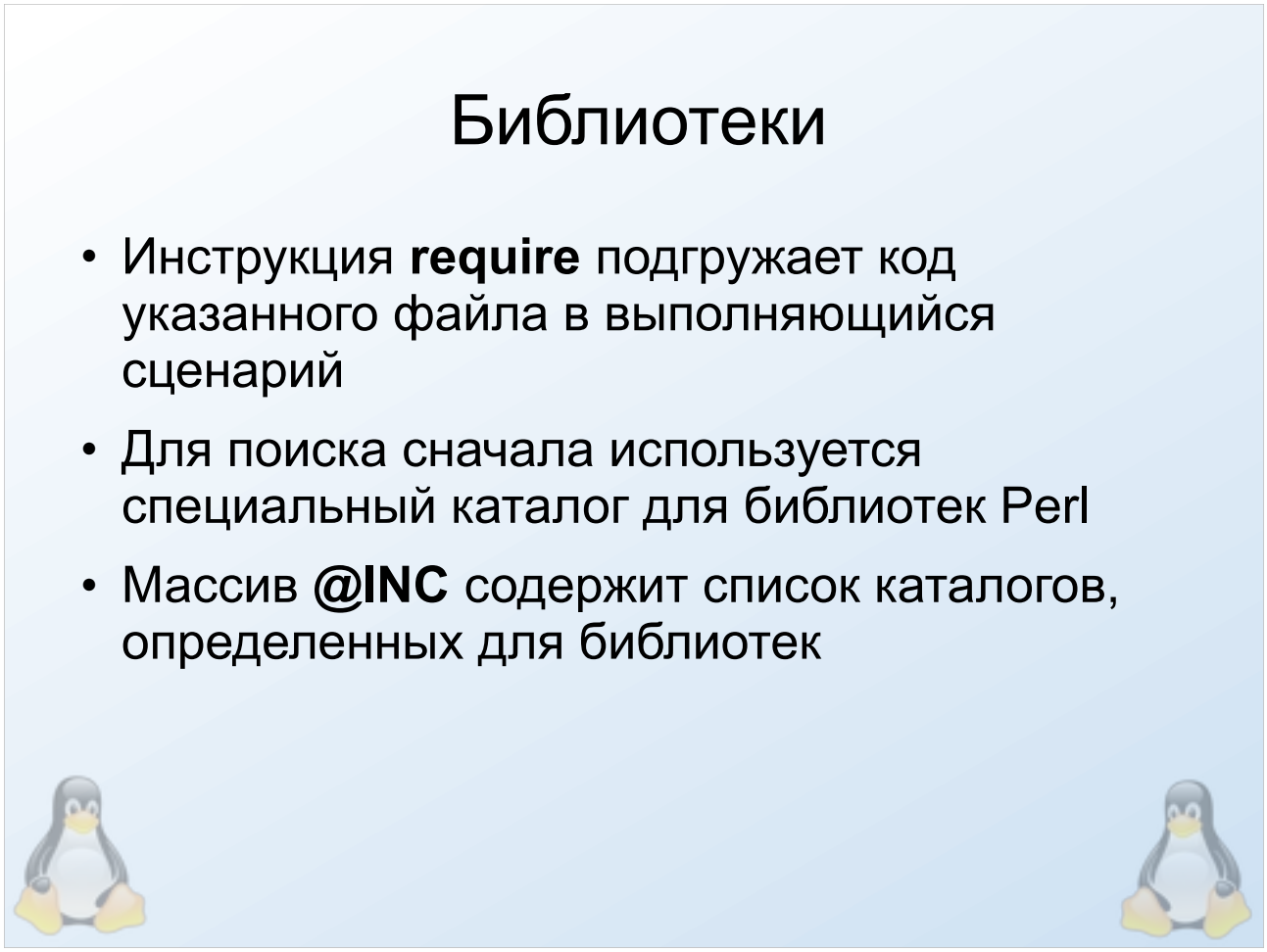
8. Подпрограммы Perl могут возвращать значения. Для этого используется инструкция return.
9. Perl не требует обязательного использования инструкции return. Если подпрограмма не содержит инструкцию return,то в качестве возвращаемого значения будет взято последнее оцененное выражение.

Пример: В следующем примере складываются два параметра и возвращается результат:

```
sub add_values
  {
    return $_[0] + $_[1];
  }
```

```
print 'The result is: ', &add_values(1001,2002);
```

7.8. Библиотека подпрограмм



1. Perl фактически не поддерживает концепцию библиотеки.

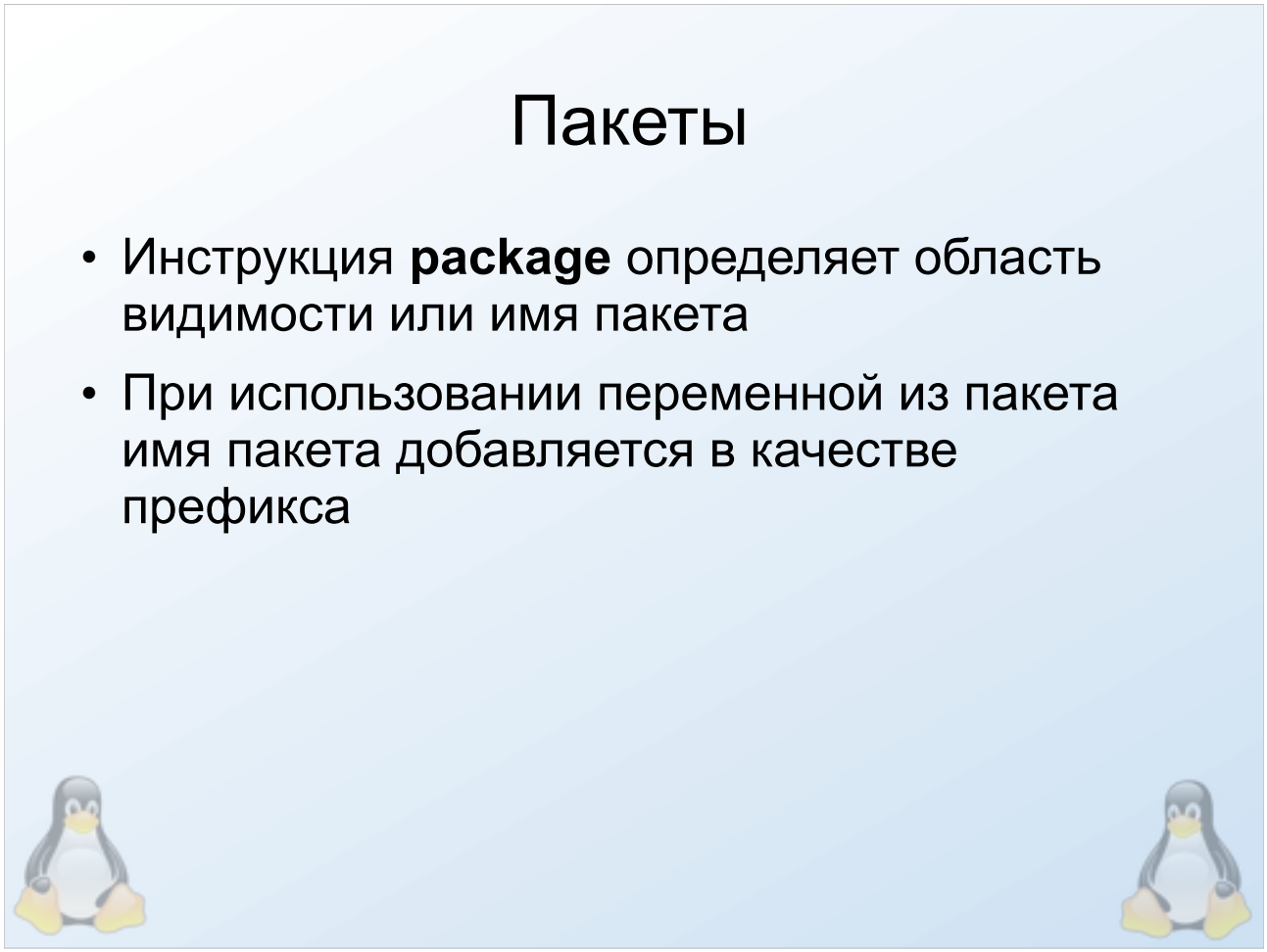
Примечание: Тем не менее, в нем имеется механизм,позволяющий скриптам использовать исходный код из другого файла.

Пример: предположим, что вы храните подпрограмму add_valuesl в файле под именем addvalue.pl. Используя инструкцию require, другой скрипт на языке Perl может получить доступ к той подпрограмме, как показано ниже:

```
require "addvalue.pl";
print &add_values(10,11);
```

2. Чтобы найти файл исходных кодов, Perl сначала ищет в каталоге, определенном по умолчанию для библиотеки языка Perl (подробности можно уточнить в инструкции по инсталляции), а затем в текущем каталоге. Можно также использовать абсолютный или относительный путь, к которому добавлено имя файла.
3. Perl запоминает, какие файлы были затребованы оператором `require`, и загружает их только один раз даже в случае многочисленных обращений к этим файлам.
4. Имеется много стандартных библиотек, которые расширяют возможности языка Perl.

7.9. Использование пакетов для изоляции подпрограмм



1. Если у вас имеется много подпрограмм, особенно подпрограмм,которые вы храните в различных файлах, то может возникнуть коллизия имен переменных, когда одно и то же имя переменной используется в различных целях.
2. Perl помогает избежать этого с помощью пакетов (`packages`).
3. Если несколько подпрограмм совместно используют какие-то специфические данные, то эти данные могут потребовать глобальной области видимости, что как раз и может вести к коллизии имен. Используя пакеты, можно группировать глобальные данные в частные пространства имен (`name-spaces`), вне пределов которых глобальные переменные не видны, т. е. неизвестны.

Пример: Рассмотрим приведенный ниже простой пример, в котором две подпрограммы (находящиеся в различных файлах) используют частные, индивидуальные пространства имен.

```
# Код в файле one.pl
sub sub_one
    {
        package demo_one;
        $some_data = 10;
```

```
    }
# * * * * * * * *
# Код в файле two.pl
sub sub_two
    {
        package demo_two;
        $some_data = 20;
    }
```

Примечание: Как можно видеть, первая подпрограмма использует имя пакета demo_one, вторая подпрограмма использует имя пакета demo_two. Обе подпрограммы могут устанавливать и использовать переменную $some_data без возникновения коллизии имен между одной и другой глобальными переменными. Скрипт "знает" имя пакета, в пределах которого находится переменная, и он организует доступ к ней, использует имя пакета в качестве префикса к имени переменной.

Пример: В следующем примере имя пакета package_one или package_two добавляется в качестве префикса к имени переменной some_data:

```
&sub_one;
&sub_two;
print "Переменная 1 $package_one'some_data\n"
print "Переменная 2 $package_two'some_data\n"
```

4. Когда вы используете пакеты языка Perl, можете создать уникальное пространство имен в пределах текущего файла исходных кодов путем помещения инструкции `package` в начале файла.

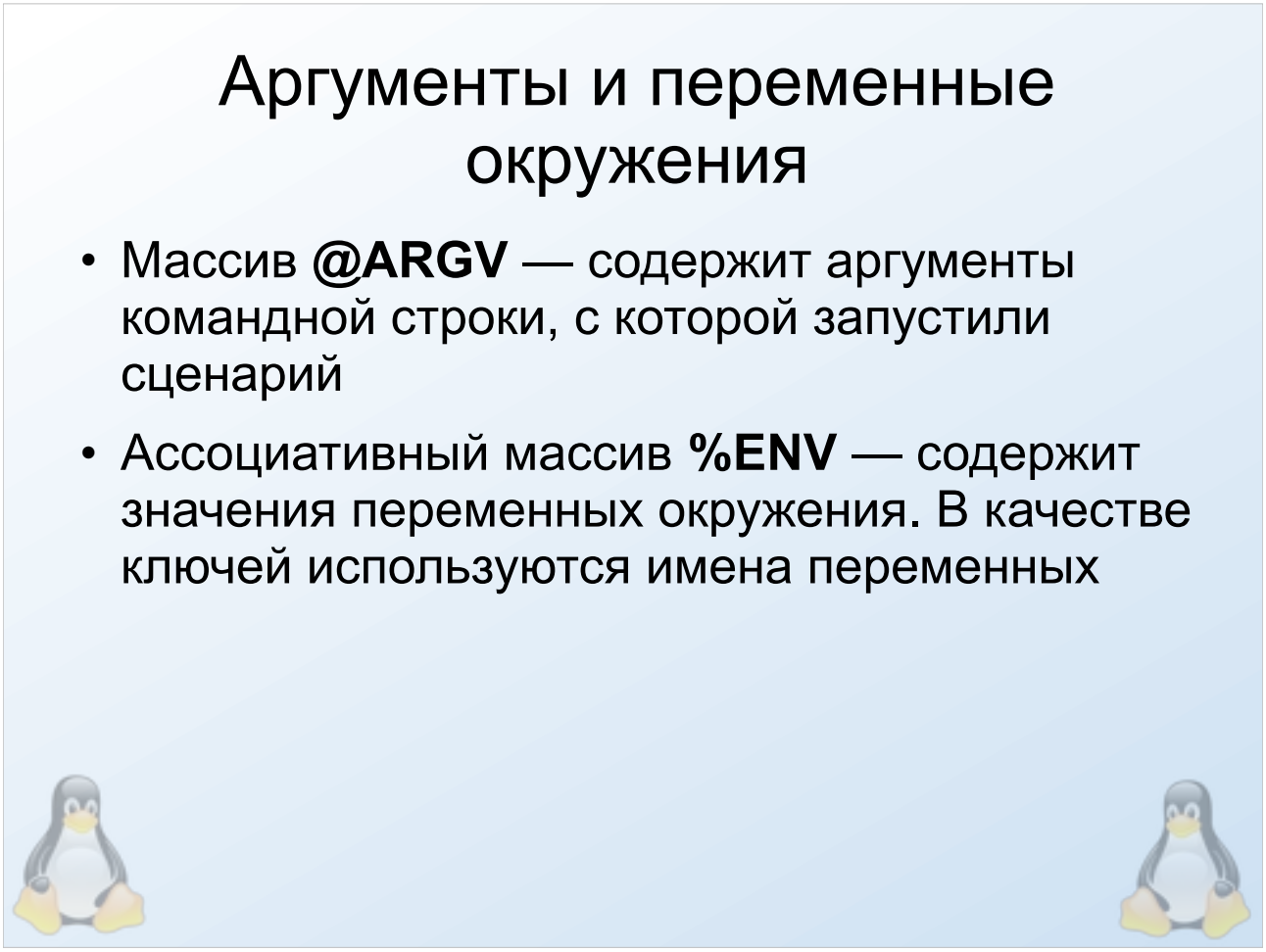
Пример:

```
package some_package_name
$some_data = 1;
sub some_sub
    {
        return $some_data;
    }
```

Примечание: В данном случае переменная $some_data существует только в пакете и поэтому защищена от некорректного доступа. Использование пакета, таким образом, обеспечивает данным ту же область видимости, что и в языке программирования C, где глобальные переменные имеют своей областью видимости тот файл исходных кодов, в котором они объявлены. При вызове подпрограммы из другого файла скриптов необходимо использовать имя пакета:

```
require 'some_package.pl';
print &some_package_name'some_sub;
```

7.10. Аргументы командной строки



1. Всякий раз, когда запускается скрипт, Perl помещает аргументы командной строки скрипта в списочную переменную `@ARGV`.

Пример: Следующий фрагмент программы служит для вывода аргументов командной строки на дисплей:

```
while ($arg = shift @ARGV)
    {
        print "$arg\n";
    }
```

7.11. Доступ к переменным окружения

1. При запуске скрипта Perl помещает копии переменных окружения в ассоциативный массив сименем %ENV.

Пример: В следующей инструкции массив %ENV используется для вывода текущего каталога:

```
print "$ENV{PWD}\n";         # Выведет текущий каталог
```

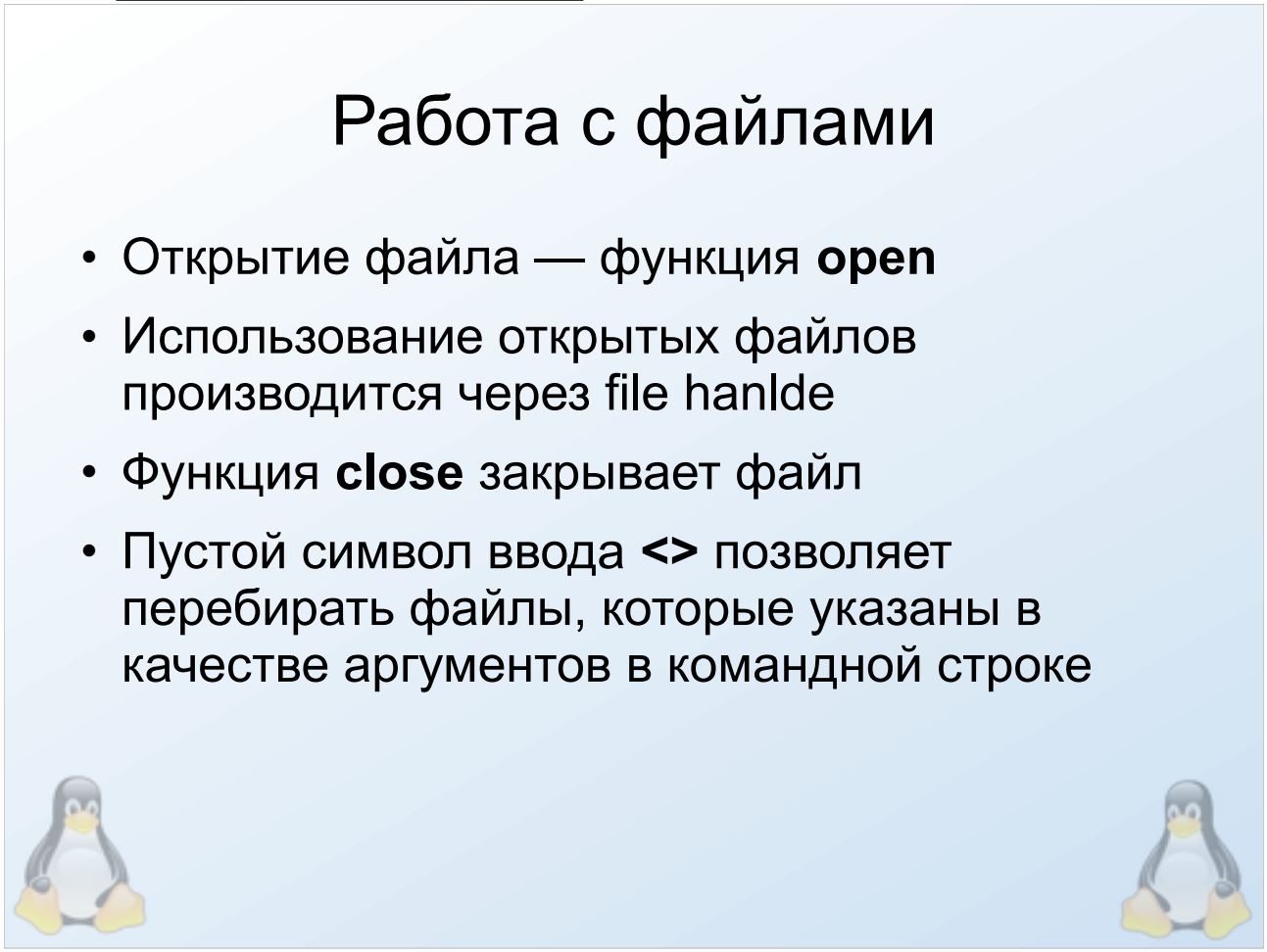
2. Кроме получения значений из массива %ENV, скрипты также могут изменять элементы массива. Такие изменения массива %ENV изменят установку переменных окружения для всякого процесса-потомка,создаваемого скриптом.

Пример: следующая инструкция использует массив %ENV для изменения текущего пути:

```
$ENV{PWD} = '/home';.
```

Примечание: Изменения, которые скрипт делает в массиве %ENV, не повлияют на исходные переменные окружения. Иными словами, после окончания работы скрипта переменные окружения системы не изменятся.

7.12. Файловый ввод и вывод



1. Perl специально разрабатывался для того, чтобы служить адекватным средством для чтения и записи в текстовые файлы.
2. Тем не менее, Perl может выполнять функции по произвольному доступу и вводу-выводу бинарных файлов.
3. Операции по работе с файлами требуют указатель файла (file handle), который является переменной, соответствующей конкретному файлу.
4. По умолчанию каждый скрипт на языке Perl имеет три стандартных указателя,которые Perl автоматически открывает при запуске скрипта: `STDIN, STDOUT, STDERR`. Эти три стандартных указателя отвечают стандартным потокам STDIN, STDOUT, STDERR.
5. Для того чтобы скрипт использовал файл, он должен вызвать функцию open. Она имеет следующий вид:

   ```
   open(FileHandle[, FileName])
   ```

6. Функция `open` языка Perl не содержит параметра, указывающего режим открытия файла. Perl определяет режим открытия файла, основываясь на имени файла. Таблица иллюстрирует связь режима открытия файла и имени файла.

Имя файла	Операция
FILE	Открыть файл только для чтения

<FILE	Открыть файл для чтения (то же самое, что <FILE>)
>FILE	Создать файл для записи
>>FILE	Открыть файл для добавления в его конец
+>FILE	Создать файл для чтения/записи
+<FILE	Открыть файл для чтения/записи
| CMD	Открыть канал из процесса, исполняющего команду <CMD>
CMD |	Открыть канал процессу, исполняющему команду <CMD>

Примечание: Режим канального (pipe) потока может существовать не на всех системах.

7. Если в вызове функции `open` опущено имя файла, то Perl подразумевает, что имя файла содержится в строковой переменной `$FileHandle`. Когда скрипт завершил использование файла, он закрывает его, используя функцию `close`, как показано ниже:

```
close(FileHandle);
```

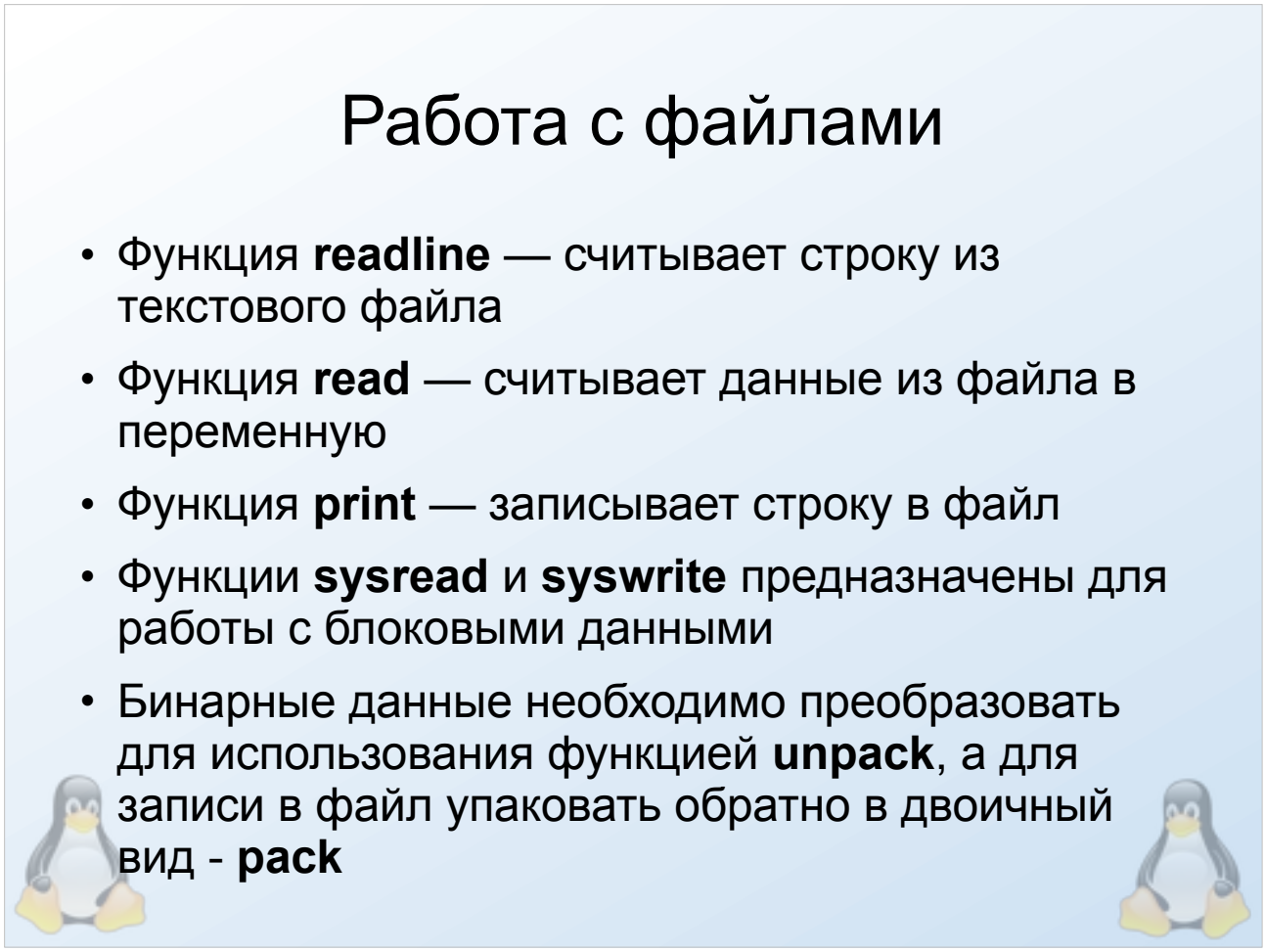
Пример: фрагмент программы иллюстрирует использование функций `open` и `close`:

```
open(InFile, "test.dat") || die;       # открываем для чтения test.dat
open(OutFile, ">test.dat") || die;     # создаём test.dat
$AuxFile = ">>test.dat";
open(Aux, $AuxFile) || die;            # открывает для дополнения test.dat
close(InFile);
close(OutFile);
close(Aux);
```

Примечание: Обратите внимание, что указатели файлов не имеют обычных односимвольных префиксов.

8. В операционной системе MS-DOS Perl поддерживает дополнительную функцию, которая называется `binmode` и позволяет файловому вводу/выводу переключаться между текстовым и бинарным режимами. В большинстве же систем различие между текстовым и бинарным режимами не имеет значения. Однако для операционной системы MS-DOS символ новой строки представляет собой последовательность из двух символов (CR+LF). Поскольку большинство программ не ожидают встретить два символа в конце строки, то система ввода/вывода должна выполнить преобразование. Для того чтобы можно было использовать функцию binmode, соответствующий указатель может быть открыт. Функция binmode имеет следующий формат:

```
binmode(FileHandle);
```



9. Функции `read` и `readline` позволяют считать данные из файла.

10. Простейшим способом для чтения скриптом строки из текстового файла (аналог `readline`) служит использование оператора `<FILEHANDLE>`.

11. В языке Perl указатель файла, окруженный треугольными скобками, становится символом ввода (input-symbol).

Пример: следующий фрагмент программы иллюстрирует использование символа ввода для чтения и вывода на экран содержимого файла Test.dat.

```
open(InFile, "Test.dat") || die;
while ($line = <InFile>)
    {
        print $line;          # Выведет строку из файла
    }
close(InFile);
```

Примечание: Когда символ ввода достигает конца файла, он возвращает значение false, которое в данном случае заканчивает выполнение цикла while.

12. Существует специальный (пустой) символ ввода, обозначаемый <>, который имеет специальное применение.

    1. В первый раз, когда скрипт использует пустой символ ввода <>, он анализирует аргументы командной строки.

2. Если строка `@ARGV` является пустой, то входной символ <> читает из STDIN.
3. Если `@ARGV` не пуста, то Perl открывает первый из файлов, указанных в переменной `@ARGV`, и читает содержимое файла.
4. Когда Perl заканчивает обработку одного файла, он приступает к следующему.
5. После того как скрипт прочитал все файлы, символ <> возвращает значение `false`.

13. Скрипты языка Perl также могут использовать символ ввода для чтения всего содержимого файла в массив так, что каждая строка файла становится элементом массива.

Пример: следующая инструкция читает из файла STDIN в массив `@lines`:

```
@lines = <STDIN>;
```

14. Запись данных в файл осуществляется функцию `print`. Полный формат функции `print` имеет следующий вид:

```
print [FileHandle]   List;
```

15. Если функция `print` не получает в качестве аргумента указателя файла, то она посылает вывод в STDOUT.

Пример: Следующий фрагмент программы иллюстрирует использование функции `print` для добавления данных в выходной файл:

```
open(LogFile, '>>logfile.dat') ||    die;
###############
($m, $d, $y) = (localtime(time)) [4,3,5];
print LogFile "Captain's log, Stardate    ++m$/$d/$y\n";
close(LogFile);
```

Примечание: Указатель файла и выходной список не разделяются запятой.

16. Если скрипт должен работать с файлом, который ориентирован на работу с блоками, а не потоками, то скрипт может использовать функции `sysread` и `syswrite` для обработки фиксированных блоков данных. Функции `sysread` и `syswrite` имеют следующие форматы:

```
$result = sysread(FileHandle, $Var, Length[, Offset]);
$result = syswrite(FileHandle, $Var, Length[, Offset]);
```

Примечание: Если в вызове функций указывается сдвиг от начала файла(Offset), то функции выполнят поиск места, с которого они начнут операции ввода/вывода. Функции sysread и syswrite обе передают данные, используя скалярную переменную строкового типа. Поскольку функции обрабатывают фиксированные блоки памяти, то данные могут содержать бинарные значения, включая нули и маркеры конца файла. Если в вызове функции указывается сдвиг от начала файла (Offset), то функция выполняет поиск места в файле, с которого начинает выполнять операции ввода/вывода.

17. Если вы работаете с блоками данных, то скрипты могут также использовать следующие функции ввода/вывода:

```
$result = seek(FileHandle, Position, Base);
$result = tell(FileHandle);
$result = eof(FileHandle);
```

18. Функция `seek` работает в точности так же, как `fseek` - функция библиотеки времени выполнения языка C. Параметр `Position` задает позицию относительно начала отсчета,

которая в свою очередь задается параметром Base следующим образом:

- 0 Поиск от начала файлов
- 1 Поиск от текущей позиции
- 2 Поиск от конца файла

19. Функция `tell` языка Perl работает в точности так же, как фикция `ftell` библиотеки времени выполнения языка C. Эта функция возвращает текущую позицию в файле, с которой выполняются операции чтения или записи.

20. Функция `eof`, так же как и функция feof языка C, возвращает значение <истинно> или <ложно>, которое скрипт может использовать для определения достижения конца файла.

21. Когда скрипт на языке Perl читает блок бинарных данных, используя функцию `sysread`, он помещает эти бинарные данные в скалярную строковую переменную. Perl не заботится о том, что это заданные, содержат ли они нули или значения, не являющиеся ASCII-символами. В пределах символьной строки Perl принимает байты как байты.

22. Если данные соответствуют кодовой таблице ASCII, то скрипт может их обрабатывать, как любой текст. Но если данные представляют собой бинарные величины, то скрипт обязан распаковать их перед тем, как Perl сможет обработать эти данные.

23. Скрипты Perl распаковывают данные, используя функцию `unpack`, которая имеет следующий формат:

```
$result = unpack(Template, Expression);
```

Примечание: Expression является обычной строковой переменной, которая содержит бинарные данные, прочитанные функцией sysread, но можетбыть также выражением, которое необходимо интерпретировать как строку. Template представляет собой символьную строку-шаблон,описывающую, как интерпретировать значения в операнде Expression.

Пример: Следующий фрагмент программы иллюстрирует использование функции `unpack`:

```
($r, $g, $b) = unpack("C3", $color);# распакует в 3 символа
@longwords = unpack("L*", $data); # распакует в список длинных слов
@stuff = unpack("S2L", $bin);   # распакует в 2 shorts и long
```

24. Каждый символ шаблона может сопровождаться числом, указывающим, сколько раз использовать этот символ. Если вместо числа стоит звездочка (*), то операция будет выполняться для всех остающихся данных в строке. Если число не поставлено, то она выполняется однократно.

25. Скрипт может поместить любое число символов шаблона в строку `Template`. В таблице перечисляются символы,входящие в строковый параметр `Template` вместе с описанием влияния каждого из них на выполнение функции `unpack`.

Символ шаблона	Описание
a	Строка ASCII без нулевого символа
A	Строка ASCII без нулевого символа
b	Битовая строка (младший бит идет первым)
B	Битовая строка (старший бит идет первым)
c	Однобайтовый символ со знаком

C	Однобайтовый символ без знака
d	Значение с плавающей запятой, двойной точности
f	Значение с плавающей запятой, одинарной точности шаблона
h	Строка шестнадцатиричных значений (младшие разряды идут первыми)
H	Строка шестнадцатиричных значений (старшие разряды идут первыми)
i	Целое со знаком
I	Целое без знака
l	Целое со знаком типа long
L	То же, только без знака
n	Короткое целое
N	Длинное целое
p	Указатель на строку
s	Короткое целое со знаком
S	Короткое целое без знака
u	Раскодировка строки
v	Короткое целое
V	Длинное целое
x	Пропустить вперед один байт
X	Пропустить назад один байт
@	Перейти на указанную позицию в строке

26.Для вывода бинарных данных скрипт должен запаковать скалярные величины в строки бинарных символов. Для этого используется функция `pack`, формат которой указан ниже:

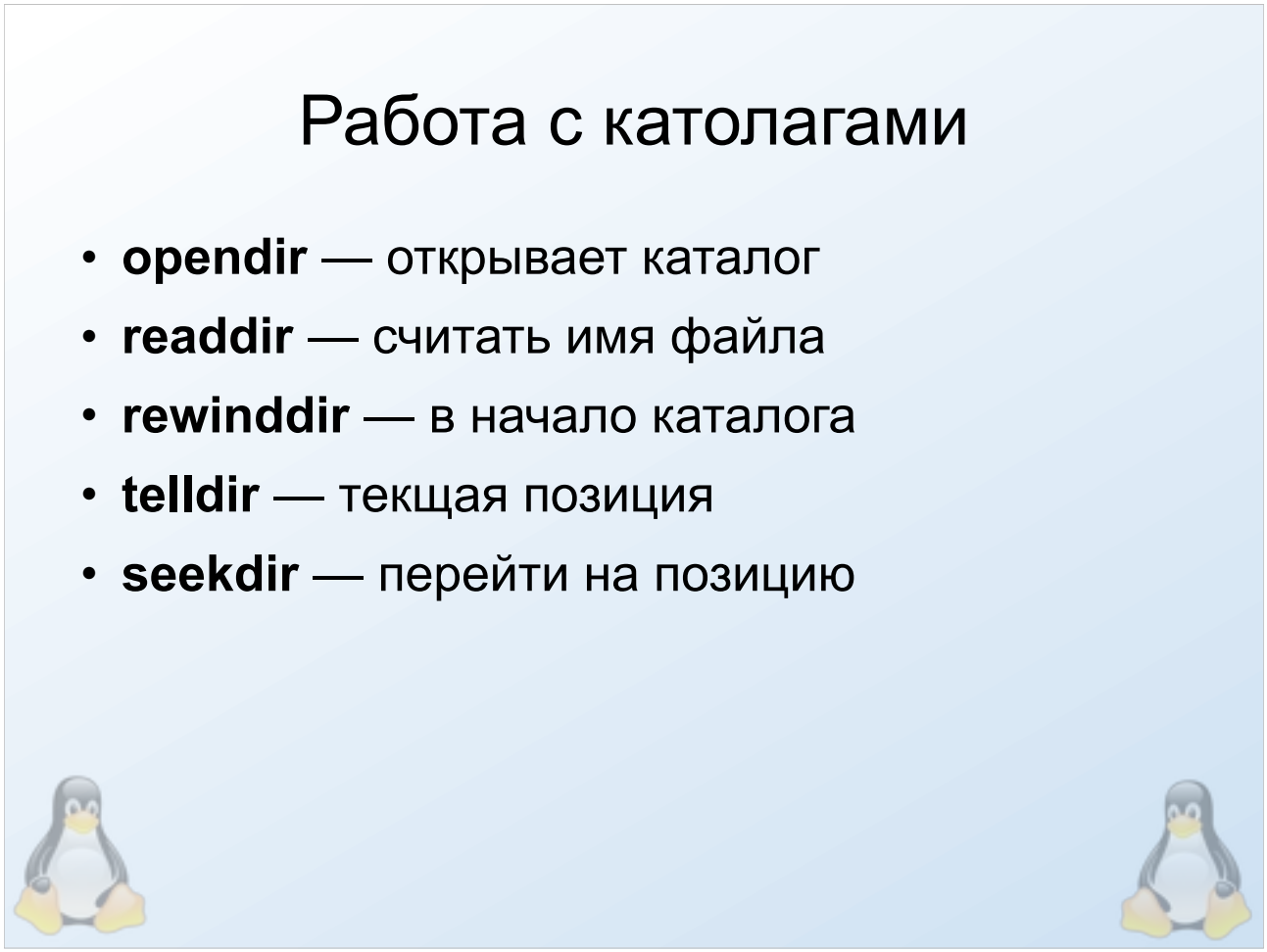
```
$result = pack(Template, List);
```

Пример: Следующий фрагмент программы иллюстрирует использование функции `pack`:

```
$color = pack("C3", $r, $g, $b);
$data = pack("L*", @longword);
$bin = pack("S2L", @stuff);
```

Примечание: Функция pack использует те же самые символы шаблона, что и функция unpack, за исключением символов a. A, u, x, X, @.

7.13. Работа с каталогами



1. Для открытия каталога скрипты используют функцию `opendir`, передавая указатель каталога и путь к нему.
2. Для чтения списка файлов, содержащихся в каталоге, скрипт использует функцию `readdir`.
3. Для закрытия каталога используется функция `closedir`.

Пример: Следующий фрагмент программы иллюстрирует использование функции `readdir` для того, чтобы вывести на экран список файлов в текущем каталоге:

```
opendir(Dir, $INC[2]) || die;
while ($file = readdir(Dir))
    {
        print "$file \n"
    }
closedir(Dir);
```

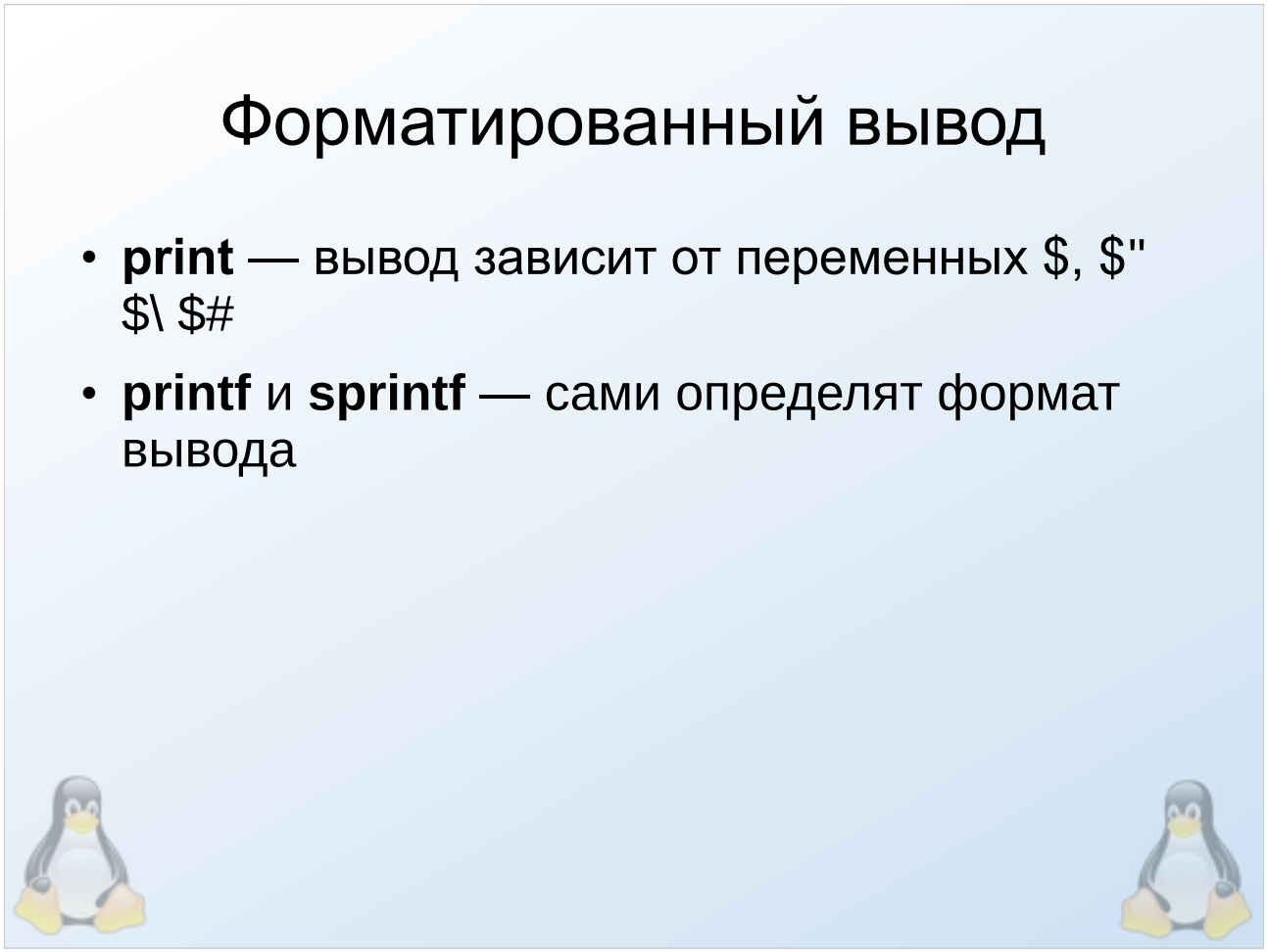
Примечание: В этом фрагменте используется переменная $INC[2] для доступа к текущему каталогу. Изменяя $INC[2] на $ARGV[0], скрипт выводит на экран список файлов, содержащихся в каталоге, который вы указали в командной строке.

4. В дополнение к функциям для работы с каталогами, которые были рассмотрены выше, Perl предлагает еще набор функций, которые позволяют позиционировать текущий указатель в списке каталога:

```
$result = rewinddir(DirHandle);
$result = telldir(DirHandle);
$result = seekdir(DirHandle, Position);
```

7.14. Форматированный вывод

7.14.1. Использование функции print



1. Perl обеспечивает специальные переменные, которые влияют на работу функции `print`. В таблице кратко характеризуются эти специальные переменные.

Переменная	Назначение
$,	Разделитель для элементов печати
$"	Разделитель элементов списка при интерполяции строки
$\	Разделитель для выходных записей
$#	Форматирование числового выхода (по умолчанию '%.20g')

Примечание: Для использования этих специальных переменных достаточно просто присвоить им значения, которые вы хотите.

Пример: в следующем фрагменте программы используется переменная $ для того,чтобы задать сепаратор между элементами печати:

```
$, = '*';
@list = 1..10;
```

```
print @list;      # Выведет 1*2*3*4*5*6*7*8*9*10
```

7.14.2. Форматированный вывод данных функцией printf

1. Perl имеет функции `printf` и `sprintf`, которые очень похожи на соответствующие функции библиотеки времени выполнения языка C. Они имеют следующий формат:

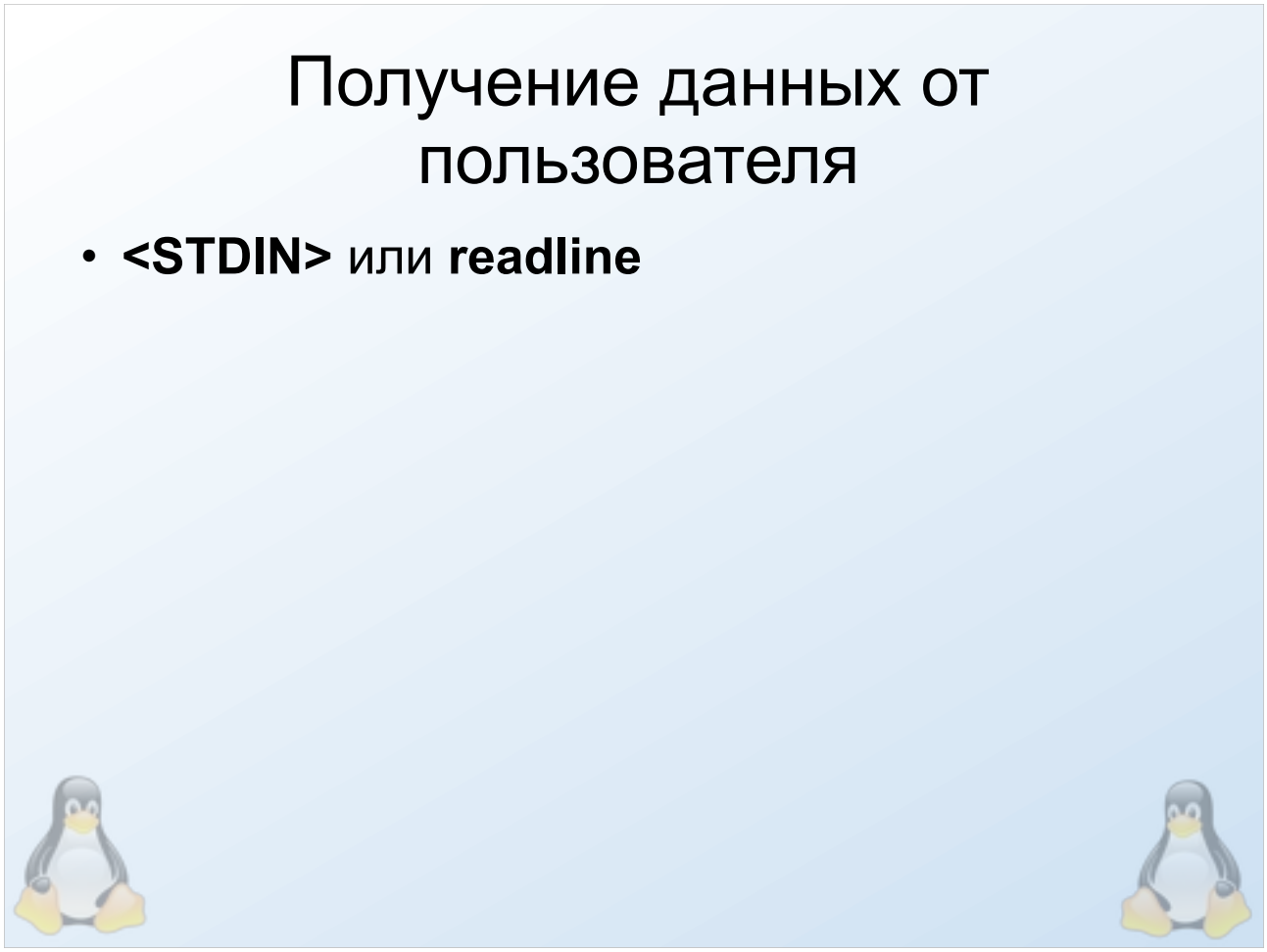
```
$result = printf([FileHandle] Format, List);
$result = sprintf(Format, List);
```

2. По умолчанию функция `printf` посылает форматированный выход на стандартный выход STDOUT, а функция `sprintf` возвращает форматированную строку.

Пример: Следующий фрагмент программы иллюстрирует использование функций `printf` и `sprintf`.

```
$precision = 2;
$pi = 3.1415;
printf("%.2f\n", $pi);            # выведет 3.14
printf("%.${precision}f", $pi); # выведет 3.14
```

7.15. Получение данных от пользователя



1. Чтение строковой переменной из стандартного потока ввода производится конструкцией вида:

```
$str = <STDIN>;
```

2. Другой способ, воспользоваться функцией `readline`

Пример:

```
$str = readline(STDIN);
```

3. Для обработки нескольких строк можно воспользоваться циклом:

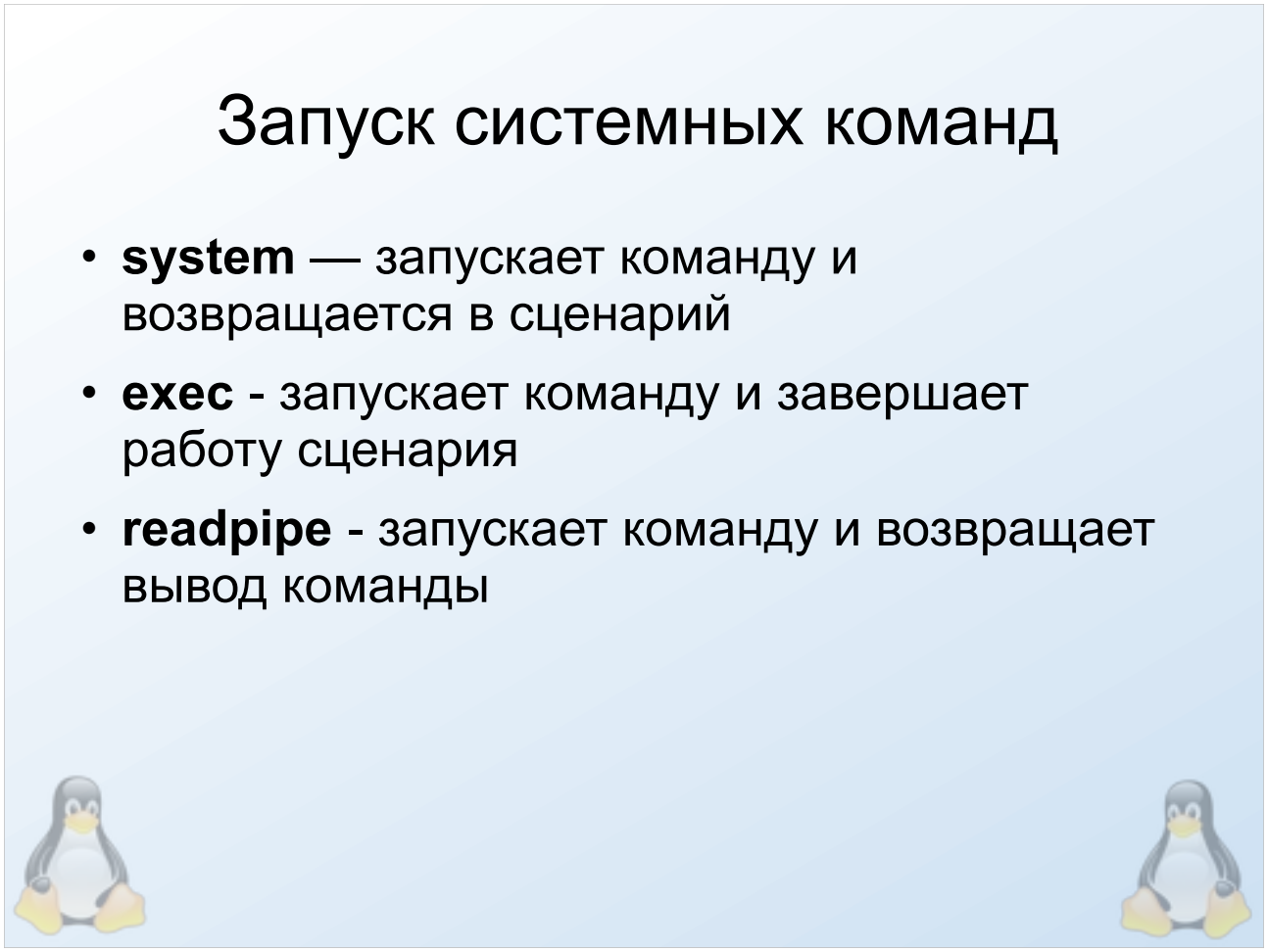
Пример:

```
while ($str = <STDIN>){
   print $str;
}
```

4. Закончить обработу строк надо передачей признака конца файла `^D`.
5. Для получения массива строк используйте конструкцию:

```
@line = <STDIN>;
```

7.16. Запуск системных команд



1. В Perl имеются несколько функций управления процессами (см. man perlfunc).
2. Среди этих функций несколько удобных для использования в административных скриптах:
    1. `system`
    2. `exec`
    3. `readpipe`
3. Функция `system` запускает указанную в качетсве аргумента команду и возвращается в сценарий.

Пример:

```
system("pwd");   #напечатает текущий каталог
```

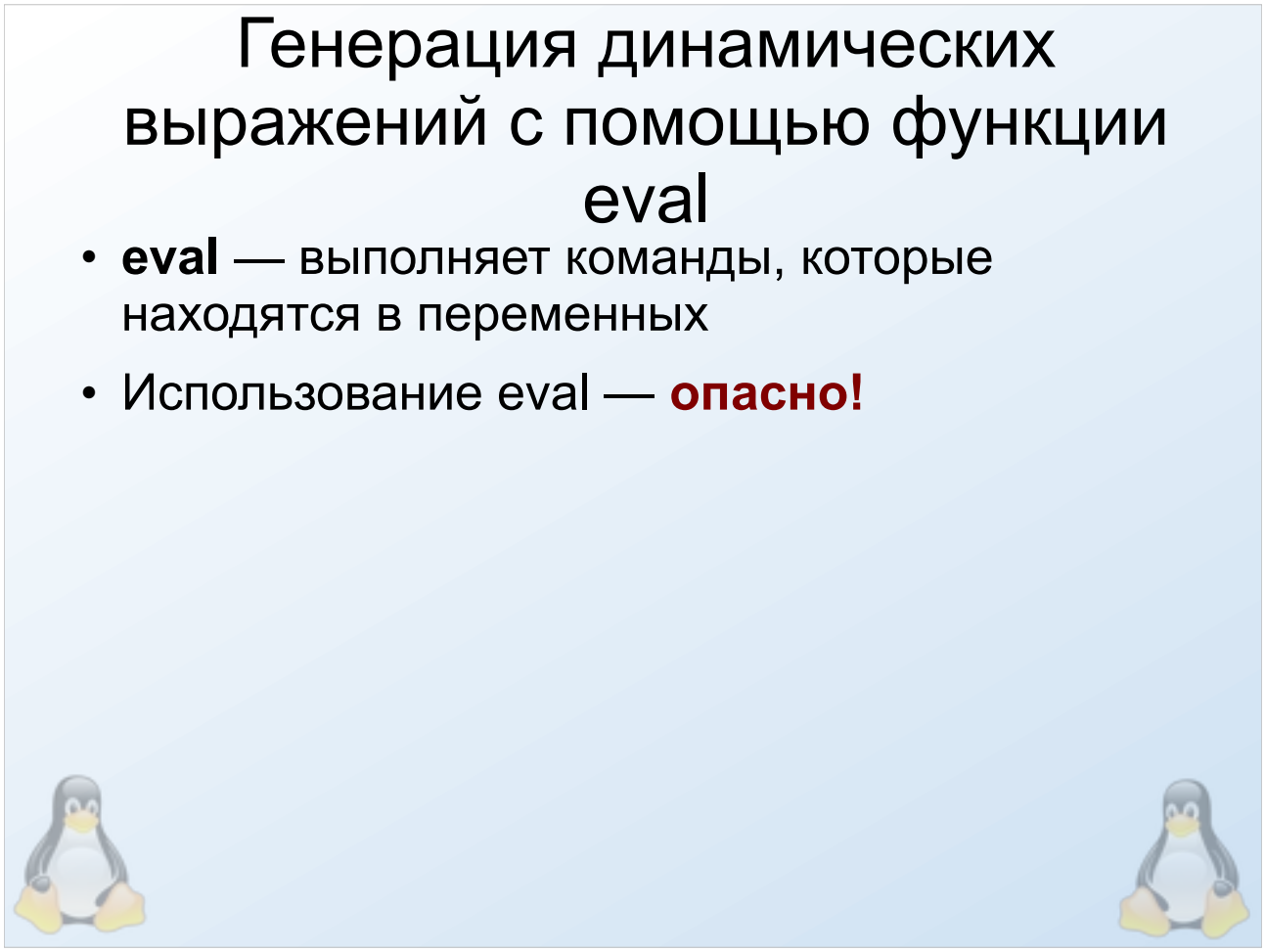
4. После запуска команды функцией `exec` происходит замещение “процесса-скрипта” процессом запускаемой команды. Это приводит к выходу из скрипта.
5. Если надо передать переменной вывод запущенной команды, то удобно использовать функцией `readpipe`.

Пример:

```
$str = readpipe("ls");
```

Примечание: Функция readpipe появилась в perl5. Вместо нее можно воспользоваться стандартной функцией open (см. файловый ввод и вывод).

7.17. Генерация динамических выражений с помощью функции eval



1. Поскольку Perl является интерпретируемым языком, ваши скрипты могут использовать Perl для того, чтобы генерировать код налету, то есть динамически, во время исполнения скрипта.
2. Именно так работает отладчик Perl (который сам является Perl-программой с именем Perldb.PL).
3. Проектируя скрипты, вы можете использовать такой динамический код, который создает динамические переменные или даже строит специальные подпрограммы.
4. Perl оценивает динамические выражения, используя функцию eval.

Пример: В следующем примере создается динамическая инструкция путем присвоения переменной в качестве значения текста, который, собственно, и содержит требуемую инструкцию языка Perl. Дальнейшие инструкции используют функцию eval для того, чтобы выполнить эту инструкцию:

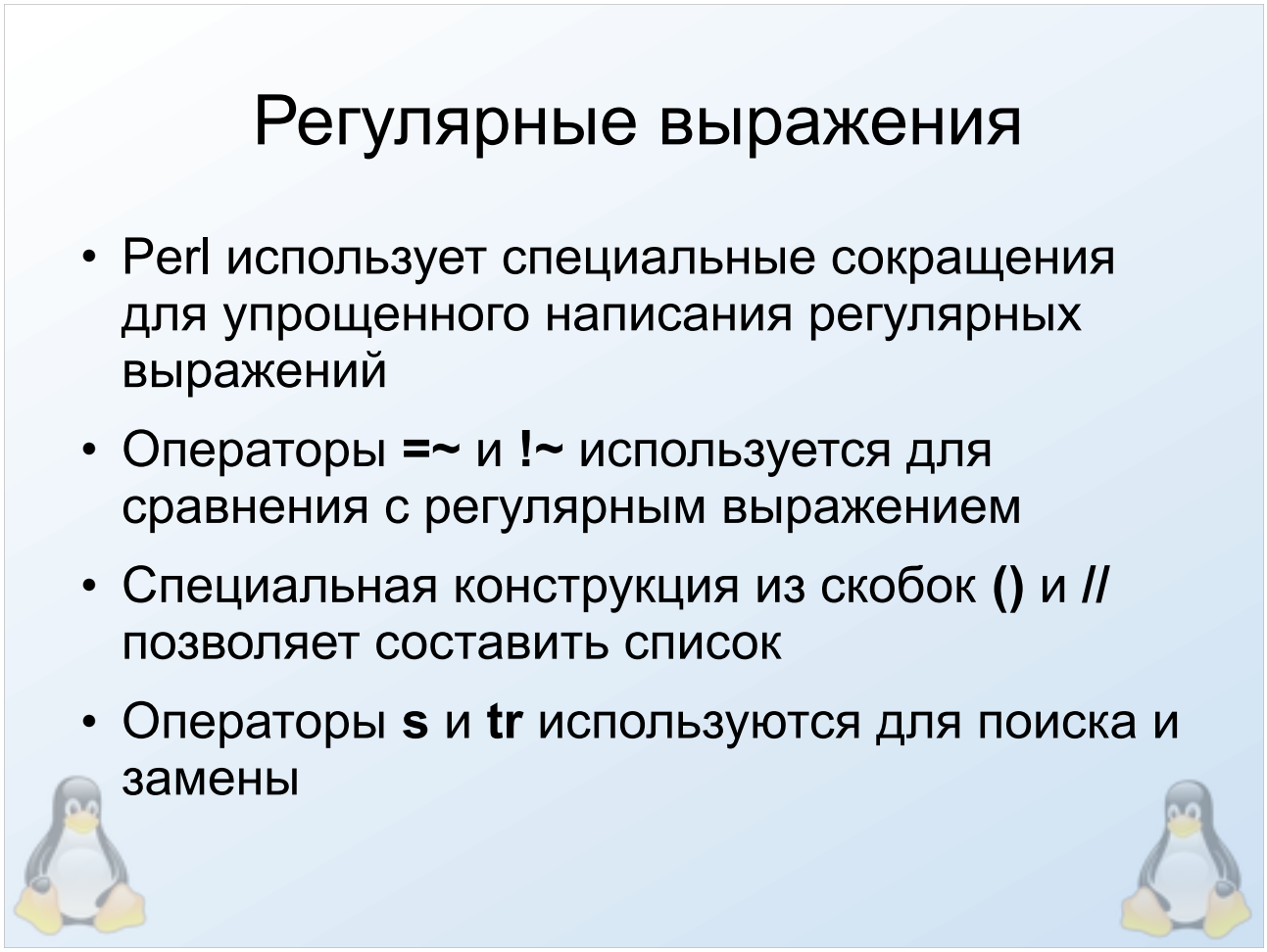
```
$perl_statement = 'print "Hello, world\n";';
eval $perl_statement;            # выведет Hello, world
$i = 1001;
$varname = '$i';
```

```
print eval $varname;              # выведет значение $i
```

5. Использование скриптом функции `eval` таит в себе опасности, в особенности, если скрипт передает функции `eval` данные, полученные от пользователя.
6. Используя функцию `eval`, скрипт может выполнить произвольную команду языка Perl, включая даже системные команды. Это дает возможность пользователю контролировать программу, что может быть особенно рискованным для публичных сетей.

7.18. Регулярные выражения

7.18.1. Синтаксис регулярных выражений



1. Для сокращения размеров регулярных выражений, Perl использует специальные символы. Таблица содержит список некоторых из символов, используемых скриптами языка Perl в регулярных выражениях.

Символ	Описание
.	Соответствует любому символу (за исключением символа новой строки)
(..)	Группирует последовательность элементов
+	Удовлетворяет предыдущему образцу один или большее количество раз
?	Удовлетворяет образцу нуль или один раз
*	Соответствует образцу любое количество раз
[...]	Соответствует символу из заданного множества
[^...]	Соответствует символу из множества, полученного отрицанием

(...|...|...)	Соответствует одной из альтернатив
^	Соответствует началу строки
$	Соответствует образцу в конце строки
{n,m}	Соответствует образцу от n до m раз
{n}	Соответствует образцу точно n раз
{n,}	Соответствует образцу минимум n раз
\n\t etc.	Соответствует знаку новой линии, символу табуляции и т. д.
\b	Соответствует на границе слова
\B	Соответствует внутри границ слова
\d	Соответствует цифре
\D	Соответствует не цифре
\s	Соответствует пробелу
\S	Соответствует не пробелу
\w	Соответствует букве или цифре
\W	Соответствует символу, не являющемуся ни буквой, ни цифрой

2. Perl помещает регулярные выражения (образцы, шаблоны) в слэши, т. е. в наклонные черточки.

Пример: Следующий фрагмент программы иллюстрирует регулярные выражения языка Perl:

```
# the following regular expressions are true if:
/ig/          #    string contains 'ig'
/(b|d|f)ig/      # string    contains 'big', 'dig' or 'fig'
/[0-9]+/        #    string contains a number
/[A-Za-z][A-Za-a0-9_]*/ # string contains an identifier
```

7.18.2. Использование регулярных выражений для поиска по ключевым словам

1. Скрипты языка Perl используют регулярные выражения для того,чтобы упростить сравнение строк. Для того чтобы проверить, содержит ли строка заданный образец, скрипт может использовать регулярные выражения следующим образом:

```
if ($str =~ /pattern/)
```

Примечание: В данном случае регулярные выражения принимают значение <истинно>, если образец найден в строке ($str). Если строка по содержит образца, то выражение возвращает значение <ложно>. Например, следующее выражение проверяет, содержит ли строка текст Web Programming:

```
if ($str =~ /Web Programming/)
```

2. Perl поддерживает также другую форму сравнения с образцом,использующую оператор (=~), который добавляет отрицание результата: (!~). Этот оператор эквивалентен выражению !($str=~/pattern/).

7.18.3. Использование регулярных выражений для анализа входных данных

1. Используя символы группировки () внутри регулярного выражения, скрипт может извлечь соответствующие образцу значения из строки и сформировать из них список.

Пример: следующий фрагмент программы использует регулярные выражения для того, чтобы извлечь месяцы, дни и годы из списка:

```
$str = " January 1, 1997, ";
($m, $d, $y) = $str =~ /\s*(\S*)\s + (\d+)\D + (\d{4})/;
```

Примечание: В этом случае можно прочитать регулярные выражения следующим образом: Пропустить вначале любой специальный символ; записать все символы, не являющиеся специальными, в переменную $m - (переменная для обозначения месяцев); пропустить специальный символ; поместить все цифры в переменную $d (переменная для записи дней); пропустить все знаки, не являющиеся цифрами; записать четыре цифры в переменную $y (переменная для обозначения лет).

7.18.4. Регулярные выражения для поиска и замены строк

1. Perl поддерживает два регулярных выражения, которые модифицируют проверяемую строковую переменную. В записанной дальше инструкции Perl замещает часть строки, которая соответствует образцу, на заданную строку:

```
$str =~ s/pattern/replacement/;
```

Пример, следующая инструкция заменит слово <colour> на<color>:

```
$str =~ s/\bcolour\b/color/;
```

2. При наличии g в конце выражения указывает языку Perl на необходимость глобальной подстановки.

Пример: Небольшая модификация позволяет заменить все слова <colour>на <color>:

```
$str =~ s/\bcolour\b/color/g;
```

3. Используя суффикс i, можно задать выполнение поиска с учётом регистра.
4. В противоположность простой проверке на соответствие образцу, следующее выражение осуществляет также и замену символов:

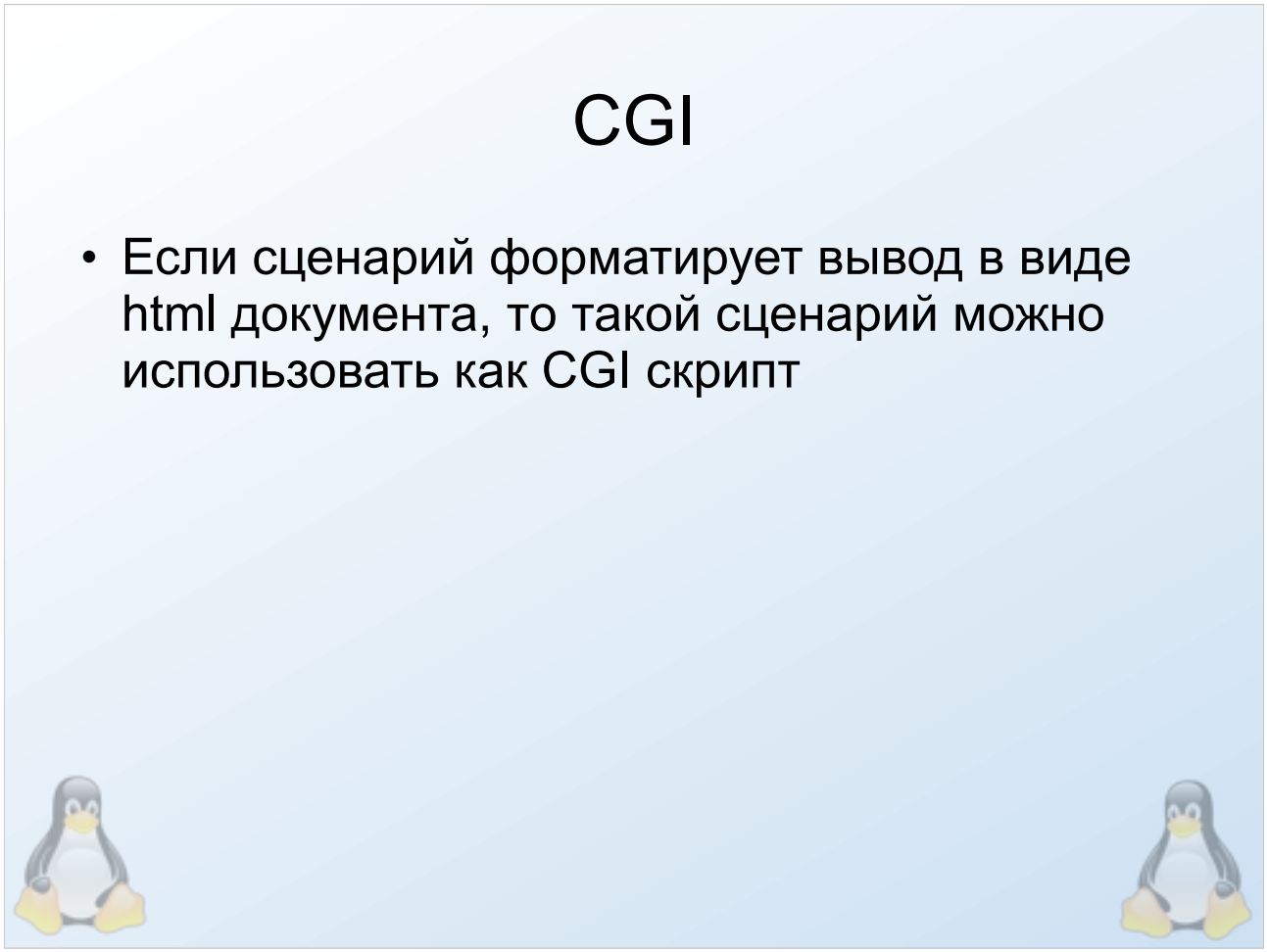
```
$str =~ tr/SearchList/ReplacementList/;
```

Пример: замена всех символов нижнего регистра теми же символами верхнею регистра может быть осуществлена таким образом:

```
$str =~ tr/a-z/A-Z/;  # меняет регистр, с нижнего на верхний
```

7.19. Создание скриптов cgi с помощью Perl

7.19.1. Вызов cgi-скрипта



1. В UNIX-системах программисты могут исполнять скрипты, написанные на языке Perl, как выполняемые файлы. Иными словами, для них нет необходимости делать что-то специальное для вызова скрипта.
2. В системах, основанных на DOS или Windows, некоторые серверы не исполняют скрипты Perl автоматически. В этом случае придется писать пакетный файл, который вызывает команды Perl для запуска скрипта. Для некоторых систем придется подробнее познакомиться (документацией по вашему HTTP-серверу относительно запуска скриптов Perl.)
3. Большинство серверов HTTP предполагает, что скрипты CGI находятся в каталоге под названием cgi-bin. В этом случае можно вызвать скрипт с помощью URL, подобно следующему:

```
http://your-domain/cgi-bin/your-script
```

Примечание: В URL задан каталог cgi-bin, но в действительности скрипт может находиться где угодно в системе. Вы должны определить его расположение при установке сервера HTTP.

7.19.2. Создание текста и html-документа с использованием языка Perl

1. Создание текстовых документов с использованием языка Perl представляет собой тривиальную задачу. Вам только необходимо удостовериться, что вы поместили правильный HTML заголовок в начале текстового документа.

Пример: следующий скрипт создает простой текстовый документ, содержащий сообщение Hello, world:

```
print "Content-type: text/plain\n\n";
print "Hello, world\n";
```

2. Создание текстовых документов HTML также очень легко.

Пример: в следующем фрагменте текста создается простой документ HTML:

```
print <<HTML;
Content-type: text/html
<HTML>
<HEAD><TITLE>Test Using HTML</TITLE></HEAD>
<BODY>
<H1><CENTER>
Hello, world
</CENTER></H1>
</BODY></HTML>
HTML
```

Примечание: Эти примеры больше походят на исходные коды HTML, чем на программу на языке Perl. Действительно, если вы удалите несколько строк, вы будете иметь HTML-файл. Эта иллюстративная программа использует свойства языка Perl, которые не были рассмотрены в предыдущей главе, но они достаточно просты для понимания. Строка <<HTML и последняя строка с символами HTML представляют собой конструкцию, называемую <здесь-документ> (heredocument), которая перешла сюда из терминологии программирования shell для UNIX. <Здесьдокумент> представляет собой просто несколько строк литералов. Perl обрабатывает <здесь-документ> как строку в двойных кавычках. Поскольку <здесь-документ> делает исходный код на языке Perl легким для чтения, то использование такой конструкции является идеальным для создания HTML-документов. Используя конструкцию <здесь-документ>, скрипт на языке Perl может не иметь символов цитирования и символов новой строки, и не должен содержать функций printf. Конструкция <здесь-документ> широко используется в рассматриваемых далее в этой главе примерах.

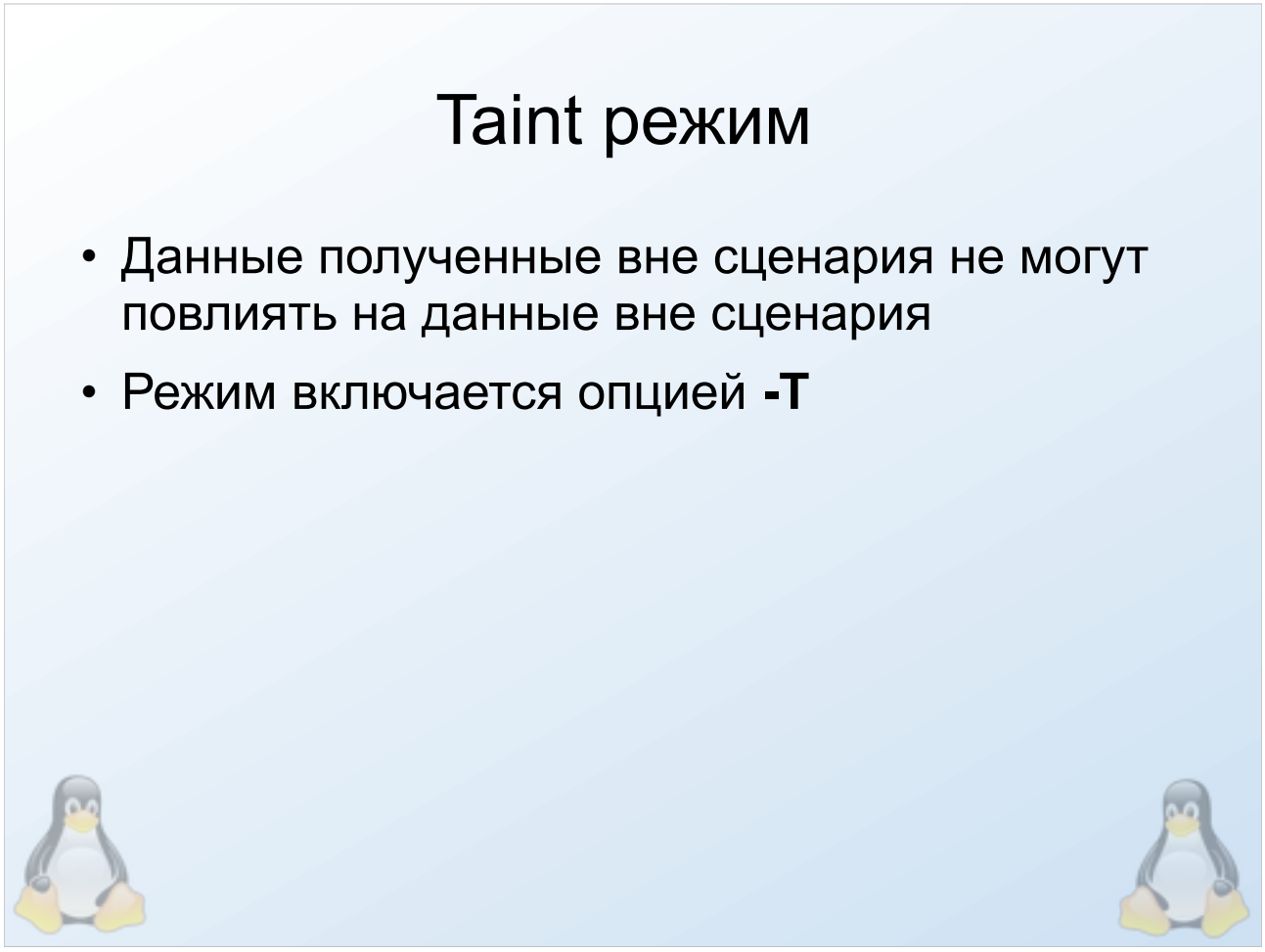
Пример: более лаконичный код с использованием модуля CGI:

```
$ cat test.pl
#!/bin/perl
use strict;
use warnings;
use CGI qw/:all/;

print
  header,
```

```
  start_html(-title=>'Test Using',-head=>meta({-http_equiv=>"Content-Type",
content=>"text/html; charset=utf-8"})),
  h1('Hello world!!!'),
  end_html;
```

7.20. Taint режим



1. При использовании сценариев, получающих данные от пользователей, могут возникать проблемы связанные с безопасностью.

Примечание: проблемы связаны с тем, что пользователь может подсунуть скрипту недопустимые параметры или эскейп последовательности, приводящие к ненормальному завершению работы сценария или запускаемых им команд.

2. Чтобы избежать таких проблем в Perl предусмотрен специальный режим работы, называемый ***taint-mode***.

Примечание: taint переводится как испорченный, запятнанный.

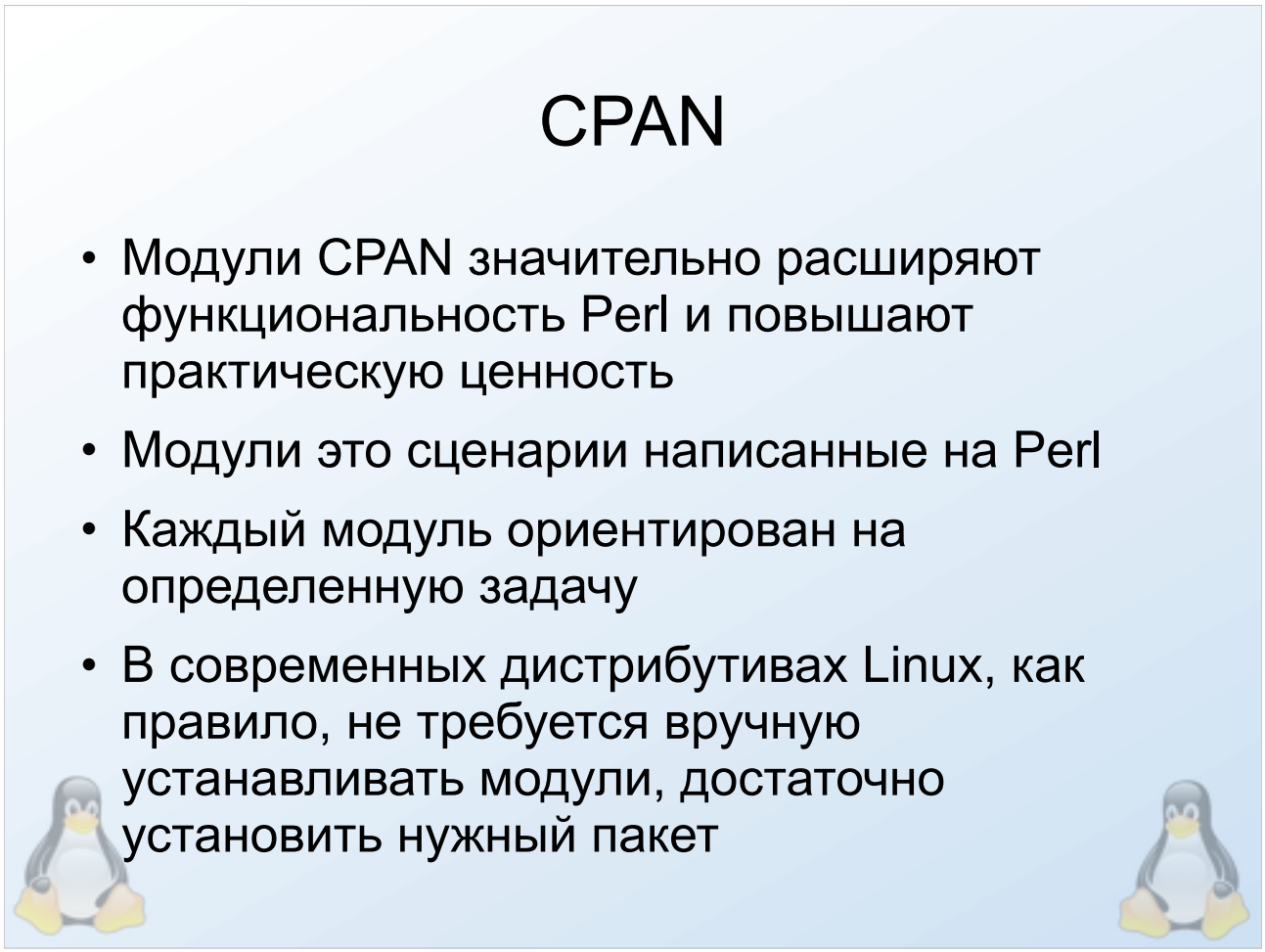
3. В taint-mode все данные, полученные от пользователя, помечаются как не заслуживающие доверия. Perl проверяет эти данные прежде чем использовать в системных командах.
4. Данные полученные вне сценария не могут повлиять на данные вне сценария.
5. В результате помеченные данные нельзя использовать прямо или косвенно в командах запускающих оболочки или командах модифицирующих файлы, каталоги или процессы.
6. Taint режим включается в сценариях опцией `-T`.

Пример:

```
#!/usr/bin/perl -T
```

7. Taint режим не панацея и он не снимает ответственности с программиста. Этот режим только помагает в написании сценариев.

7.21. Использование CPAN для интерактивной и автоматической инсталляции модулей



1. В Perl имеется большое число готовых модулей, которые позволяют использовать Perl практически в любой области.
2. Множество модулей включено в базовую поставку или доступно в веде отдельного пакета, но еще большее количество вы можете найти воспользовавшись Comprehensive Perl Archive Network (Всеобъемлющую Сеть Архивов Perl), называемую также *CPAN*, который располагается по адресу http://www.perl.com/CPAN/CPAN.html.
3. Модули находящиеся на CPAN можно скачивать и компилировать вручную (как правило последовательность команд компиляции и требуемые модули, перечислены в файле `README`, обычно входящем в состав дистрибутива модуля).
4. В состав дистрибутива Perl входит полезный модуль под названием `CPAN`. Он позволяет автоматизировать операции установки необходимых программисту модулей, включая установку модулей, от которых зависит устанавливаемый модуль.
5. Модуль работает в интерактивном и пакетном режимах, что позволяет использовать его всем пользователям. Настройки, которые влияют на то, откуда и как скачиваются модули, как правило задаются при первом запуске данного модуля, и хранятся в файле `$PERLLIB_PREFIX/$PERL_VERSION/CPAN/Config.pm`

Пример: `/usr/lib/perl5/5.6.1/CPAN/Config.pm`). Также в этом файле задаются дополнительные флаги для `Makefile` и прочая информация.

6. Вход в интерактивный режим производится запуском Perl следующей командной строкой `perl -MCPAN -e shell`.
7. При самом первом запуске данного модуля вам придется ответить на несколько вопросов относительно параметров системы и расположении сервера *CPAN*, с которого будет производиться загрузка модулей.
8. Для установки модуля, имя которого вы знаете точно, используется команда `install ModuleName`. Если устанавливаемому модулю для работы требуется модуль, который отсутствует в системе, то у пользователя запросят подтверждение на установку необходимого модуля.
9. При запуске команды `install` выполняется проверка того, нуждается ли модуль в обновлении.
10. Установка файлов в систему, производится только в том случае, если выполнение `make test` прошло без ошибок. Для принудительной установки модуля используется команда `force install ModuleName`.
11. Для поиска модуля используется команда `i /text/`, при этом будут найдены все модули в названии которых входит слово `text`. Затем полученные имена модулей можно использовать для установки.
12. Для поиска по именам пакетов, авторов, модулей и файлов дистрибутивов используются команды `b`, `a`, `m` и `d` соответственно.
13. Команда `get` только скачивает указанный ей пакет, и не выполняет больше никаких действий.
14. Команда `readme` отображает файл `README` для указанного дистрибутива.
15. Также полезной является команда `r`, которая перечисляет модули, версии которых меньше текущих версий соответствующих модулей, доступных на *CPAN*.
16. В пакетном режиме для программиста доступны все эти же команды, только они выполняются без запроса пользователя.

Пример: можно заставить Perl выполнять автоматическое обновление модулей, установленных в системе. Это выполняется с помощью команды:

```
perl -MCPAN -e 'CPAN::Shell->install(CPAN::Shell->r)'
```

17. Каждая из доступных команд, возвращающих списки модулей, также может возвращать списки идентификаторов модулей. Затем данные идентификаторы могут быть преобразованы в соответствующие реальные объекты с помощью метода `CPAN::Shell->expand("Module",@things)`. Затем с помощью полученного объекта может быть произведена установка модуля.

Пример: вот как будет выглядеть установка модулей `Net::FTP`, `MD5` и `Data::Dumper`:

```
for $mod (qw(Net::FTP MD5 Data::Dumper)){
    my $obj = CPAN::Shell->expand('Module',$mod);
    $obj->install;
}
```

18. Модуль `CPAN` поддерживает концепцию пакетов (`bundles`). Эта концепция упрощают работу с группами модулей, связанных общими свойствами. Пакеты являются обычными модулями в пространстве имен `Bundle::`.
19. Они не определяют никаких функций или методов, и как правило содержат только документацию. Когда пользователь устанавливает такой модуль, то происходит установка всех модулей, которые перечислены в тексте заданного для установки модуля.

20. Для удобной работы с модулем `CPAN` лучше установить дополнительные модули.

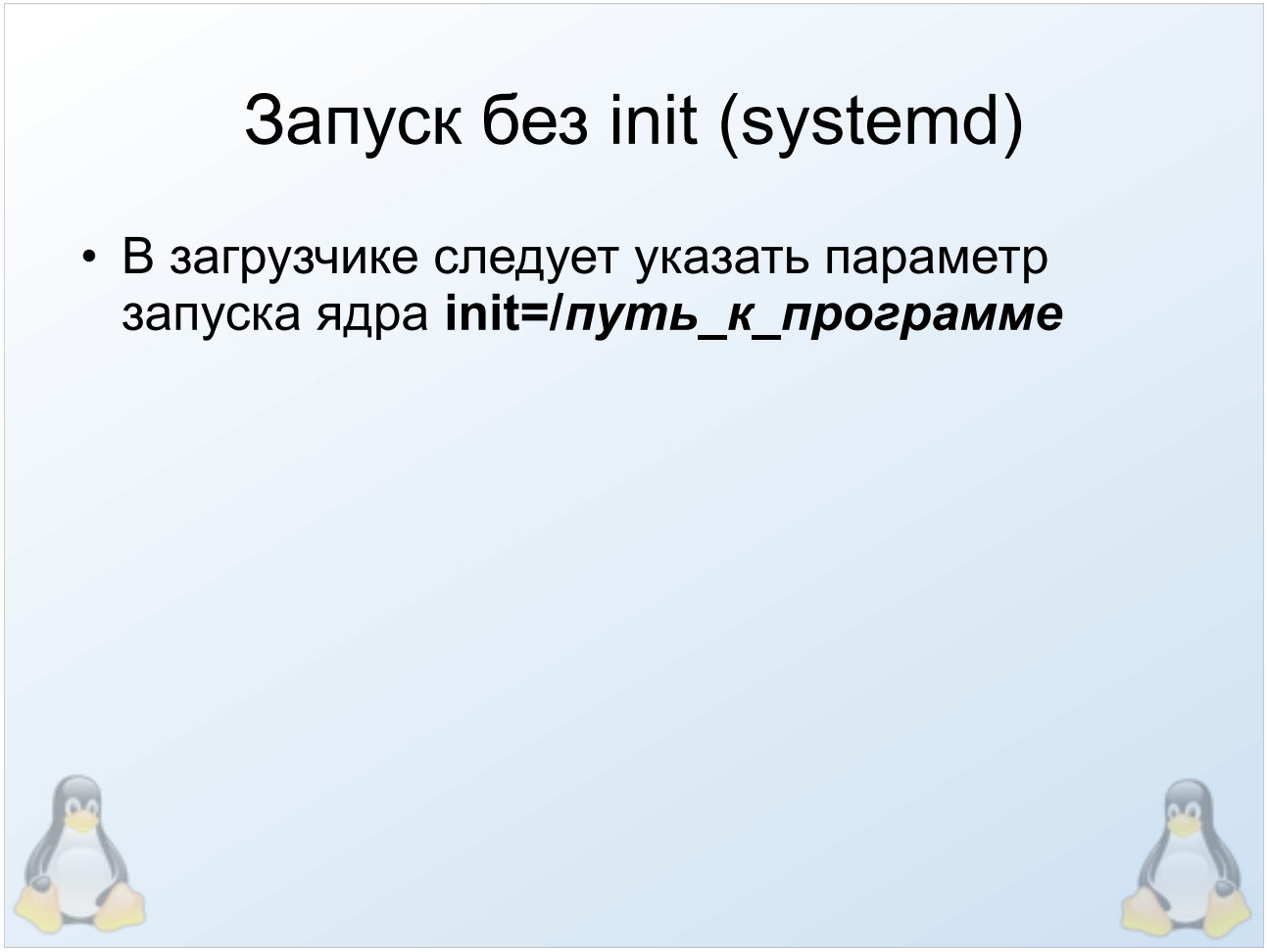
Пример: `Term::ReadKey` и `Term::ReadLine::GNU`, которые обеспечивают дополнение в командной строке по клавише табуляции и другие удобные возможности. Установка всех модулей производится командой `install Bundle::CPAN`.

21. Если вы не являетесь системным администратором, то для для установки двоичных файлов, модулей и документации для себя, вам необходимо добавить строку `PREFIX=/путь/куда/вы/имеете/право/записи` к параметру настройки `makepl_arg`. Все файлы будут установлены в каталоги, расположенные ниже каталога `PREFIX`, в не в системные каталоги. Для использования установленных таким образом модулей, вам необходимо будет добавить `PREFIX` в пути поиска файлов Perl.

22. Для получения дополнительной информации о модуле `CPAN`, его командах и параметрах настройки, смотрите `perldoc CPAN`.

Глава 8. Решение проблем

8.1. Запуск без использования /sbin/init



1. В GNU/Linux имеется возможность запустить ядро с опцией `init`, которая указывает какой процесс запустить в качестве процесса init с PID=1.
2. По умолчанию это программа `/sbin/init`.
3. Если нормальный запуск системы невозможен, вы можете использовать опцию `init` чтобы ядро сразу после старта запускало оболочку.

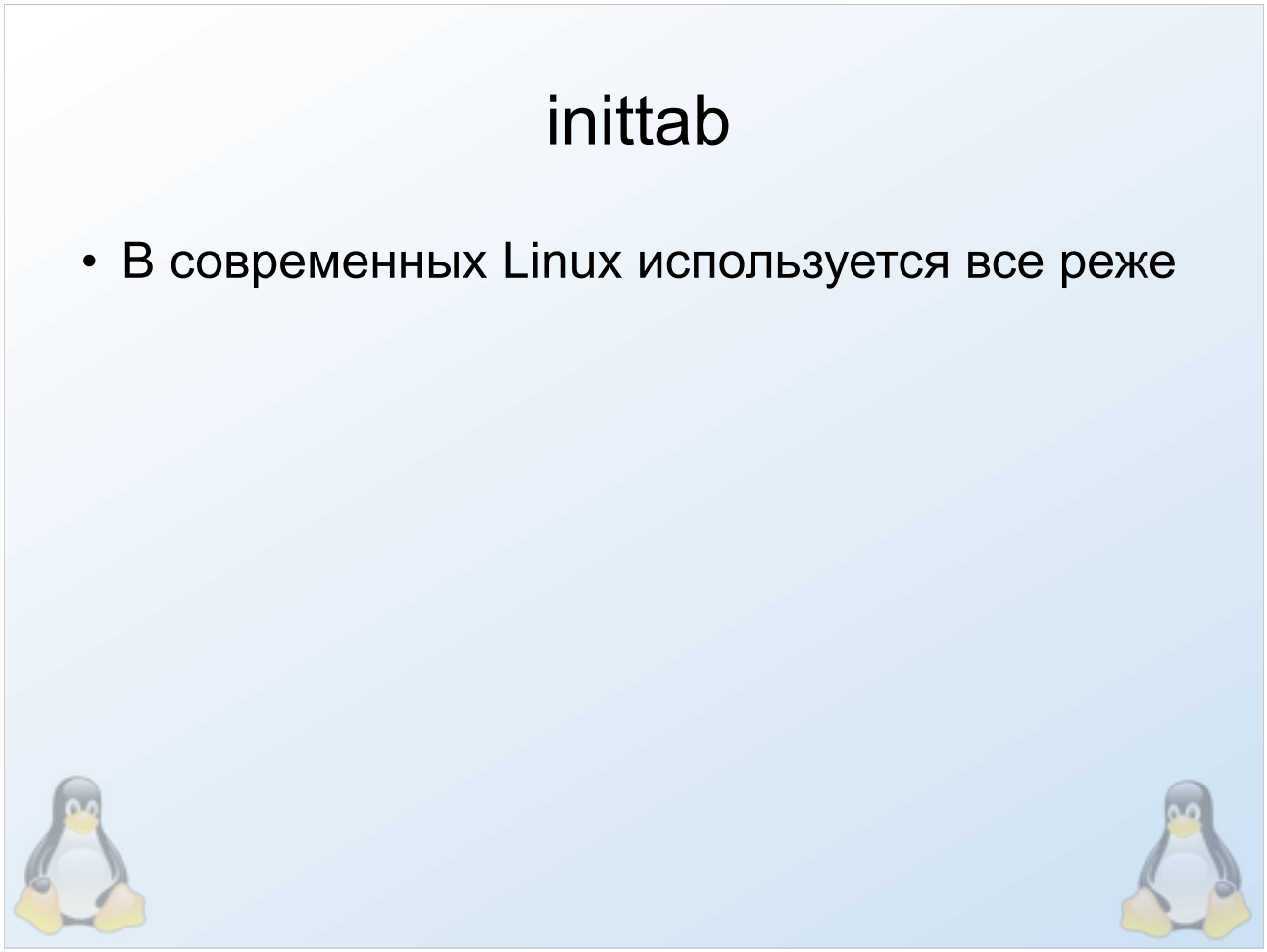
Пример:

```
linux init=/bin/bash
```

Примечание: В этом примере необходимо обеспечить доступ к разделу на котором находится bash.

4. При использовании данной опции вы получаете полный доступ к системе без проверки паролей, что является брешью в физической безопасности системы.

8.2. Настройка /etc/inittab



1. Для правильной загрузки системы очень важно иметь правильно настроенный файл `/etc/inittab`.
2. Основные настройки, на которые следует обратить внимание:
    1. Уровень выполнения загружаемый по умолчанию.
    2. Первый инициализационный скрипт.

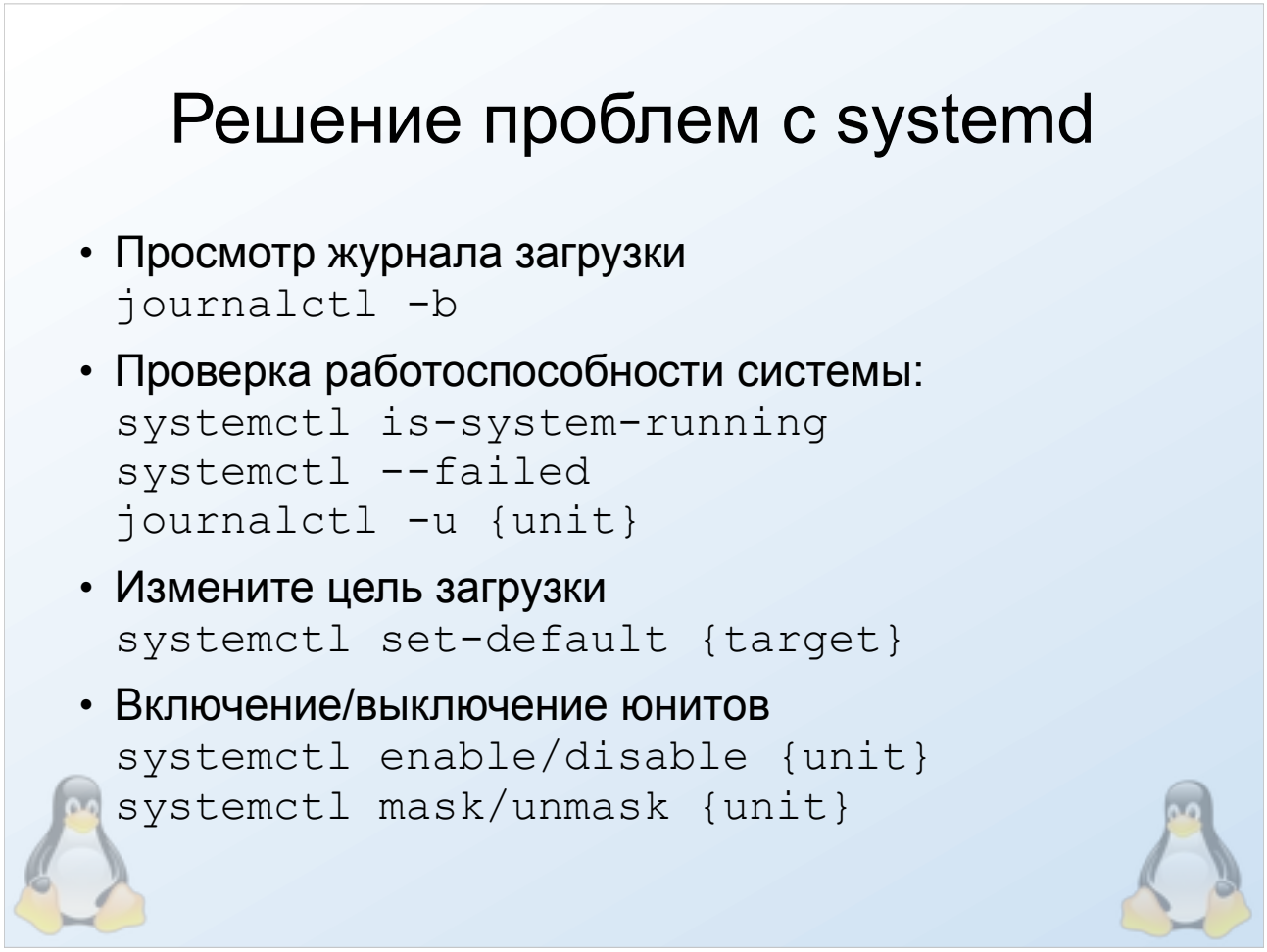
Примечание: скрипт в разных системах называется по разному и необходимо проверить, что он существует. Например, в RedHat Linux он называется /etc/rc.d/rc.sysinit.

    3. Запуск скрипта `rc` для необходимого уровня исполнения.
    4. Настройка терминалов на уровнях исполнения.

8.3. Инициализационные скрипты /etc/rc...

1. В правильном запуске системы очень важную роль играют скрипты `rc`.
2. Посредством запуска скрипта `rc` с аргументом номером уровня исполнения система осуществляет переход на заданный уровень исполнения. (см. `/etc/inittab`).
3. Одна из главных функций этого скрипта запуск скриптов в каталогах `rcN.d`, где `N` это уровень исполнения.
4. В каталогах `rcN.d` находятся символьные ссылки на скрипты запуска служб, которые надо:
   1. запустить – тогда сссылка начинается с символа S (start), а затем номер указывающий на порядок запуска службы;
   2. остановить – тогда ссылка начинается с символа K (kill), а затем номер указывающий на очередность остановки.
5. Для настройки каталогов `rcN.d` имеется удобная утилита `chkconfig`.

8.4. Настройка systemd



1. За инициализацию системы после запуска ядром первого процесса, в большинстве современных систем, отвечает демон systemd. Он также позволяет получить сведения о текущем состоянии или о процессе загрузки.
2. Просмотреть журнал текущей загрузки можно командой `journalctl -b`. Команда `journalctl --list-boots` покажет какие загрузки еще имеются в журнале `systemd`.
3. В systemd вы можете оценить общее состояние системы и быстро определить причину проблемы.

Пример:

```
# systemctl is-system-running
degraded
# systemctl --failed
  UNIT            LOAD   ACTIVE SUB    DESCRIPTION
● chronyd.service loaded failed failed NTP client/server

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
```

```
SUB    = The low-level unit activation state, values depend on unit type.


1 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
# journalctl -u chronyd.service | tail
мар 18 15:04:13 centos8-stream chronyd[848]: System clock was stepped by -
8.524354 seconds
мар 18 15:04:13 centos8-stream chronyd[848]: System clock TAI offset set to 37
seconds
мар 18 17:33:06 centos8-stream systemd[1]: Stopping NTP client/server...
мар 18 17:33:06 centos8-stream systemd[1]: chronyd.service: Succeeded.
мар 18 17:33:06 centos8-stream systemd[1]: Stopped NTP client/server.
мар 18 17:33:06 centos8-stream systemd[1]: Starting NTP client/server...
мар 18 17:33:06 centos8-stream chronyd[10275]: Invalid directive at line 1 in
file /etc/chrony.conf
мар 18 17:33:06 centos8-stream systemd[1]: chronyd.service: Control process
exited, code=exited status=1
мар 18 17:33:06 centos8-stream systemd[1]: chronyd.service: Failed with result
'exit-code'.
мар 18 17:33:06 centos8-stream systemd[1]: Failed to start NTP client/server.


# head -1 /etc/chrony.conf
ERROR IN CONFIG# Use public servers from the pool.ntp.org project.


# vi +1 /etc/chrony.conf
# systemctl start chronyd.service


# systemctl --failed
0 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.


# systemctl is-system-running
running
```

4. Изменение цели загрузки выполняется командой `systemctl set-default {target}`. Обычно используются цели multi-user или graphical.
5. Если не запустился или наоборот запустился, а не должен был, какой-то юнит, то настройка автоматического старта юнитов выполняется командой `systemctl enable/disable {unit}`. Помимо запрета на автоматический старт юнита можно установить полный запрет на старт юнита командой `systemctl mask {unit}`.

Пример: Разрешим запуск getty на последовательной линии (COM1)

```
# systemctl list-unit-files | grep serial
serial-getty@.service                          disabled
# systemctl list-units | grep serial-getty

# systemctl enable serial-getty@ttyS0
Created symlink /etc/systemd/system/getty.target.wants/serial-
getty@ttyS0.service → /usr/lib/systemd/system/serial-getty@.service.
# systemctl list-unit-files | grep serial
serial-getty@.service                          indirect
# systemctl list-units | grep serial-getty

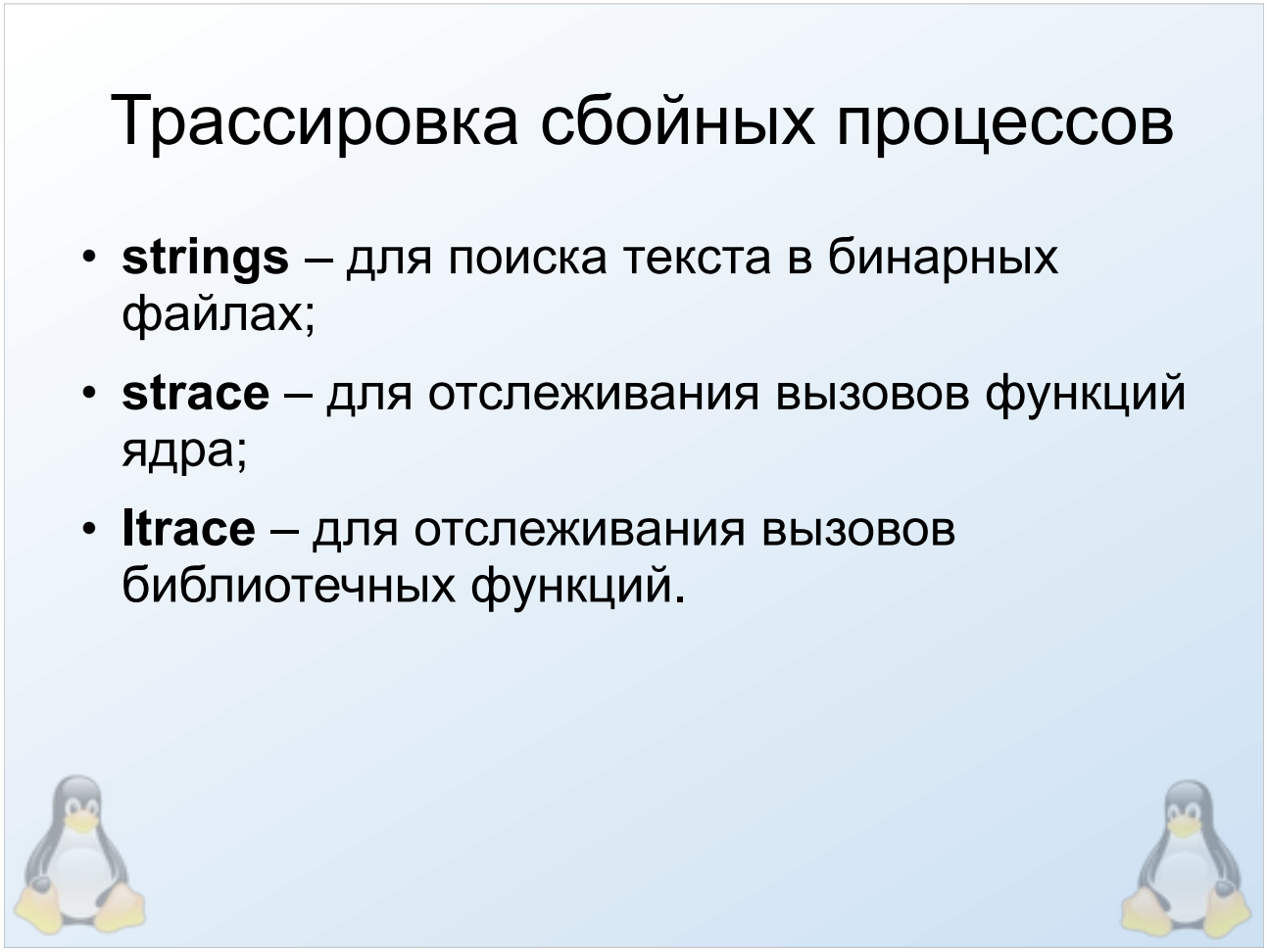
# systemctl status serial-getty@ttyS0 | head -5
● serial-getty@ttyS0.service - Serial Getty on ttyS0
   Loaded: loaded (/usr/lib/systemd/system/serial-getty@.service; enabled;
vendor preset: disabled)
   Active: inactive (dead)
     Docs: man:agetty(8)
           man:systemd-getty-generator(8)

# systemctl start serial-getty@ttyS0
# systemctl status serial-getty@ttyS0 | head -5
● serial-getty@ttyS0.service - Serial Getty on ttyS0
   Loaded: loaded (/usr/lib/systemd/system/serial-getty@.service; enabled;
vendor preset: disabled)
   Active: active (running) since Fri 2022-03-18 17:51:21 +05; 2s ago
     Docs: man:agetty(8)
           man:systemd-getty-generator(8)
# systemctl list-units | grep serial-getty
serial-getty@ttyS0.service
      loaded active running   Serial Getty on tty
```

8.5. Сообщения ядра dmesg

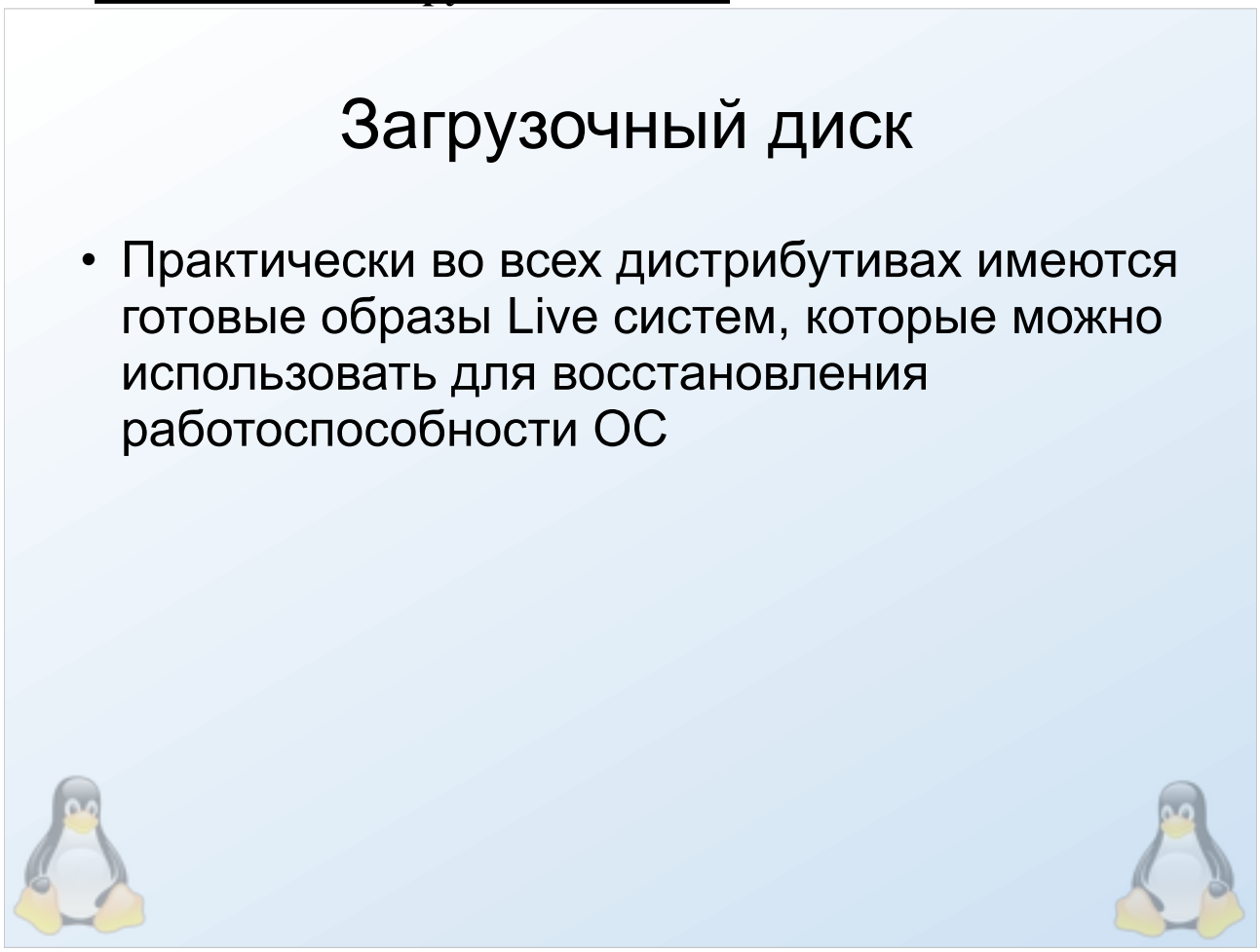
1. Утилита `dmesg` читает кольцевой буфер ядра. Кольцевой буфер – это буфер, в котором сообщения записываются по циклу. Новые сообщения затирают старые.
2. dmesg по умолчанию возвращает содержание всего буфера. Опция `-s` позволяет указать размер получаемых значений.
3. Размер буфера задается на этапе компилирования ядра. Если текущего размера не достаточно, то можно изменить параметр `LOG_BUF_LEN` в файле `kernel/printk.c`. Размер буфера должен быть кратен 8192.
4. Для очистки буфера применяется команда `dmesg -c`.
5. Команда `dmesg -n 1` отображает только сообщения уровня emergency.
6. Посмотреть сообщения ядра можно и командой `journalctl`: `journalctl -b -t kernel`.

8.6. Трассировка сбойных процессов



1. При проблемах с запуском программ можно использовать следующие утилиты:
    1. `strings` – для поиска текста в бинарных файлах;
    2. `strace` – для отслеживания вызовов функций ядра;
    3. `ltrace` – для отслеживания вызовов библиотечных функций.

8.7. Изготовление загрузочного диска

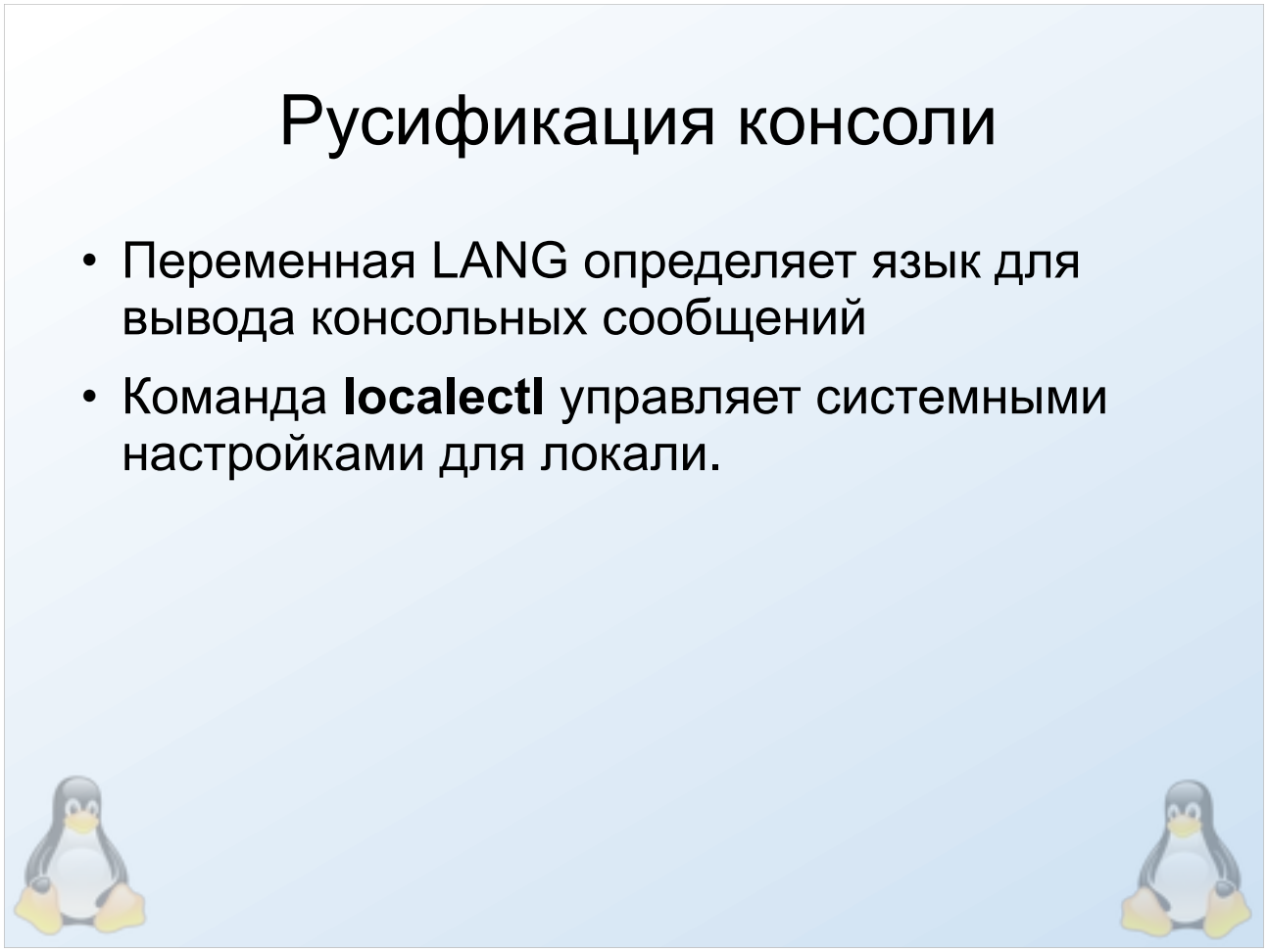


1. В Linux вы имеете возможность своими руками создать диск аварийного восстановления.
2. При создании диска (ISO образ) необходимо учитывать следующие моменты:
    1. Модули ядра. На диске нужно иметь все необходимые модули ядра, для работы с устройствами. Один из вариантов этого решения поместить модули на отдельный диск.
    2. Административные утилиты. Помимо простого наличия этих утилит на диске, необходимо, чтобы они были скомпилированы без использования динамических библиотек, иначе библиотеки вам также понадобятся на диске.
3. Во многих дистрибутивах имеется Live образ, которы можно использовать для восстановления системы.
4. Установочные CD-диски многих дистрибутивов, также имеют опцию восстановления системы.
5. Для создания загрузочных дисков имеется удобный скрипт `mkbootdisk`.
6. При создании дисков можно использовать следующие опции:
    1. `--iso` - создавать загрузочный iso образ;
    2. `--device устройство` - создать образ на указанном устройстве;
    3. `--kernelargs аргументы` - загружать ядро с указанными аргументами;

4. `--size размер` – размер диска в килобайтах.

7. Для работы `mkbootdisk` требуется пакет `syslinux`.

8.8. Русификация консоли



1. Выбор языка системы производится при установке системы или при помощи соответствующих конфигураторов: `yast` в Suse Linux или `setup` в Red Hat Linux.
2. В случае некорректного отображения русских символов нужно установить правильную локаль.
3. В Debian перед этим, возможно придется создать эту локаль.
4. В Red Hat все локали уже имеются нужно только включить правильный язык системы.
5. Управление локалью производится командой `localectl`.

Пример: Fedora

```
$ grep PRETTY_NAME /etc/os-release
PRETTY_NAME="Fedora Linux 36 (Workstation Edition)"

$ localectl
   System Locale: LANG=ru_RU.UTF-8
       VC Keymap: us
      X11 Layout: us,ru
```

```
     X11 Variant: ,typewriter
     X11 Options: grp:ctrl_shift_toggle,grp:win_space_toggle

$ localectl list-locales | head -3
C.UTF-8
aa_DJ.UTF-8
aa_ER.UTF-8

$ cat /etc/locale.conf
LANG=ru_RU.UTF-8

$ ls -ld /etc/
drwxr-xr-x. 1 root root 5318 мая 22 12:16 /etc/
```

Пример: Debian

```
$ grep PRETTY_NAME /etc/os-release
PRETTY_NAME="Debian GNU/Linux 11 (bullseye)"

$ localectl status
   System Locale: LANG=en_US.UTF-8
                  LANGUAGE=en_US:en
       VC Keymap: n/a
      X11 Layout: us
       X11 Model: pc105

$ ls -ld /etc/
drwxr-xr-x 125 root root 12288 May 22 13:09 /etc/

$ localectl set-locale ru_RU.UTF-8
==== AUTHENTICATING FOR org.freedesktop.locale1.set-locale ===
Authentication is required to set the system locale.
Authenticating as: admuser,,, (admuser)
Password:
==== AUTHENTICATION COMPLETE ===

$ grep -v ^# /etc/locale.gen


en_US.UTF-8 UTF-8
```

```
ru_RU.UTF-8 UTF-8

$ localectl list-locales
C.UTF-8
en_US.UTF-8
ru_RU.UTF-8

$ localectl
   System Locale: LANG=ru_RU.UTF-8
                  LANGUAGE=en_US:en
       VC Keymap: n/a
      X11 Layout: us
       X11 Model: pc105

$ localectl set-locale LANGUAGE=ru_RU:ru
==== AUTHENTICATING FOR org.freedesktop.locale1.set-locale ===
Authentication is required to set the system locale.
Authenticating as: admuser,,, (admuser)
Password:
==== AUTHENTICATION COMPLETE ===

$ localectl
   System Locale: LANG=ru_RU.UTF-8
                  LANGUAGE=ru_RU:ru
       VC Keymap: n/a
      X11 Layout: us
       X11 Model: pc105

$ ls -ld /etc/
drwxr-xr-x 125 root root 12288 May 22 13:09 /etc/
```

Примечание: Необходимо еще перелогиниться, чтобы изменения вступили в силу. После выхода и входа получаем:

```
$ ls -ld /etc/
drwxr-xr-x 125 root root 12288 мая 22 13:09 /etc/
$ localectl
   System Locale: LANG=ru_RU.UTF-8
                  LANGUAGE=ru_RU:ru
       VC Keymap: n/a
      X11 Layout: us
       X11 Model: pc105
```

Пример: изменение языка для одной команды:

```
$ localectl list-locales
C.UTF-8
en_US.UTF-8
ru_RU.UTF-8
$ LANG=C bash -c wrongcomm
bash: line 1: wrongcomm: command not found
$ bash -c wrongcomm
bash: строка 1: wrongcomm: команда не найдена
 LANG=fi_FI.UTF-8 bash -c wrongcomm
bash: line 1: wrongcomm: command not found
admuser@debian11:~$ LANG=uz_UZ.UTF-8 bash -c wrongcomm
bash: line 1: wrongcomm: command not found
```

Примечание: В двух последних примерах локали для языков не установлены, поэтому отображение идет в локсли С. Иная ситуация в Fedora Linux

```
$ localectl list-locales | egrep '^(fi|uz)'
fi_FI.UTF-8
fil_PH.UTF-8
uz_UZ.UTF-8
uz_UZ.UTF-8@cyrillic

$ LANG=fi_FI.UTF-8 bash -c wrongcomm
bash: rivi 1: wrongcomm: komentoa ei löydy

$ LANG='uz_UZ.UTF-8@cyrillic' bash -c wrongcomm
bash: line 1: wrongcomm: command not found

$ LANG='uz_UZ.UTF-8@cyrillic' ls -ld /etc
drwxr-xr-x. 1 root root 5318 Май 22 12:16 /etc

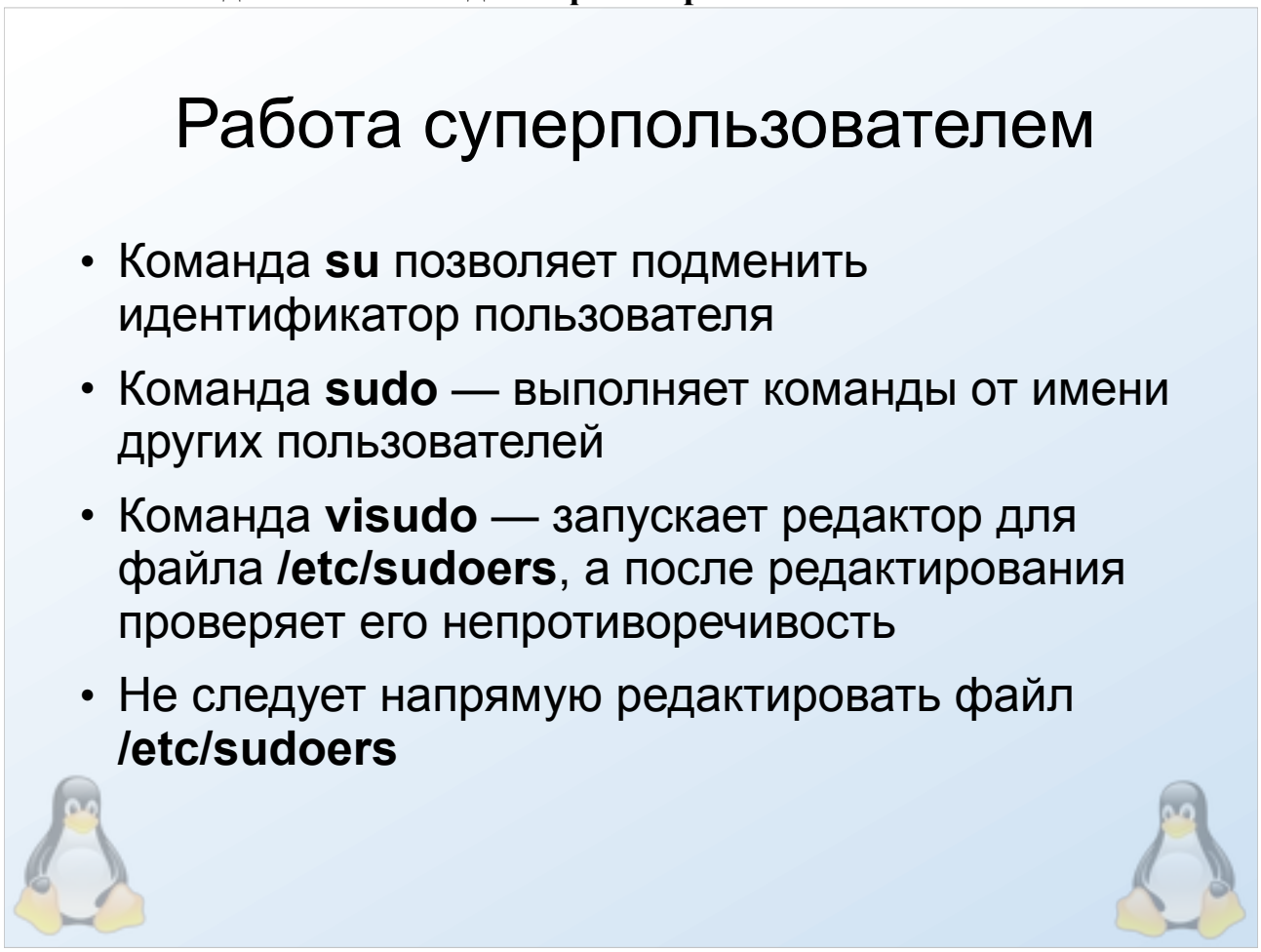
$ LANG='uz_UZ.UTF-8' ls -ld /etc
drwxr-xr-x. 1 root root 5318 May 22 12:16 /etc
```

Примечание: как показала проверка поддержка узбекского языка минимальна.

6. При монтировании диска, в котором имена файлов сохранены в кодовой странице 866 необходимо явно указывать номер кодовой страницы, поскольку по умолчанию монтирование производится с кодовой страницей 437.
7. На этапе конфигурирования ядра можно также задать опции по кодовой странице по умолчанию – `Native Language Support`. В меню `File System`.
8. Обратите внимание на переменную `LANG`. Обычно ее значение должно быть примерно таким ru, ru_RU, ru_RU.UTF-8 и т.д.

8.9. Работа в качестве суперпользователя

8.9.1. Команда su: замена идентификатора пользователя



1. Поскольку суперпользователь является таким же членом системы, как и остальные пользователи, можно войти в систему непосредственно под именем root. Однако оказывается, что это довольно неудачное решение.
    1. Во-первых, не будет сделано никаких записей о том, какие операции выполнял суперпользователь.
    2. Во-вторых, сценарий регистрации суперпользователя не предполагает сбора дополнительной идентифицирующей информации. Когда под именем root в систему могут входить несколько пользователей, не существует способа определить, кто именно из них и когда это делал.
2. Вследствие упомянутых причин рекомендуется регистрацию под именем root запрещать на терминалах и по сети, т.е. везде, кроме системной консоли.
3. С точки зрения безопасности лучше получать доступ к учетной записи root с помощью команды `su`.

Примечание: Будучи вызванной без аргументов, она выдает приглашение на ввод пароля суперпользователя, а затем запускает интерпретатор команд с правами пользователя root.

Интерпретатор будет выполняться в привилегированном режиме, пока не завершит работу (по команде exit или при нажатии клавиш <CTRL+D>). Команда su не фиксирует действия, производимые в среде интерпретатора, но добавляет запись в журнальный файл с указанием, кто и когда вошел в систему под паролем суперпользователя.

4. Команда `su` способна также подставлять вместо имени root имена других пользователей.
5. Рекомендуется взять за правило при вводе команды указывать полное имя, например `/bin/su`, а не просто `su`. Это послужит определенной защитой от тех программ с именем `su`, которые преднамеренно были прописаны в переменной среды `PATH` злоумышленником.

8.9.2. Утилита sudo: ограниченный вариант команды su

1. Использование утилиты `sudo` имеет следующие преимущества:
    1. благодаря регистрации команд значительно повышается степень административного контроля над системой,
    2. операторы могут выполнять специальные задачи, не имея неограниченных привилегий;
    3. настоящий пароль суперпользователя могут знать всего один-два человека;
    4. вызывать утилиту sudo быстрее, чем выполнять команду su или входить в систему под именем root,
    5. пользователя можно лишить привилегий, не меняя пароль суперпользователя
    6. ведется список всех пользователей с правами пользователя root;
    7. меньше вероятность того, что интерпретатор команд, запущенный суперпользователем, будет оставлен без присмотра;
    8. управлять доступом ко всей сети можно с помощью одного файла.
2. Утилита имеет `sudo` и недостатки. Самый большой из них заключается в том, что любая брешь в системе защиты того или иного привилегированного пользователя эквивалентна нарушению безопасности самой учетной записи root.
3. Утилита `sudo` в качестве аргумента принимает командную строку, которая подлежит выполнению от имени пользователя `root` (или другого уполномоченного пользователя).
4. Утилита обращается к файлу `/etc/sudoers`, где содержится список пользователей, имеющих разрешение на ее выполнение, и перечень команд, которые они могут вводить на конкретном компьютере. Если запрашиваемая команда разрешена, утилита `sudo` приглашает пользователя ввести его собственный пароль и выполняет команду от имени суперпользователя.
5. Далее утилита `sudo` позволяет, не вводя пароль, выполнять другие команды, но только до тех пор, пока не наступит пятиминутный период бездействия (его продолжительность можно менять).
6. Утилита `sudo` ведет журнал, где регистрируются выполненные команды и вызвавшие их пользователи, а также каталоги, из которых запускались команды, и время их вызова. Эта информация может направляться в систему Syslog или сохраняться в любом журнальном файле по усмотрению пользователя.

Пример: Строка журнального файла, содержащая данные о пользователе `user1`, который выполнил команду `sudo /bin/cat /etc/sudoers`, может выглядеть следующим образом:

```
Dec 7 10:47:23 comp1 sudo: user1: TTY=ttyp0 ; PWD=/comp1/users/user1; USER=root;
COMMAND=/bin/cat /etc/sudoers
```

7. В файле `/etc/sudoers` можно указать псевдонимы:
    1. `Host_Alias` – для компьютеров с которых можно запускать команды;
    2. `Cmnd_Alias` – для определения команд;
    3. `User_Alias` – для определения групп пользователей;
    4. `Runas_Alias` – для определения пользователей от имени которых запускаются команды.
8. В каждую спецификацию прав доступа включается информация о:
    1. пользователях, к которым относится запись;
    2. компьютерах, на которых пользователям разрешено выполнять соответствующие действия;
    3. командах, которые могут выполняться указанными пользователями;
    4. пользователях, от имени которых могут выполняться команды.

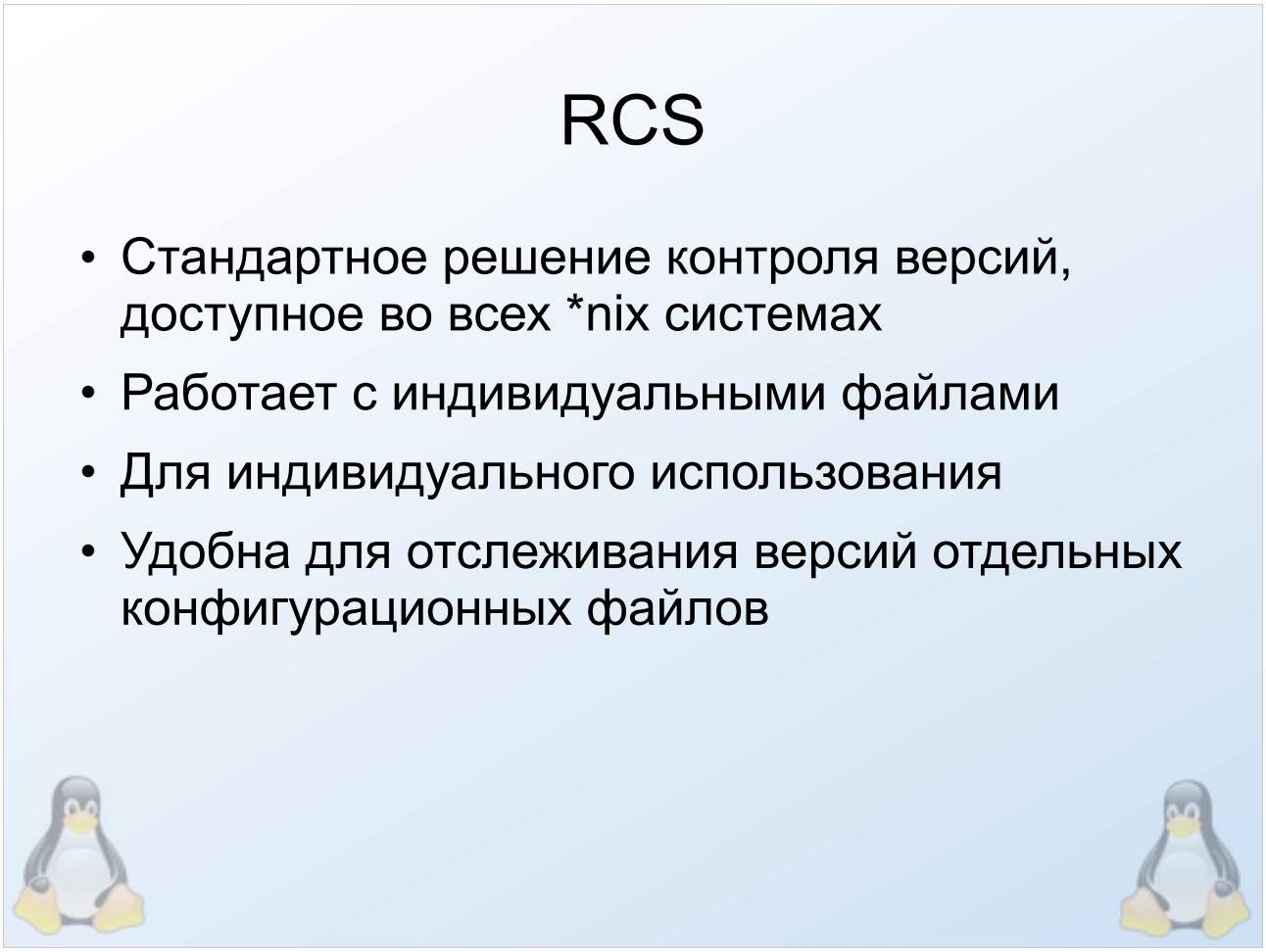
Пример: строка в `sudoerrs`, позволяющая пользователям группы users выключать компьютер

```
%users  localhost=/sbin/shutdown -h now
```

9. Для модификации файла `/etc/sudoers` предназначена специальная команда `visudo`, которая проверяет, не редактируется ли файл кем-то посторонним, затем открывает его в редакторе, а перед инсталляцией файла выполняет синтаксический контроль. Последний этап особенно важен, поскольку ошибка в файле `/etc/sudoers` может не позволить повторно вызвать утилиту `sudo` для исправления файла.

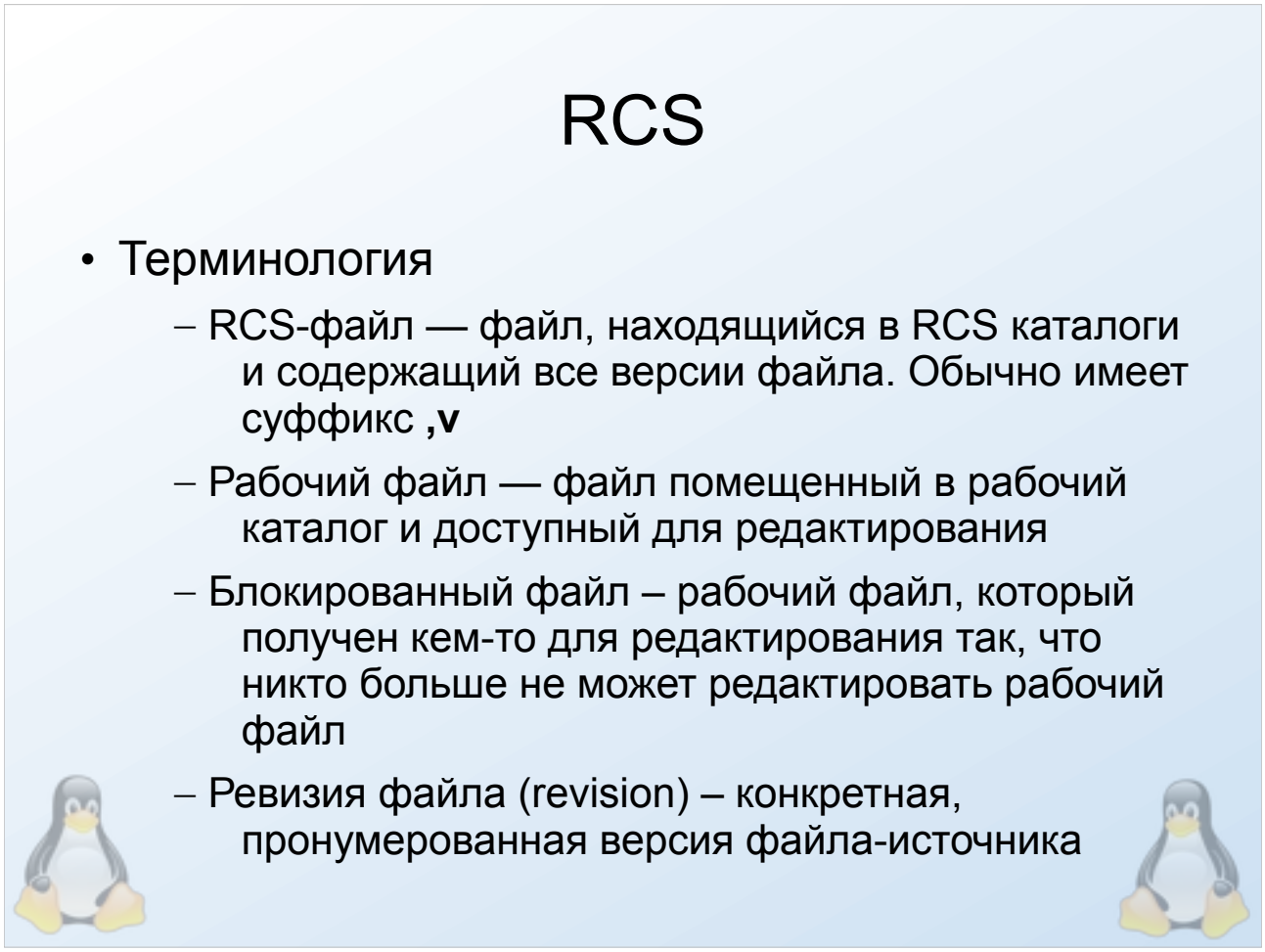
Глава 9. Инструменты поддержки версий

9.1. Система RCS



1. Системы управления версиями позволяют автоматизировать процесс, который отслеживает и управляет изменениями, сделанными в текстовых файлах.
2. Системы управления версиями рекомендуются использовать не только программистам, но и администраторам и всем, кому нужно регулярно редактировать текстовые файлы.
3. RCS (Revision Control System) – стандартное решение для контроля версий, доступное во всех версиях UNIX и LINUX.
4. Существует две альтернативные RCS системы: SCCS (Source Code Control System) и CVS (Concurrent Version System).
5. CVS – надстройка над RCS, поэтому невозможно использовать CVS без знания RCS
6. CVS имеет две важных дополнительных черты по сравнению с RCS:
    1. CVS умеет лучше работать с иерархией каталогов, поэтому он более расчитан на поддержку проектов. RCS – файл-ориентированная система.
    2. CVS – многопользовательская распределенная система, может поддерживать работу большого количества разработчиков, которые расположены в разных географических

местах.

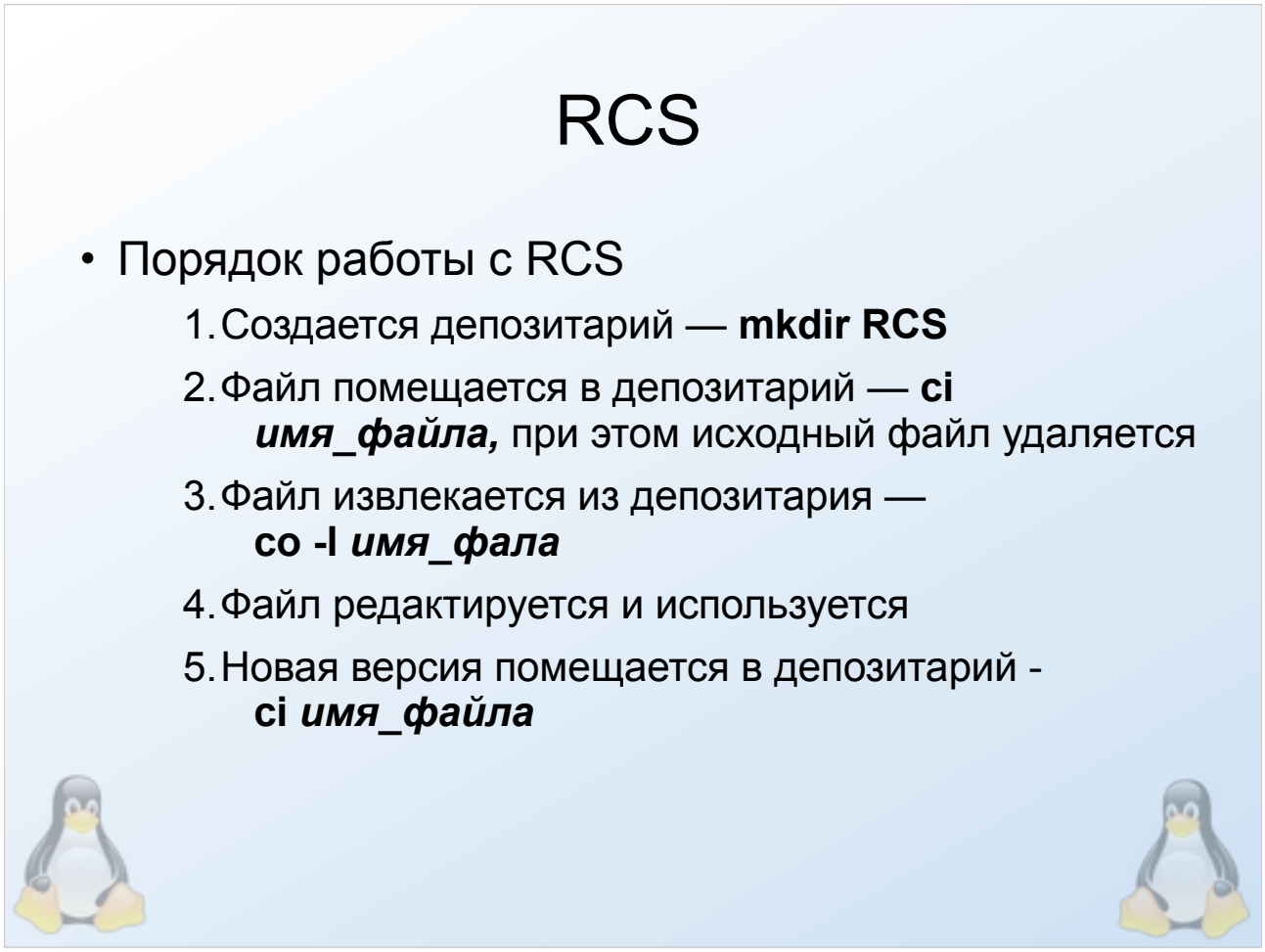


7. RCS–файл – любой файл, который размещается в RCS каталоге, доступ к которому можно получить через RCS систему. RCS файлы содержат все версии файла-оригинала. Обычно, RCS-файл имеет суффикс “,v”
8. Рабочий файл (working file) – файл, который из RCS-каталога (RCS-депозитария) помещен (опубликован) в текущий рабочий каталог и доступен для дальнейшего редактирования.
9. Блокированный файл (lock) – рабочий файл, который получен кем-то для редактирования так, что никто больше не может редактировать рабочий файл.
10. Ревизия файла (revision) – конкретная, пронумерованная версия файла-источника. Версии начинаются с 1.1 и последовательно увеличиваются.
11. Версию RCS можно узнать командой

```
rcs -V
```

12. RCS поддерживает три основные команды, осуществляющие управление версиями:

```
ci (check in)
co (check out)
ident
```



13. Перед тем, как начать работать с поддержкой версий, необходимо создать в каталоге, где будут располагаться файлы-источники, подкаталог RCS

```
mkdir RCS
```

14. Каталог RCS – депозитарий файлов-версий, именно в нем будут содержатся ревизии.

15. Перед тем, как редактировать файл, нужно воспользоваться командой:

```
ci имя_файла_источника
```

Примечание: которая сначала переместит исходный файл в депозитарий, а затем удалит файл-источник. Команда ci либо создает ревизию, либо добавляет существующий файл в ревизию.

Пример:

```
$ cat hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               linux1 localhost.localdomain localhost
$ ls
RCS        hosts
$ ci hosts
RCS/hosts,v  <--  hosts
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
```

```
>> Example of RCS usage
>> .
initial revision: 1.1
done
$ ls
RCS
$ ls RCS
```

hosts,v

16.Для того, чтобы редактировать файл, нужно использовать команду

```
co -l имя_файла_источника
```

Примечание: Команда со позволяет получить файл из репозитария, без опции файл получается для чтения. Если нужно редактировать файл, то следует использовать опцию -l, которая блокирует этот файл.

Пример:

```
$ co -l hosts
RCS/hosts,v  -->  hosts
revision 1.1 (locked)
done
$ ls . RCS
.:
RCS             hosts

RCS:
hosts,v
$ vi hosts   (далее редактируем файл)
$ cat hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
```

127.0.0.1 linux2 localhost.localdomain localhost

17.После окончания редактирования файла нужно создать новую ревизию командой `ci`. Если нужно оставить рабочий файл (без опций он автоматически удаляется), можно воспользоваться опциями либо `-l`, либо `-u`.

1. Опция `-l` создает новую ревизию, но оставляет рабочий файл, при этом блокирует его и делает доступным для дальнейшего редактирования.
2. Опция `-u` оставляет рабочий файл, не блокируя его.

Пример:

```
$ ci hosts
RCS/hosts,v  <--  hosts
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
```

```
>> End of line is added
>> .
done
$ ls
RCS
$ co -l hosts
RCS/hosts,v  -->  hosts
revision 1.2 (locked)
writable hosts exists; remove it? [ny](n): y
done
$ ls
RCS             hosts
```

18.Команда `rcsdiff` позволяет увидеть какие изменения были осуществлены:

```
rcsdiff имя_рабочего_файла.
```

Примечание: Команда rcsdiff по умолчанию сравнивает рабочий файл с последней его ревизией

Пример:

```
$ rcsdiff -r1.1 hosts
===================================================================
RCS file: RCS/hosts,v
retrieving revision 1.1
diff -r1.1 hosts
3c3
< 127.0.0.1           linux1 localhost.localdomain localhost
---
> 127.0.0.1           linux2 localhost.localdomain localhost
```

19.Команда `rlog` выводит информацию о файлах, которые хранятся в депозитарии.

20.Команда `rcsclean` удаляет все рабочие файлы, которые не изменились после того, как была создана их ревизия.

21.Многие команды (`ci, co, rcsdiff`, ...) принимают такую важную опцию, как `-r`. С

помощью этой опции указывается номер ревизии.

22. RCS поддерживает специальные макросы, которые позволяют вводит идентифицирующую информацию (дату версии, имя исходного файла, имя пользователя-редактора и т.д.) в файлы с исходным текстом, объектным и бинарным текстом. Такие макросы называются ключевыми словами RCS.

23. Ключевые слова вставляются в рабочие файлы, так

```
$ключевое_слово$
```

Пример:

```
$cat hosts
# $Id$
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               linux2 localhost.localdomain localhost
```

24. После создания ревизии (командой ci) и получения рабочей версии (командой co) RCS заменяет ключевое слово его значением.

Пример:

```
$cat hosts
# $Id: hosts,v 1.4 2005/03/05 19:37:13 root Exp root $
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               linux1 localhost.localdomain localhost
```

25. Ключевое слово `$Id$` позволяет в рабочие файлы подставлять имя файла, номер ревизии, дату и время ревизии, автора, состояние, имя пользователя, кто заблокировал файл.

26. Ключевое слово `$Log$` подставляет в рабочий файл журнальные сообщения RCS.

27. Можно использовать ключевые слова для вставки идентифицирующих строк в скомпилированный код. Для этого можно присвоить глобальной переменной данное ключевое слово, а затем выполнить команды ci и co для подстановки значения ключевого слова. После компиляции исходного текста идентификационные строки можно изъять командой

```
ident имя_бинарного_файла
```

Пример:

```
$cat getpid.c
/* $Id: getpid.c,v 1.3 2003/01/15 16:09:22 svt Exp svt $
*/
#include <stdio.h>
#include <unistd.h>

static char rcsid[] = "$Id$\n";
```

```
int main (void)
{
        printf("PID=%d\n", getpid());
        return 0;
$ci getpid.c
$co -l getpid.c
RCS/getpid.c,v  -->  getpid.c
revision 1.4 (locked)
done
$cat getpid.c
/* $Id: getpid.c,v 1.4 2003/01/15 17:01:56 svt Exp svt $
*/
#include <stdio.h>
#include <unistd.h>

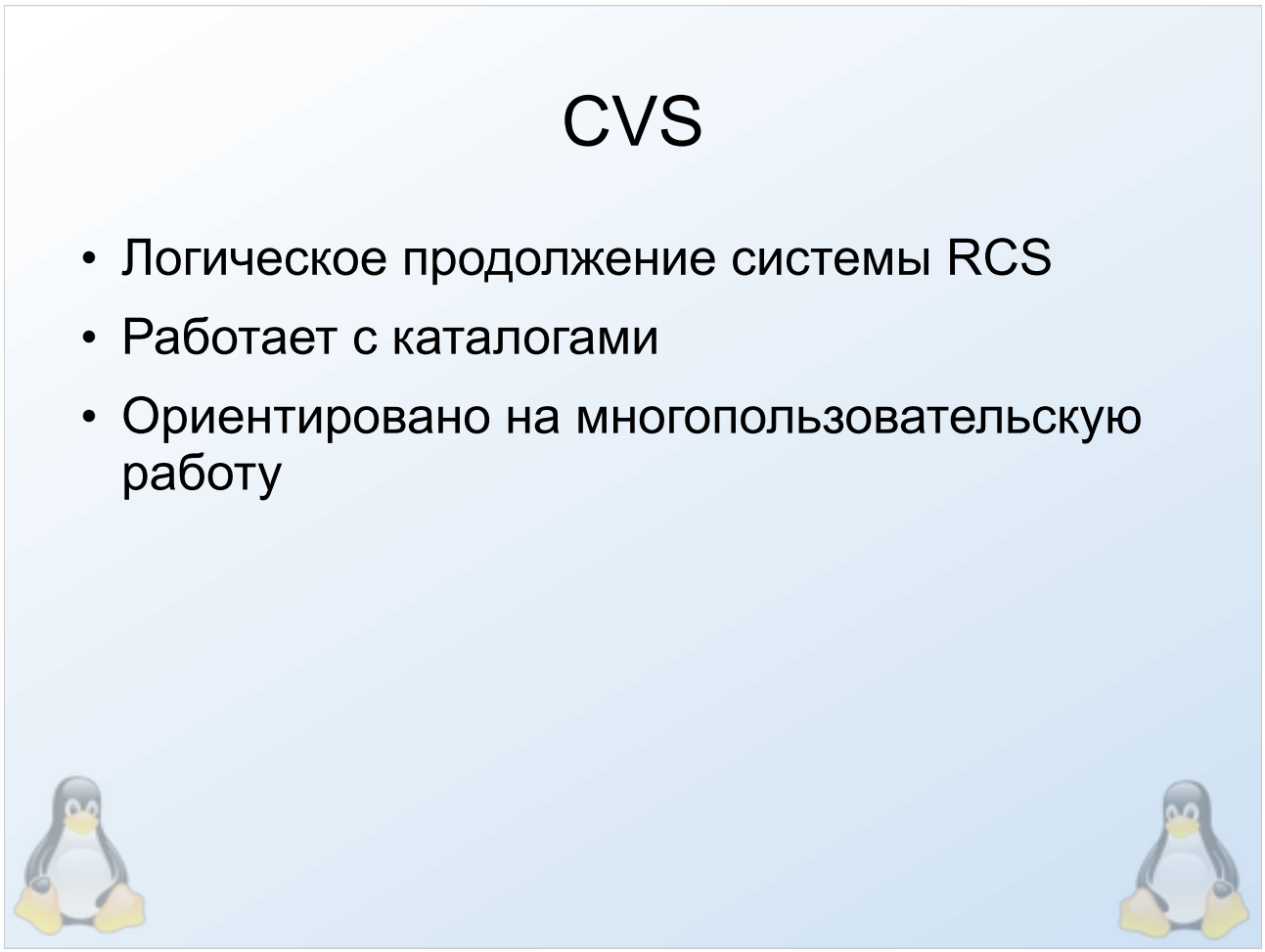
static char rcsid[] = "$Id: getpid.c,v 1.4 2003/01/15 17:01:56 svt Exp svt $\n"

int main (void)
{
        printf("PID=%d\n", getpid());
        return 0;
}
$gcc -o pid getpid.c
$ident pid
pid:
     $Id: getpid.c,v 1.4 2003/01/15 17:01:56 svt Exp svt $
     $FreeBSD: projects/trustedbsd/mac/lib/csu/i386-elf/crt1.c,v 1.3 2002/04/16
16:24:35 peter Exp $
```

28. Графическая логическая схема использования RCS

9.2. Система CVS

9.2.1. Репозиторий



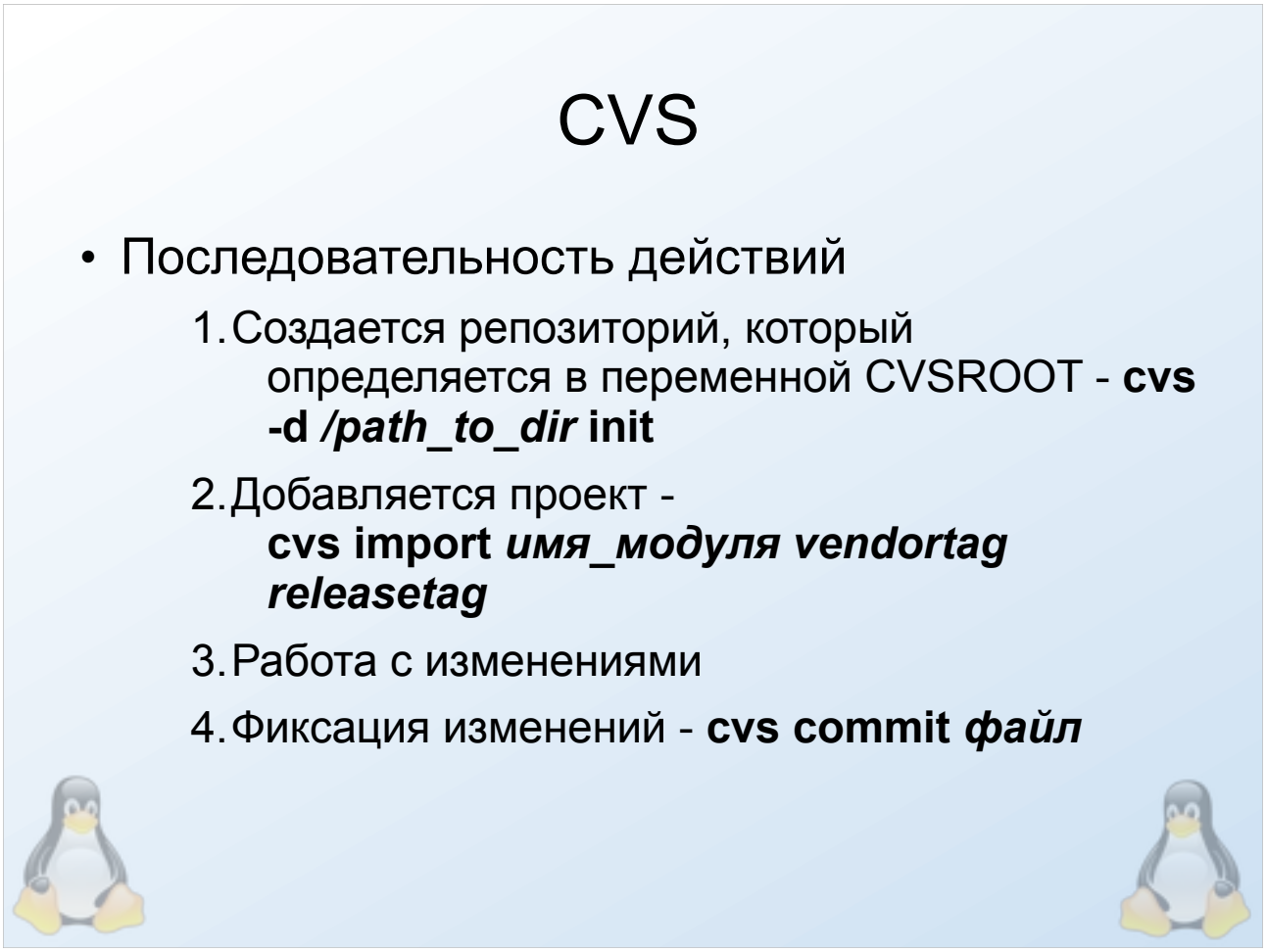
1. CVS поддерживает историю дерева каталогов, работая с последовательностью изменений.
2. CVS маркирует каждое изменение моментом времени, когда оно было сделано, и именем пользователя, совершившим изменение. Обычно человек, совершивший изменение, также предоставляет текстовое описание причины, по которой произошло изменение.
3. Вооружившись всей этой информацией, CVS может отвечать на такие вопросы, как
   1. Кто совершил данное изменение?
   2. Когда они его совершили?
   3. Зачем они это сделали?
   4. Какие еще изменения произошли в то же самое время?
4. CVS хранит все изменения в данном проекте в дереве каталогов, называемом **репозиторием** (`repository`).
5. Перед тем, как начать использовать CVS, вам необходимо настроить переменную среды `CVSROOT` так, чтобы она указывала на каталог репозитория.

6. Либо явно указывать где находится репозиторий опция `-d` команды `cvs`.
7. CVS не может работать в обычном дереве каталогов; наоборот, вы должны работать в каталоге, который CVS создаст для вас.
8. Репозиторий должен находится на машине, где достаточно свободного дискового пространства для того, чтобы хранить все файлы и данные обо всех предполагаемых изменениях.
9. Обеспечьте, чтобы все пользователи со всех используемых компьютеров имели доступ к машине, на которой расположен архив.
10. Создайте корневую директорию архива . Чаще всего архивы хранятся в `/home/cvsroot` или `/usr/local/cvsroot`.
11. Используйте `cvs init`, чтобы настроить директорию в качестве CVS архива.

```
cvs -d /home/cvsroot init
```

Примечание: В Debian Linux есть скрипт cvs-makerepos, который создаст архив на основе предыдущих скриптов настройки Debian. Для получения дополнительной информации ознакомьтесь с инструкцией по cvs-makerepos (man cvs-makerepos) и инструкцией по cvsconfig (man cvsconfig) для автоматизированных систем для настройки CVS-архива в Debian.

9.2.2. Добавление проекта



1. Прежде чем загружать свой проект в CVS, продумайте его структуру.
2. Перемещение или переименование файлов и директорий может повредить записи обо всех изменениях в файлах CVS. Удаление директории может стоить вам записей обо всех ее файлах и поддиректорий. По этой причине CVS не имеет возможностей для удаления или переименования файлов и директорий, равно как и удаления директорий.
3. Создайте исходную структуру ваших директорий – даже если пока это всего одна директория.
4. Добавьте любые желаемые исходные файлы.
5. Запустите в корневой директории вашего проекта команду

```
cvs -d путь_к_архиву_cvs import имя_модуля vendortag releasetag.
```

Примечание: В большинстве случаев вам не обязательно знать о vendor tags и release tags. CVS требует их наличие, но вы можете просто использовать имя модуля в качестве vendor tag и название текущей версии в качестве release tag.

Пример:

```
$ cd example
$ cvs -d /home/cvsroot import example example_project ver_0-1
```

6. Тэги присваивают символическое имя измененной версии файла или последовательности

файлов.

7. Команда `cvs tag` ставит метки на все архивные версии всех файлов в текущей рабочей директории и ее поддиректориях.
8. Команда `cvs rtag` основана на временной метке. Она присваивает символическое имя той версии файла или директории, которая является ближайшей из более старых, независимо от рабочей директории.
9. CVS не допускает наличия в тэгах символа '.'.

```
cvs tag имя_тэга имя_файла
```

```
cvs tag имя_тэга
```

Примечание: Отмечает архивные версии всех файлов текущей рабочей директории.

```
cvs tag -c имя_тэга
```

Примечание: Прерывается, если архивная копия отличается от рабочей копии.

Пример:

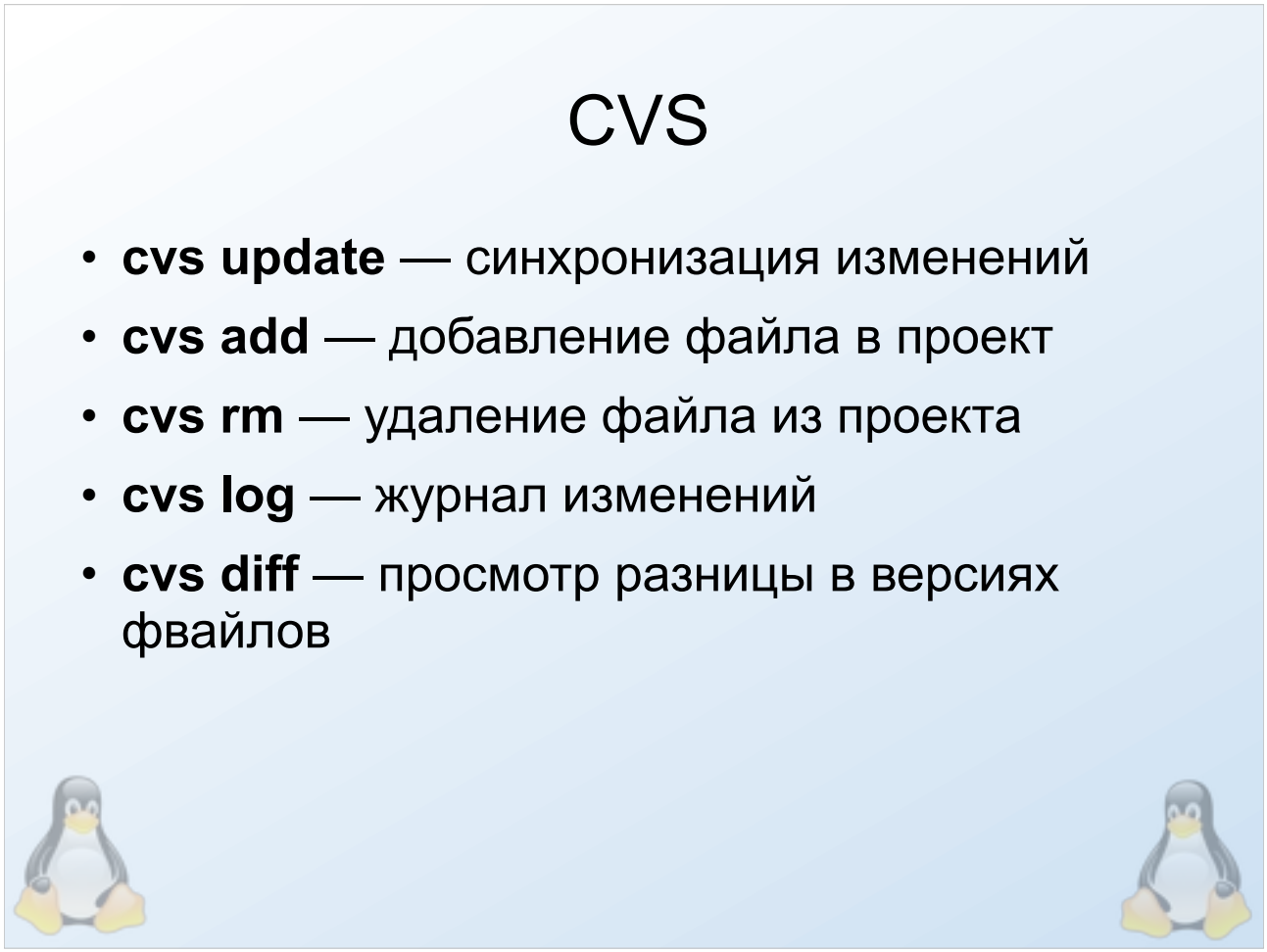
```
cvs/example$ cvs tag release-1-0 src/sample.c
```

```
cvs/example/src$ cvs tag release-1-0
```

```
cvs/example/src$ cvs tag -c release-1-0
```

10. Команда `cvs checkout каталог`, позволяет получить от CVS рабочее дерево каталогов, которая означает извлечь дерево исходных текстов с именем "`каталог`" из репозитория, указанного в переменной окружения "`CVSROOT`".
11. После того, как CVS создал рабочее дерево каталогов, вы можете обычным образом редактировать, компилировать и проверять находящиеся в нем файлы - это просто файлы.

9.2.3. Добавление и удаление файлов в проекте



1. CVS обращается с добавлением и удалением файлов так же, как и с прочими изменениями, записывая такие события в истории файлов.

Примечание: Можно смотреть на это так, как будто CVS сохраняет историю каталогов вместе с историей файлов.

2. CVS не считает, что созданные в рабочем каталоге файлы должны оказаться под его контролем; это не так во многих случаях.
3. Когда вы создадите новый файл, `cvs update` маркирует этот файл флагом `` `?' ``, пока вы не скажете CVS, что именно вы намереваетесь сделать с этим файлом.
4. Чтобы добавить файл в проект, сначала вы должны создать его, затем использовать команду `cvs add`, чтобы маркировать его как добавленный. Затем при следующем выполнении команды `cvs commit` CVS добавит этот файл в репозиторий.
5. CVS обращается с удаленными файлами почти так же. Если вы удалите файл и выполните `` `cvs update' ``, CVS не считает, что вы намереваетесь удалить файл из проекта. Вместо этого он восстанавливает последнюю сохраненную в репозитории версию файла и маркирует его флагом `U`, точно так же, как и любой другое обновление.

Примечание: Это означает, что если вы хотите отменить изменения файла в рабочем каталоге, вы можете просто удалить его и позволить команде `` `cvs update' `` создать его заново.

6. Чтобы удалить файл из проекта, вы должны сначала удалить его, а потом использовать

команду `cvs rm', чтобы пометить его для удаления. При следующем запуске команда `cvs commit' удалит файл из репозитория.

7. Фиксирование файла, маркированного с помощью `cvs rm' не уничтожает историю этого файла - к ней просто добавляется еще одна редакция - "не существует".
8. Для переименования файла существует несколько стратегий; самая простая - переименовать файл в рабочем каталоге, затем выполнить `cvs rm' со старым именем и `cvs add' с новым. Недостаток этого подхода в том, что журнальные записи о содержимом старого файла не переносятся в новый файл.
9. Вы можете добавлять каталоги точно так же, как и обычные файлы.

9.2.4. Работа с изменениями

1. Так как каждый использует свой собственный рабочий каталог, изменения, которые вы делаете в своем каталоге, не становятся автоматически видимыми всем остальным в вашей команде. CVS не публикует изменений, пока они не закончены.
2. Когда вы протестируете изменения, вы должны **зафиксировать** (commit) их в репозитории и сделать их доступными остальным.
3. Перед тем, как фиксировать ваши изменения, CVS требует, чтобы файлы были синхронизированы со всеми изменениями, которые сделали остальные члены группы.
4. Команда cvs update позаботится об этом.
5. В выводе cvs update строки вида `U файл' означают, что файл просто был обновлен; кто-то еще внес в этот файл изменения, и CVS скопировал измененный файл в ваш рабочий каталог.
6. CVS объединяет изменения только в вашей рабочей копии; репозиторий и рабочие каталоги других разработчиков остаются нетронутыми.
7. Строка вида `M файл' означает, что файл был модифицирован вами и содержит изменения, которые еще не стали видны другим разработчикам. Это изменения, которые вам следует зафиксировать.
8. Вы можете зафиксировать их так:

```
$cvs commit файл
```

Примечание: В этом месте CVS запустит текстовый редактор и попросит вас ввести описание изменений. После того, как вы выйдете из редактора, CVS зафиксирует ваши изменения. Заметьте, что когда вы зафиксировали ваши изменения и они видны всем остальным членам группы. Когда другой пользователь исполняет cvs update, CVS внесет ваши изменения в файлы в его рабочем каталоге.

9. Чтобы увидеть журнальные записи для измененных файлов, можно использовать команду cvs log.
10. Журнальные записи выводятся на экран в обратном хронологическом порядке, исходя из предположения, что недавние изменения обычно более интересны. Каждая запись описывает одно изменение в файле, и может быть разобрано на составные части так:
    1. `revision версия' Каждая версия файла имеет уникальный номер редакции. Номера редакции выглядят как `1.1', `1.2', `1.3.2.2' или даже `1.3.2.2.4.5'. По умолчанию номер 1.1 -- это первая редакция файла. Каждое следующее редактирование

увеличивает последнюю цифру на единицу.

2. `date: дата; author: имя; ...' В этой строке находится дата изменения и имя пользователя, зафиксировавшего это изменение; остаток строки не очень интересен.
3. `описание' Это описание изменения.

11. Команда `cvs log` может выбирать журнальные записи по дате или по номеру редакции; за описанием деталей обращайтесь к руководству.
12. Если вы хотите взглянуть на соответствующее изменение, то можете использовать команду `cvs diff`.

Пример: если вы хотите увидеть, какие изменения зафиксированы в качестве редакции 1.7, используйте такую команду:

```
$ cvs diff -c -r 1.6 -r 1.7 файл
```

13. Как и вывод команды `diff`, вывод команды `cvs diff` обычно называется заплатой (`patch`), потому что разработчики традиционно использовали этот формат для распространения исправлений и небольших новый возможностей.
14. Если можно использовать `cvs diff', чтобы получить точное содержание любого изменения, то зачем тогда придумывать еще журнальную запись о нем? Очевидно, что журнальные записи короче, чем тексты изменений, и позволяют читателю получить общее понимание изменения без необходимости углубляться в детали.
15. Хорошая запись в журнале описывает причину, по которой было сделано изменение.

Пример: плохая журнальная запись для редакции 1.7 может звучать как "Преобразовать `t' к нижнему регистру". Это правильно, но бесполезно - `cvs diff' предоставляет точно ту же информацию, и гораздо яснее. Гораздо лучшей журнальной записью было бы "Сделать эту проверку независящей от регистра", потому что это гораздо яснее описывает причину любому, кто понимает, что происходит в файле.

16. Как уже упоминалось, команда `cvs update' объединяет изменения, сделанные другими разработчиками, с исходными текстами в вашем рабочем каталоге. Если вы отредактировали файл одновременно с кем-то другим, CVS объединит ваши изменения.
17. Довольно легко представить себе, как работает объединение, если изменения были совершены в разных участках файла, но что если вы оба изменили одну и ту же строку? CVS называет эту ситуацию **конфликтом** и предоставляет вам самому разобраться с ним.
18. Важно понимать, что именно CVS считает конфликтом. CVS не понимает структуры файлов.

Пример: Если один разработчик добавляет новый аргумент в функцию и исправляет все ее вызовы, пока другой разработчик одновременно добавляет новый вызов этой функции, и не передает ей этот новый аргумент, что определенно является конфликтом - два изменения несовместимы - но CVS не сообщит об этом. Его понимание конфликтов строго текстуально.

Примечание: На практике, однако, конфликты случаются редко. Обычно они происходят потому, что два человека пытаются справиться с одной и той же проблемой, от недостатка взаимодействия между разработчиками, или от разногласий по поводу архитектуры программы. Правильное распределение задач между разработчиками уменьшает вероятность конфликтов.

19. Многие системы контроля версий позволяют разработчику **блокировать** файл, предотвращая внесение в него изменений до тех пор, пока его собственные изменения не будут зафиксированы. Блокировки уместны в некоторых ситуациях, но их использование

не всегда лучше, чем использование CVS - без блокировок.

20. Изменения обычно объединяются без проблем, а разработчики иногда забывают убрать блокировку, в обоих случаях явное блокирование приводит к ненужным задержкам. Более того, блокировки предотвращают только текстуальные конфликты - они ничего не могут поделать с семантическими конфликтами типа вышеописанного - когда два разработчика редактируют разные файлы.

9.3. Безопасность CVS

1. Если архив находится на локальной машине, то с доступом и защитой все более-менее понятно. Вы можете установить переменную среды `$CVSROOT` в корневую директорию CVS-архива, или вызвать `checkout` с опцией `-d <directory>`.
2. Если архив находится на удаленной машине, необходимо указать CVS на какой именно машине он находится, и каким способом к ней может быть получен доступ.
3. Существует несколько способов, но для простоты и безопасности лучше использовать SSH.
4. Порядок определения удаленного *$CVSROOT* следующий:

```
:метод:[[имя_пользователя]:[пароль]@]имя_хоста[:[порт]]:/путь/к/архиву;
```

Пример:

```
:ext:jenn@cvs.example.com.au:/usr/local/cvsroot
```

Примечание: Для использования SSH мы применяем метод :ext:. В этом методе для соединения с CVS-сервером используется внешний `rsh` для CVS или совместимая с `rsh` программа.

5. Чтобы указать CVS использовать `SSH` вместо `rsh`, установите переменную среды `$CVS_RSH` равной `SSH`.

Примечание: Обеспечьте, чтобы SSH был установлен на сервере и на всех клиентах, чтобы были сгенерированы SSH-ключи и у пользователей на всех машинах были имена и пароли. Если пользователи те же, часть имя_пользователя@ в строке CVSROOT не нужна. Если используется стандартный порт SSH, порт указывать не нужно.

```
cvs -d :ext:cvs.example.com.au:/usr/local/cvsroot checkout sample
```

6. Все файлы в архиве предназначены только для чтения. Права доступа для этих файлов не должны изменяться.
7. Для контроля доступа используйте права доступа к директориям.

Пример: можно создать группу для тех, кто должен иметь доступ к модулю, и устанавить для этой группы право записи в директорию.

8. При использовании удаленного архива установите корневой директории модуля SGID-бит, чтобы обеспечить правильные права доступа для всех подчиненных директорий.
9. При использовании локального архива можно установить `$CVSUMASK` для контроля прав доступа к файлам и директориям в архиве.
10. Безопасность проекта включает в себя безопасность архива и сохранность всех извлеченных копий.

Примечание: Недостаточно просто обеспечить сохранность архива, тщательное шифрование всех пересылаемых данных, если кто-нибудь может спокойно зайти по окончании рабочего дня в офис и записать CD с вашими данными. Осуществляйте повседневную физическую и сетевую защиту машин, куда поступают извлекаемые файлы.

Глава 10. (Приложение) Операторы, конструкции и функции языка Perl.

10.1. Операторы Perl

1. Все операторы C присутствуют в языке Perl, за исключением оператора приведения типов (type), оператора обращения к содержимому указателя *ptr и оператора выбора члена структуры var.member или var->member.
2. Кроме того, в языке Perl реализовано много новых операторов для использования в таких операциях как сравнение и обработка строк.

10.1.1. Арифметические операторы

1. Арифметические операторы действуют на числовые значения, и их результатом является число.
2. Если выражение включает строковые операнды, то Perl конвертирует строковые значения в числовые перед тем, как оценить выражение.
3. Perl выполняет преобразование строк в числа подобно тому, как это делает функция atof() языка C.
4. В настоящее время Perl поддерживает следующие арифметические операторы:
   1. + сложение
   2. - вычитание или изменение знака
   3. * умножение
   4. / деление (только для чисел с плавающей запятой)
   5. % взятие по модулю (только для целочисленных значений)

Пример: арифметических операций языка Perl:

```
$x = 2.5;
$y = 3;
print ($x + 2*$y);              # выведет 8.5
print (7 / $y);               # выведет 2.3333333
print int (7 / $y);             # выведет 2
print (7 % $y);               # выведет 1
print (7.5 % $y);              # выведет 1
```

Примечание: В языке Perl оператор деления всегда имеет результатом число с плавающей точкой, а результатом взятия одного числа по модулю другого является целое число и причем предварительно оба операнда преобразуются к целому типу.

Пример: Рассмотрим следующую операцию взятия по модулю:

```
print (7.9 % 3.6);       #    выведет 1 то же (7 % 3) = 1
```

5. Perl также поддерживает операторы инкремента и декремента:

1. ++ инкремент в префиксной или постфиксной форме
2. - декремент в префиксной или постфиксной форме

Пример: операций инкремента и декремента:

```
$x = 4;
++$x;
print $x;    # выведет 5
$y = $x-;    # уменьшит x после присвоения    y значения x
print "$y $x"  # выведет 5 4
```

6. Perl обеспечивает арифметический оператор для возведения в степень (**).

Пример: использования операции возведения в степень:

```
$x = 2 ** 3;     # результат 8
$x = 2 ** 0.5;    # квадратный корень из 2
$x = -2 ** -3;    # 1/(-2 в кубе), результат    -1/8 (-0.125)
```

10.1.2. Побитовые операторы

1. Побитовые операторы воздействуют на бинарное представление целых чисел и имеют целочисленный результат.
2. Если операндом является строка или дробное число, Perl предварительно преобразуемого в целое число, обрабатывает операнд, используя 32-битное представление.
3. Все побитовые операторы C представлены в языке Perl:
   1. | побитовое ИЛИ
   2. & побитовое И
   3. ^ побитовое исключающее ИЛИ
   4. ~ побитовая инверсия
   5. << сдвиг влево
   6. >> сдвиг вправо
4. **Пример:** побитовых операций:

```
$x = 5;               # 101 в двоичном
  $y = 3;               # 011 в двоичном
  print $x | $y;            # 7 (111)
  print $x & $y;            # 1 (001)
  print $x ^ $y            # 6 (110)
  print $x & ~1;           # 4 (100)
  print $x << 2            # 20 (10100)
  print $x >> 1            # 2 (10)
```

10.1.3. Операторы сравнения

1. Операторы сравнения сравнивают величины двух операндов.
2. Также как при работе с арифметическими операторами, Perl преобразует строчные операнды в численные перед тем, как выполнять сравнение.
3. Для того чтобы позволить скрипту сравнивать строки, которые не являются числами, Perl имеет дополнительные операторы строкового сравнения. Эти операторы сравнивают строки, используя величины ASCII.
4. Если численное значение задано как операнд присравнении строк, оно сначала преобразуется в строку.
5. Таблица перечисляет операторы сравнения:

Число	Строка	Значение
= =	eq	равно
!=	ne	не равно
>	gt	больше чем
<	it	меньше чем
>=	ge	больше или равно
<=	le	меньше или равно
<=>	cmp	не равно (результат со знаком)

6. Результатом операции сравнения является единица, если сравнение истинно и нуль в противном случае.
7. Однако последняя операция (<=> или cmp) может возвращать значения -1, 0 или 1 в зависимости от того, является ли значение первого операнда меньше, второго, равным ему или большим.

Примечание: Оператор cmp языка Perl ведет себя, аналогично функции Strcmp() библиотеки времени выполнения языка C.

Пример: сравнения:

```
$x = 5;                # x равно 5
print ($x < 4);         #    если false, то выведет 0
```

10.1.4. Логические операторы

1. Логические операторы анализируют булевы выражения и возвращают значения <истинно> или <ложно> в качестве результата.
2. Perl обрабатывает операнды логических операций как булевы величины, т.е. как истинное или ложное значение.
3. Логические операторы языка Perl включают следующие:
   1. || логическое ИЛИ

   2. && логическое И
4. Perl всегда обрабатывает логические выражения слева направо.
5. Кроме того. Perl всегда прекращает оценку, если уже выполненной оценки достаточно, чтобы определить значение результата.
6. В дополнение к общим логическим операторам Perl поддерживает следующие дополнительные логические операторы:
   1. ! логическое отрицание ()
   2. -: условная операция
   3. , последовательное выполнение
7. Оператор логического отрицания (!) заменяет значение булевой величины на противоположную.
8. Так же как и в C, в языке Perl условный оператор (-:) использует три операнда. Выражение, использующее условный оператор, имеет следующую форму:

```
 Condition
- true-result : false-result
```

Пример: следующее выражение использует условный оператор для того, чтобы предоставить Бобу полный доступ, а всем остальным ограниченный:

```
   $access = ($user eq 'Bob'
- 'Full' : 'Limited');
```

9. Оператор последовательного выполнения (,) (также известный как оператор запятая) не является вполне логическим оператором, поскольку он не анализирует истинность своих операндов. Perl выполняет операнды оператора последовательного выполнения слева направо и возвращает значение самого правого операнда.

Пример: иллюстрирует использование оператора запятая в цикле for.

```
   For ($i=0, $j=10; $i<10; $i++, $j-)
       {
          print i$,' ',$j
       }
```

10.1.5. Строковые операторы

1. Поскольку Perl представляет собой язык для обработки текста, неудивительно, что в него включены дополнительные операторы для работы со строками.
2. Ниже перечисляются операторы обработки строк:
   1. . конкатенация строк
   2. x репликация
   3. =~ сопоставление переменной с образцом
   4. !~ то же, что и предыдущее, но с дополненным отрицанием результата

Пример:

```
   print 'b' . 'an' x 2 . 'a';       # выведет 'banana'
```

Примечание: это выражение использует конкатенацию строк и оператор репликации для того, чтобы напечатать строку <banana>.Два последних оператора используются для проверки того, включает ли строковый операнд заданный образец. Этот вопрос детально обсуждается в разделе <Регулярные выражения>. Следующий пример иллюстрирует их использование:

```
$var = 'banana';
print ($var =~ /ana/)


- TRUE : FALSE;
```

Примечание: В этом случае оператор проверки вхождения в строку образца(=~) использовался для проверки того, входит ли образец ana в переменную $var. В данном случае выражение принимает значение <истинно>.

10.1.6. Операторы присваивания

1. Операторы присваивания заставляют Perl выполнить специальные операции со значениями, которые появились с правой стороны оператора, и затем выполнить присваивание:

   ```
   =    +=    -=    *=    /=    %=    |=    &=
   ^=    ~=    <<=    >>=    **=    .=    x=
   ```

2. lvalue представляет собой имя того, что стоит с левой стороны оператора присваивания.
3. lvalue представляет собой целостность, которой может быть присвоено значение, например, lvalue может быть переменной.

Пример: скрипт Perl не может присвоить значение строке символов, наподобие выражения <Bob> = 32, поскольку <Bob> не является lvalue. Тем не менее, скрипт может присвоить значение переменной $Bob, например, следующим образом $Bob = 32, поскольку переменная $Bob является lvalue. В языке Perl любая целостность,которая может использоваться как lvalue, обычно таковой и является. Например, следующее выражение упаковывает (pack) и распаковывает (unpack) список значений, причем список переменных в первом случае и три скалярных во втором являются lvalues:

```
@color = ($r, $g, $b);          # пакет цветов
($r, $g, $b) = @color;          # распаковка цвета
```

4. Когда вы работаете со списками в языке Perl, оператор присваивания не обязательно относится ко всему списку. Скрипт может присваивать значения отдельным элементам списка, как показано ниже:

   ```
   @items[2,4,7] = (100,200,300);
   ```

Примечание: В этом случае оператор присваивает значение трем элементам списка. Аналогичным образом следующее выражение распаковывает элементы списка, присваивая значения двух первых элементов двум скалярным переменным, а остаток массива - списочной переменной:

```
($arg1,$arg2,@rest) = @ARGV;  # можно смешать скаляры и массивы
```

10.1.7. Операции для работы со списками

1. В состав операций для работы со списками входят следующие:
   1. , конструктор списков

   2. .. оператор области
   3. x оператор репликации
2. Оператор области возвращает в качестве значения последовательность целых чисел, которая начинается от левого операнда и продолжается до правого операнда включительно.
3. Скрипты часто используют оператор области совместно с конструктором списков для создания списков.

Пример: следующее выражение использует оператор области для того, чтобы создать список под именем @digits,который содержит числа от нуля до девяти:

```
@digits = 0..9;     # список (1,2,3,4,5,6,7,8,9)
```

Примечание: Аналогичным образом, это выражение может использовать оператор области для создания области изменений индексов массива. Предположим, что список @days содержит дни недели (начиная с воскресенья). В следующем выражении списку @weekdays присваиваютсязначения, начиная от понедельника до пятницы:

```
@weekend = @days[1..5];
```

Пример: следующее выражение использует два оператора области для создания списка шестнадцатиричных цифр:

```
@hex_digits = (0..9,a..f);
```

4. Оператор репликации просто создает копии данного операнда указанное число раз.

10.1.8. Операторы для работы с файлами

1. Perl содержит обширный список операторов для работы с файлами. Имеется не менее 27 операторов, возвращающих специфическую информацию о файле, даже не открывая его.
2. Многие операторы языка Perl ориентированы на системы UNIX, но следующие операторы работают на любых системах:
   1. -d проверяет наличие каталога
   2. -e определяет наличие файла
   3. -s определяет размер файла
   4. -w определяет, можно ли писать в данный файл

Пример: Следующие два файловых оператора возвращают булево значение. Третий оператор возвращает размер файла в байтах. Следующий текст иллюстрирует использование этих операторов:

```
if (-e,'perl.exe')
   {
      print 'File size    is:' -s 'perl.exe';
   }
  else
   {
      print 'can\'    t find perl.exe\n';
   }
```

```
(-w 'SomeFile') || die "Cannot    write to SomeFile\n";
```

10.2. Конструкции языка Perl

10.2.1. Простые и составные операторы

1. Простым выражением называется любая допустимая комбинация операторов и операндов.
2. В языке Perl оператором является выражение, заканчивающееся точкой с запятой.
3. Когда выводите текст программы в отладчик, можно опускать точку с запятой, поскольку отладчик поставит ее за вас.

Пример: простой оператор присваивания на языке Perl:

```
$Title = 'Web Programming';
```

4. Так же как и при программировании на C, скрипты Perl могут содержать блоки операторов, или составные операторы, которые помещаются в фигурные скобки {}, как показано ниже:

```
{
    # Операторы
    # Другой блок операторов
}
```

5. Ваши скрипты будут широко использовать блоки инструкций наряду с более сложными операторами.
6. Скрипты на языке Perl могут использовать блоки инструкций для определения области видимости (scope) локальных переменных.
7. Определение локальных переменных в блоке не является автоматическим. Для их декларации скрипт должен использовать ключевое слово local.

10.2.2. Условные операторы

1. В языке Perl инструкции оператора if обязательно должны быть заключены в фигурные скобки, образуя блок.

Пример:

```
if (expr)
    statement;           // приемлемо для C но не для Perl
if (expr)
    {
        statement;     # вот так нужно делать в Perl
    }
```

2. Аналогично инструкции `else` также должны быть заключены в фигурные скобки и образовать блок:

Пример:

```
if (expr1)
  {
    statament1;
  }
elsif (expr2)
  {
    statement2;
  }
else
  {
    ststement3;
   }
```

10.2.3. Оператор unless

1. Скрипты Perl часто содержат оператор `unless`, который обеспечивает логическое отрицание

Пример:

```
unless (expr)
  {
    statement;
  }
```

Примечание: В отличие от языка C, Perl не содержит оператора переключения switch.

10.2.4. Оператор do

1. Одним из частных случаев блочных операторов служит оператор do, который позволяет блоку инструкций возвращать значения.
2. Значением, которое оператор do возвращает, является значение последнего выражения, оцененного в рамках блока.

Пример: следующий оператор do сравнивает строковую переменную $Month с месяцами года и присваивает переменной $DayCount значение, равное числу дней в месяце:

```
$DayCount = do
 {
if ($Month eq 'September' || $Month eq 'April' ||    $Month eq 'June' ||
$Month eq 'November')
 {
    30;
 }
elsif ($Month eq 'Februry')
```

```
     {
       $Year & 3
- 28 : 29;   #    Проверка на весокосный год
     }
   else
     {
       31;
     }
         };
```

Примечание: Обратите внимание, что Perl требует наличия точки с запятой в конце блока do.

10.2.5. Циклы и ветвления

1. Perl поддерживает операторы циклов for, while и do с небольшими отличиями от их реализации в языке C. Существенным отличием служит то, что Perl требует использования инструкций блоками, заключенными в фигурные скобки.
2. Кроме того, Perl расширяет конструкцию цикла, чтобы обеспечить ее некоторые новые формы. В следующих примерах циклы for, while и do работают аналогичным образом на языках C и Perl:

Пример:

```
for($i = 0; $i< 100;$i++)
 {
printf("%d\n", $i) ;
     }
while ($i > 0)
 {
    printf("%d\n", $i-);
 }
do {
printf("%d\n", $i++);
   } while ($i < 0);
```

3. Конструкция циклов на языке Perl обеспечивает некоторые новые, более гибкие и более интуитивно понятные конструкции:
   1. last выход из цикла (как оператор C break)
   2. next начать новую итерацию (как оператор C continue)
   3. redo повторить текущую итерацию
4. Для понимания конструкций циклов на языке Perl, необходимо разобраться с использованием блока continue.

Пример: Рассмотрим следующий цикл while, который содержит блок continue:

```
$i = 100;
```

```
while ($i > 0)
 {
print $i;
  } continue {$i-}
```

5. Вы можете представлять себе блок continue как третье выражение в цикле for, которое выполняется на каждой итерации. Аналогичным образом Perl выполняет блок continue в конце каждой итерации.
6. Однако, блок continue обеспечивает скрипту более полный контроль над процессом, чем обеспечивает цикл for.
   1. Когда цикл на языке Perl использует оператор next, блок continue все равно выполняется, если только он существует.
   2. Однако если цикл использует оператор redo, блок continue не исполняется.

10.2.6. Метки

1. В скрипте Perl метки просто означают имя, соответствующее некоему положению внутри скрипта.
2. Имена меток оканчиваются двоеточием.
3. Используя оператор `goto`, скрипт может осуществлять переходы на метку.
4. Дополнительно могут использоваться операторы `last`, `next` и `redo`, для перехода к метке.

Пример: Следующий код иллюстрирует использование оператора `last` для перехода на метку:

```
outerloop: while ($i > 0)
       {
       while ($j > 0)
        {
               #Здесь какой-нибудь другой процесс
               if ($needToAboutLoop)
               {
                   last outerloop;
               }
        }
    }
```

Примечание: В этом случае инструкция содержит ветвь last для перехода наметку outerloop и окончания выполнения цикла.

10.2.7. Цикл until

1. В цикле `until`, инструкции выполняются до тех пор, пока не будет выполнено условие.

Пример:

```
until (expr)
{
    statement;
}
```

2. Аналогичным образом конструкция `do until` выполняет действие, а затем проверяет выражение:

```
do
{
    statement;
} until (expr);
```

10.2.8. Циклы for и foreach

1. Perl поддерживает цикл `for` совершенно аналогично языку C:

```
for (statement1; expression; statement2)
  {
    statement3;
  }
```

2. Дополнительно Perl содержит конструкцию цикла `foreach`, который позволяет скрипту организовывать итерации в списках и массивах.

Пример:

```
@list = ('a','b','c');
foreach $arg (@list)
  {
    print "List item: $arg\n";
  }
foreach $i (1..10)
  {
    print "iteration $i\n"
   }
```

Примечание: В первом случае цикл foreach осуществлял перебор значений в списочной переменной @list. Во втором примере в цикле foreach осуществляется перебор чисел в диапазоне от 1 до 10. Внутри цикла foreach может фигурировать список, состоящий из литералов, или массив, как было проиллюстрировано в предыдущем примере. После выполнения одной итерации циклом, специальная скалярная переменная {$arg в первом случае и $i во втором случае) принимает значение из заданного списка элементов.

3. Область видимости этой скалярной переменной в цикле `foreach` ограничивается телом цикла. Поэтому скалярная переменная цикла, `foreach` не будет конфликтовать с идентичным именем переменной, определенной вне цикла.

Пример: В следующем коде переменная с именем $i используется внутри и вне цикла foreach:

```
$i = 1001;
```

```
foreach $i (1..9)
  {
    print "$i\n";         # цикл выведет 123456789
  }
print "$i\n";
```

Примечание: Как можно увидеть из этого примера, переменная $i, используемая для организации итераций цикла, не конфликтует с переменной $i, определенной вне цикла.

4. Особенностью цикла, `foreach` является возможность модификации элементов массива. Будьте осторожны при реализации этой возможности!

Пример: Рассмотрим следующий цикл `foreach`, который добавляет значение 10 каждому элементу массива:

```
@list = 1..5;
foreach $i (@list)
   {
    $i += 10;
   }
$, = ' ';
print @list;      # выведет 11 12 13    14 15
```

5. В языке Perl имена `foreach` и `for` рассматриваются как синонимы. Поэтому в скриптах можно использовать эти имена попеременно. Perl, в свою очередь, будет определять тип цикла, основываясь на его контексте.

10.2.9. Оператор безусловного перехода goto

1. Perl поддерживает оператор безусловного перехода `goto`, который является идентичным такому же оператору языка программирования C.

Пример: использования оператора goto для вывода чисел от 1 до 10:

```
$i = 1;
loop:
    print $i++, ' ';
    if ($i <=10)
        {
               goto loop;
  }
```

10.3. Функции

10.3.1. Функции обработки строк

1. Функция `chop` удаляет последний символ строки. Она имеет следующий формат:

```
$character = chop(Str);
```

Примечание: Функция chop возвращает удаленный символ. Скрипты языка Perl широко используют chop для удаления символа перехода на новую строку и символа конца строки.

2. Функция `index` осуществляет поиск заданной подстроки в строке. Она имеет следующий формат:

```
$location = index(Str, SubStr[, Offset]);
```

   1. Функция `index` возвращает индекс первого вхождения подстроки(`SubStr`) в строку (`Str`).
   2. Опционально может быть задан сдвиг от начала (`Offset`), после которого начинается поиск. Если подстрока не найдена, возвращается значение -1.

Пример: В следующем примере функция `index` ищет вхождения подстроки 'na' после третьего символа встроке 'banana':

```
print index('banana','na',3); # Выведет 4.
```

3. Функция `rindex` ищет последнее, самое правое вхождение подстроки в строку и возвращает значение позиции первого символа подстроки. Функция имеет следующий формат:

```
$location = rindex(Str, SubStr);
```

Примечание: Эта функция аналогична функции index, за исключением того,что она возвращает последнее вхождение, а не первое.

4. Функция length возвращает число символов в строке. Она имеет следующий формат:

```
$len = length(Str);
```

5. Функция substr используется для удаления части строки. Она имеет следующий формат:

```
$substring = substr(Str, Offset[,Len]);
```

   1. Функция возвращает подстроку, т. е. часть строки, длина которой не превышает величины, заданной необязательным параметром `Len`.
   2. Возвращаемая подстрока строки `str` начинается с символа в позиции, заданной сдвигом `Offset`.
   3. Если параметр `Len` опущен, то возвращаемая строка содержит символы до конца строки включительно.
   4. Если параметр `Offset` отрицательный, то сдвиг вычисляется от конца строки.
   5. Наконец, скрипт может использовать `substr` как `lvalue` для выполнения операции присваивания.

Пример: Следующий фрагмент кода иллюстрирует использование функции `substr`.

```
print substr('orange',3);   #Выведет 'nge'
print substr('orange',-2);   # Выведет 'ge'
print substr('orange',2,2);  # Выведет 'an'
$str = 'apple';
substr($str,-3) = 'ricot';
print $str;             # Выведет 'apricot'
```

Примечание: Часто использование операторов языка Perl для регулярных выражений оказывается

более эффективным, чем функции substr. Регулярные выражения обсуждаются ниже в данной главе.

6. Функция join соединяет список элементов в строку, разделяя каждый элемент заданным символом. Она имеет следующий формат:

```
$new_string = join(Str,List);
```

7. Функция `join` конвертирует каждый элемент списка в строку и соединяет строки.

Пример: Следующий фрагмент кода иллюстрирует использование функции `join`:

```
$str = join(',', 0..4,10,20); # Список будет '0,1,2,3,4,10,20'
$strn = join ("\t", $a, $b, $c);# Смешает списки
```

8. Функция `split` разделяет содержимое строки на список элементов. Она имеет следующий формат:

```
split(Delimeter, Str[,Limit]);
```

   1. Аргумент Delimeter определяет символ, по которому осуществляется разделение, например, пробел, слово, символ табуляции и т. д.
   2. Параметр Limit задает максимальное число элементов, которое может содержать список.

10.3.2. Функции для обработки списков

1. Функция reverse реверсирует элементы списка. Она имеет следующий формат:

```
@new_list = reverse(@List);
```

Пример: Следующий пример иллюстрирует использование функции reverse:

```
@list = reverse(1..5); # Результат 5,4,3,2,1
@list = reverse(@list); # Результат 1,2,3,4,5
```

2. Функция `sort` сортирует элементы списка. Она имеет следующийформат:

```
@new_list = sort(@List);
```

или

```
@new_list = sort(Subroutine @List);
```

или

```
@new_list = sort(BlockStatement @List);
```

   1. Функция `sort` размещает элементы в списке, упорядочивая их в соответствии с порядковыми номерами символов в таблице ASCII-кодов.
   2. Так же как и функция `reverse`, функция `sort` возвращает в качестве значения новый список и не воздействует на исходный список.

Пример: Следующий пример иллюстрирует использование функции sort:

```
@list = sort (1,5,2,3,4);       #    Результат 1,2,3,4,5
@list = sort(1,2,10);            # 1,10,2 сортировка в ASCII
```

3. В подпрограмме или блоке можно изменять упорядочение, в соответствии с которым выполняется сортировка.

Пример: иллюстрирует использование функции sort.

```
@list = sort({$a <=> $b} (2,1,10));          # @list 1,2,10
@list = sort({$b <=> $a}) (2,1,10);          # @list 10,2,1
sub mycomp
    {
        $b <=> $a
    }
@list = sort(mycomp (2,1,10));      # @list    10,2,1
```

10.3.3. Функции работы с массивами

1. Скрипты языка Perl используют фикции `push` и `pop` для того,чтобы добавлять и удалять элементы с конца массива. Иными словами, функции `push` и `pop` позволяют скриптам выполнять операции со стеком по принципу: последним вошел, первым вышел.
2. Функция `push` имеет следующий формат:

```
push(@ARRAY, LIST);
```

Пример: Следующий фрагмент иллюстрирует использование функции `push`:

```
@list = ();
push(@list,10,20);        # @list    теперь (10,20)
push(@list,1..3);        #    @list теперь (10,20,1,2,3)
```

3. В противоположность этому функция `pop` удаляет элемент, который был вставлен в стек последним и возвращает значение этого элемента. Функция `pop` имеет следующий формат:

```
$value = pop(@ARRAY);
```

Пример: Следующий фрагмент программы иллюстрирует использование функции `pop`:

```
# Возьмём @list из предыдущего примера
    print pop(@list);              # Выведет 3
    print pop(@list);              # Выведет 2
  # Теперь @list (10,20)
```

4. Функция `shift` удаляет и возвращает элемент из начала массива. Эта функция аналогична функции pop с тем только отличием, что работает от начала массива по принципу FIFO (<первым вошел, первым вышел>). Функция `shift` имеет следующий формат:

```
$value = shift(@ARRAY);
```

Пример: Следующий фрагмент программы иллюстрирует использование функции shift:

```
# Возьмём @list из предыдущего примера
print shift(@list);      # Выведет    10
print shift(@list);      # Выведет    20
    # Теперь @list ()
```

5. Функция `unshift` добавляет один или больше элементов к началу массива. Она имеет следующий код: