Федеральное государственное автономное образовательное
учреждение высшего образования
«Санкт-Петербургский государственный университет
аэрокосмического приборостроения»

Основы программной инженерии

Методические указания к выполнению лабораторных работ

Составители:

к.т.н., стар. преп. П.А. Охтилев, асс. А.Э. Зянчурин

Рецензент:

д.т.н., проф. М.Ю. Охтилев

Санкт-Петербург – 2021

Содержание

Требования к оформлению отчетов

Структура и форма отчета о лабораторной работе

Итоговый отчёт по лабораторным работам должен состоять из частей, перечисленных ниже. Отсутствие указанных ниже частей не допускается. Общий объем отчёта должен составлять не менее 15 страниц. Страницы должны быть пронумерованы. Размер шрифта основного текста — 14 пунктов.

1. *Титульный лист* должен соответствовать образцу на сайте ГУАП. При оформлении титульного листа обязательно наличие следующей информации:
   - название дисциплины;
   - ФИО преподавателя, принимающего работу;
   - ФИО обучающихся, выполнивших работу.
   - Отчёты, содержащие неверную информацию на титульном листе, к сдаче не принимаются.
2. *Содержание* с указанием номеров страниц (желательно составленное автоматически).
3. *Подписанное преподавателем задание* на лабораторные работы.
4. *Краткое описание хода выполнения работ*: постановка задачи на каждую работу; описание использованных моделей, алгоритмов, программных компонент. Описание должно быть сопровождено расчетными материалами, снимками с экрана компьютера («скриншотами»), отражающими ход выполнения работы.
5. *Выводы* по результатам выполняемых лабораторных работ.
6. *Список цитируемой и использованной литературы.*

Требования к оформлению отчета о лабораторной работе

Изложение текста и оформление лабораторных работ следует выполнять в соответствии с ГОСТ 2.105-2019 – ЕСКД и общие требования к текстовым документам ГОСТ 7.32 – 2017 – СИБИД, соблюдая следующие требования.

1. *Оформление титульного листа*. Титульный лист следует оформлять на бланке. Бланки для оформления титульных листов учебных работ представлены на сайте ГУАП в разделе «Нормативная документация» для учебного процесса.
2. *Оформление основного текста работы:*
   - следует использовать шрифт Times New Roman размером не менее 12 пт (допускается 14 пт), строчный, без выделения, с выравниванием по ширине;
   - абзацный отступ должен быть одинаковым и равен по всему тексту 1,25 см;

- строки разделяются полуторным интервалом;
- поля страницы: верхнее и нижнее – 20 мм, левое – 30 мм, правое – 15 мм;
- полужирный шрифт применяется только для заголовков разделов и подразделов. заголовков структурных элементов;
- разрешается использовать компьютерные возможности акцентирования внимания на определенных терминах, формулах, теоремах, применяя шрифты разной гарнитуры;
- наименования структурных элементов работы: «СОДЕРЖАНИЕ», «ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ», «ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ», «ВВЕДЕНИЕ», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ», «ПРИЛОЖЕНИЕ» следует располагать в середине строки без точки в конце, прописными (заглавными) буквами, не подчеркивая;
- введение и заключение не нумеруются.
- каждый структурный элемент и каждый раздел основной части следует начинать с новой страницы.

3. *Оформление основной части работы следует делить на разделы и подразделы:*
   - разделы и подразделы должны иметь порядковую нумерацию в пределах всего текста, за исключением приложений;
   - нумеровать их следует арабскими цифрами;
   - номер подраздела должен включать номер раздела и порядковый номер подраздела, разделенные точкой;
   - после номера раздела и подраздела в тексте точка не ставится;
   - разделы и подразделы должны иметь заголовки;
   - если заголовок раздела, подраздела или пункта занимает не одну строку, то каждая следующая строка должна начинаться с начала строки, без абзацного отступа;
   - заголовки разделов и подразделов следует печатать с абзацного отступа с прописной буквы, полужирным шрифтом, без точки в конце, не подчеркивая;
   - если заголовок состоит из двух предложений, их разделяют точкой;
   - переносы слов в заголовках не допускаются;
   - обозначение подразделов следует располагать после абзацного отступа, равного двум знакам относительно обозначения разделов;
   - обозначение пунктов приводят после абзацного отступа, равного четырем знакам относительно обозначения разделов;
   - в содержании должны приводиться наименования структурных элементов, после заголовка каждого из них ставят отточие и приводят номер страницы;
   - содержание должно включать введение, наименование всех разделов и подразделов, заключение, список использованных

источников и наименование приложений с указанием номеров страниц, с которых начинаются эти элементы работы;
- перечень сокращений и обозначений следует располагать в алфавитном порядке. Если условных обозначений в отчете менее трех, перечень не составляется.

4. *Оформление нумерации страниц:*
   - страницы следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту работы;
   - номер страницы следует проставлять в центре нижней части листа без точки;
   - титульный лист должен включаться в общую нумерацию страниц;
   - номер страницы на титульном листе не проставляется.
5. *Оформление рисунков:*
   - на все рисунки должны быть ссылки: …в соответствии с рисунком 1;
   - рисунки, за исключением рисунков приложений, следует нумеровать арабскими цифрами;
   - рисунки могут иметь наименование и пояснительные данные, которые помещаются в строке над названием рисунка: Рисунок 1 – Детали прибора
   - рисунки каждого приложения должны обозначаться отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения: Рисунок А.3 (третий рисунок приложения А).
6. *Оформление таблиц:*
   - на все таблицы должны быть ссылки, при ссылке следует писать слово «таблица» с указанием ее номера;
   - таблицы, за исключением таблиц приложений, следует нумеровать арабскими цифрами сквозной нумерацией;
   - наименование таблицы следует помещать над таблицей слева, без абзацного отступа: Таблица 1 – Детали прибора
   - таблицы каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения:
     Таблица Б.2 (вторая таблица приложения Б)
   - если таблица переносится на следующую страницу, под заголовком граф должна быть строка с номером колонок, на следующей странице под названием «Продолжение таблицы 1» дается строка с номером колонок.
7. *Оформление приложений:*
   - в тексте отчета на все приложения должны быть ссылки, приложения располагаются в порядке ссылок на них в тексте отчета;

- каждое приложение следует размещать с новой страницы с указанием в верхней части страницы слова «ПРИЛОЖЕНИЕ»;
- заголовок приложения записывают с прописной буквы, полужирным шрифтом, отдельной строкой по центру без точки в конце;
- приложения обозначаются прописными буквами кириллического алфавита, начиная с А, за исключением букв Ё, З, Й О, Ч, Ъ, Ы, Ь;
- допускается обозначение приложений буквами латинского алфавита, за исключением I и O;
- в случае полного использования букв кириллического и латинского алфавита допускается обозначать приложения арабскими цифрами;
- приложение следует располагать после списка использованных источников.

8. *Список использованных источников.* Сведения об источниках следует располагать в порядке появления ссылок на источник и в тексте работы и нумеровать арабскими цифрами с точкой и печатать с абзацного отступа. Список использованных источников следует оформлять в соответствии с ГОСТ 7.0.100 – 2018 «Библиографическая запись. Библиографическое описание». Примеры библиографического описания в соответствии с требованиями ГОСТ 7.0.100 – 2018 представлены на сайте ГУАП в разделе «Нормативная документация» для учебного процесса.

Лабораторная работа №4. Применение инструментальных средств автоматизации процессов разработки программного обеспечения

Цель работы

Целью работы является формирование практических навыков применения инструментальных средств автоматизации процессов разработки программного обеспечения.

Задание на лабораторную работу

Автоматизация процессов компоновки, компиляции, развертывания и доставки прикладного программного обеспечения, а также установки и конфигурирования системного программного обеспечения посредством специальных сценариев последовательного выполнения программ (команд) входящих в состав операционных систем. Выполнение лабораторной работы предполагает индивидуальную работу в соответствии с вариантом задания.

Общие рекомендации по выполнению лабораторной работы.

Раздел №1. Назначение командно-сценарных языков

Знание языка командной оболочки является залогом успешного решения задач администрирования операционной системы. Даже если вы не предполагаете заниматься написанием своих сценариев. Во время загрузки Linux выполняется целый ряд сценариев из /etc/rc.d, которые настраивают конфигурацию операционной системы и запускают различные сервисы, поэтому очень важно четко понимать эти скрипты и иметь достаточно знаний, чтобы вносить в них какие-либо изменения.

Язык сценариев легок в изучении, в нем не так много специфических операторов и конструкций. Синтаксис языка достаточно прост и прямолинеен, он очень напоминает команды, которые приходится вводить в командной строке. Короткие сценарии практически не нуждаются в отладке, и даже отладка больших скриптов отнимает весьма незначительное время.

Bash-скрипты очень хорошо подходят для быстрого создания прототипов сложных приложений, даже не смотря на ограниченный набор языковых конструкций и определенную "медлительность". Такой метод позволяет детально проработать структуру будущего приложения,

обнаружить возможные "ловушки" и лишь затем приступить к кодированию на C, C++, Java, или Perl.

Скрипты возвращают нас к классической философии UNIX -- "разделяй и влавствуй" т.е. разделение сложного проекта на ряд простых подзадач. Многие считают такой подход наилучшим или, по меньшей мере, наиболее эстетичным способом решения возникающих проблем, нежели использование нового поколения языков -- "все-в-одном", таких как Python, Perl.

Для каких задач неприменимы скрипты:

- для ресурсоемких задач, особенно когда важна скорость исполнения (поиск, сортировка и т.п.);
- для задач, связанных с выполнением математических вычислений, особенно это касается вычислений с плавающей запятой, вычислений с повышенной точностью, комплексных чисел (для таких задач лучше использовать C++ или FORTRAN);
- для кросс-платформенного программирования (для этого лучше подходит язык C);
- для сложных приложений, когда структурирование является жизненной необходимостью (контроль за типами переменных, прототипами функций и т.п.);
- для целевых задач, от которых может зависеть успех предприятия;
- когда во главу угла поставлена безопасность системы, когда необходимо обеспечить целостность системы и защитить ее от вторжения, взлома и вандализма;
- для проектов, содержащих компоненты, очень тесно взаимодействующие между собой;
- для задач, выполняющих огромный объем работ с файлами;
- для задач, работающих с многомерными массивами;
- когда необходимо работать со структурами данных, такими как связанные списки или деревья;
- когда необходимо предоставить графический интерфейс с пользователем (GUI);
- когда необходим прямой доступ к аппаратуре компьютера;
- когда необходимо выполнять обмен через порты ввода-вывода или сокеты;
- когда необходимо использовать внешние библиотеки;
- для проприетарных, "закрытых" программ (скрипты представляют из себя исходные тексты программ, доступные для всеобщего обозрения).

Если выполняется хотя бы одно из вышеперечисленных условий, то вам лучше обратиться к более мощным скриптовым языкам

программирования, например Perl, Tcl, Python, Ruby или к высокоуровневым компилирующим языкам -- C, C++ или Java. Но даже в этом случае, создание прототипа приложения на языке shell может существенно облегчить разработку.

Название BASH – это аббревиатура от "Bourne-Again Shell" и игра слов от, ставшего уже классикой, "Bourne Shell" Стефена Бурна (Stephen Bourne). В последние годы BASH достиг такой популярности, что стал стандартной командной оболочкой de-facto для многих разновидностей UNIX. Большинство принципов программирования на BASH одинаково хорошо применимы и в других командных оболочках, таких как Korn Shell (ksh), от которой Bash позаимствовал некоторые особенности, и C Shell и его производных. (Примечательно, что C Shell не рекомендуется к использованию из-за отдельных проблем, отмеченных Томом Кристиансеном (Tom Christiansen) в октябре 1993 года на Usenet post

Раздел №2. Анализ шаблона сценария на bash

Bash – одно из самых простых и наиболее естественных решений, позволяющее совместить набор команд вместе, передать вывод от одной команды к другой и запустить какой-нибудь исполняемый файл. Хотя есть смысл писать более длинные и сложные сценарии на других языках, таких как Python, Ruby, fish или любой другой интерпретируемый язык, нет никакой уверенности в том, что он будет доступен везде.

Bash весьма далек от совершенства. Сложный синтаксис. Обработка ошибок сложна. Повсюду подводные камни. Однако работу с ним можно стандартизировать и выработать общие (актуальные в большинстве случаев) подходы.

Одним из таких подходов – стандартизация шаблона сценария на bash, на базе которого возможно разрабатывать весьма гибкие сценарии.

Шаблон:

```
#!/usr/bin/env bash

set -Eeuo pipefail
trap cleanup SIGINT SIGTERM ERR EXIT

script_dir=$(cd "$(dirname "${BASH_SOURCE[0]}")" &>/dev/null && pwd -P)
```

```
usage() {
  cat <<EOF
Usage: $(basename "${BASH_SOURCE[0]}") [-h] [-v] [-f] -p param_value arg1 [arg2...]

Script description here.

Available options:

-h, --help      Print this help and exit
-v, --verbose   Print script debug info
-f, --flag      Some flag description
-p, --param     Some param description
EOF
  exit
}

cleanup() {
  trap - SIGINT SIGTERM ERR EXIT
  # script cleanup here
}

setup_colors() {
  if [[ -t 2 ]] && [[ -z "${NO_COLOR-}" ]] && [[ "${TERM-}" != "dumb" ]]; then
    NOFORMAT='\033[0m' RED='\033[0;31m' GREEN='\033[0;32m' ORANGE='\033[0;33m' BLUE='\033[0;34m' PURPLE='\033[0;35m' CYAN='\033[0;36m' YELLOW='\033[1;33m'
  else
    NOFORMAT='' RED='' GREEN='' ORANGE='' BLUE='' PURPLE='' CYAN='' YELLOW=''
  fi
}

msg() {
  echo >&2 -e "${1-}"
}

die() {
  local msg=$1
  local code=${2-1} # default exit status 1
```

```
  msg "$msg"
  exit "$code"
}

setmsg() {
msg "${CYAN}Message!${NOFORMAT}"
}

parse_params() {
  # default values of variables set from params
  flag=0
  param=''

  while :; do
    case "${1-}" in
    -h | --help) usage ;;
    -v | --verbose) set -x ;;
    --no-color) NO_COLOR=1 ;;
    -f | --flag) flag=1 ;; # example flag
    -p | --param) ;; # example named parameter
    -c | --color) setmsg
      param="${2-}"
      shift
      ;;
    -?*) die "Unknown option: $1" ;;
    *) break ;;
    esac
    shift
  done

  args=("$@")

  # check required params and arguments
  [[ -z "${param-}" ]] && die "Missing required parameter: param"
  [[ ${#args[@]} -eq 0 ]] && die "Missing script arguments"

  return 0
```

```
}

setup_colors
parse_params "$@"
#setup_colors

# script logic here

msg "${RED}Read parameters:${NOFORMAT}"
msg "- flag: ${flag}"
msg "- param: ${param}"
msg "- arguments: ${args[*]-}"
```

Рассмотрим более подробно каждый блок шаблона сценария.

Интепретатор

#!/usr/bin/env bash.

Сценарии обычно начинаются с так называемого шебанга. Для лучшей совместимости предлагается использовать /usr/bin/env вместо непосредственно /bin/bash. Хотя, учитывая комментарии на многоих популярных ресурсах, даже это может иногда не сработать.

Работа над ошибками

set -Eeuo pipefail.

Команда 'set' изменяет параметры выполнения сценария. Обычно Bash игнорирует ошибки выполнения какой-либо команды, которая завершилась с ненулевым кодом возврата – он просто переходит к выполнению следующей строки.

Определение директории

script_dir=$(cd "$(dirname "${BASH_SOURCE[0]}")" &>/dev/null && pwd -P).

Эта строка пытается определить директорию расположения сценария. Часто наши скрипты работают по относительным путям от расположения самого скрипта, копируя файлы и выполняя команды, предполагая, что директория, где лежит скрипт, является так же нашей рабочей директорией. Но если CI-система запустит скрипт вот так: «/opt/ci/project/script.sh», то наш скрипт сработает не в директории проекта, а в каком-то совершенно другом месте. Сценарий не меняет рабочий каталог. Если скрипт выполняется из другого места и пользователь указывает относительный путь к какому-либо файлу, мы все равно сможем его прочитать.

Очистка консоли

trap cleanup SIGINT SIGTERM ERR EXIT

cleanup() {

* trap - SIGINT SIGTERM ERR EXIT*

* # script cleanup here*

}

В конце работы скрипта – при нормальном завершении, либо из-за ошибки или внешнего сигнала – будет выполнена функция cleanup(). Это то место, где можно, например, попытаться удалить все временные файлы, созданные скриптом в процессе работы.

Помните, что cleanup() может быть вызван не только в конце, но и при выполнении скриптом любой части его логики. Не обязательно, что все ресурсы, которые вы пытаетесь очистить, будут существовать.

Справочная информация

usage() {

* cat <<EOF*

Usage: $(basename "${BASH_SOURCE[0]}") [-h] [-v] [-f] -p param_value arg1 [arg2...]

Script description here.

...

EOF

* exit*

}

Данная функция решает две проблемы:

1. Отображает справочную информацию для пользователя, который хочет узнать все возможные и необходимые параметры для его запуска и не хочет изучать весь скрипт чтобы разобраться в них;

2. Имеет минимальную документацию на тот случай, когда кто-то будет вносить изменения в скрипт (например, даже вы сами две недели спустя, уже не помня, что и как было сделано).

Данный подход не является догмой и здесь необязательно нужно документировать каждую функцию. Но короткое красивое сообщение об использовании скрипта является обязательным минимумом и необходимым культурным моментом в оформлении сценариев.

Вывод сообщений

Функция msg() предназначена для печати всего, что не является непосредственно выводом скрипта. Сюда входят все журналирование и сообщения, а не только ошибки.

Пример использования:

msg "This is a ${RED}very important${NOFORMAT} message, but not a script output value!"

Анализ параметров

```
parse_params() {
  # default values of variables set from params
  flag=0
  param=''

  while :; do
    case "${1-}" in
    -h | --help) usage ;;
    -v | --verbose) set -x ;;
    --no-color) NO_COLOR=1 ;;
    -f | --flag) flag=1 ;; # example flag
    -p | --param) # example named parameter
      param="${2-}"
```

```
    shift
    ;;
  -?*) die "Unknown option: $1" ;;
  *) break ;;
  esac
  shift
 done

 args=("$@")

 # check required params and arguments
 [[ -z "${param-}" ]] && die "Missing required parameter: param"
 [[ ${#args[@]} -eq 0 ]] && die "Missing script arguments"

 return 0
}
```

Выполнение лабораторной работы

Разработать утилиту на командно-сценарном языке bash на базе шаблона из раздела №2 в соответствии с заданием на выполнение лабораторной работы. Финальную версию утилиты разместить по пути /usr/local/bin и присвоить имя соответствующее ее функциональному назначению, а также исключить расширение *.sh из имени утилиты.

Задание	Подробности
1. Установка и управление конфигурацией web-сервера Apache	Утилита должна реализовывать следующий функционал: – установку web-сервера; – конфигурирование виртуального хоста; – подключение модулей расширения; – размещение символических ссылок директорий сайтов в директорию /var/www; – справочную информацию (функция usage()) по использованию утилиты.
2. Формирование deb-пакета по заданной директории	Утилита должна реализовывать следующий функционал: – прием пути до папки, где находятся файлы для установки; – возможность указать список зависимостей (желательно в файле); – вывод журнала о ходе установки; – возможность задать ключевые параметры для «манифеста»; – размещение временных папок и файлов в системной директории /tmp; – справочная информация (функция usage()) по использованию утилиты.
3. Создание системного пользователя, назначение ACL и прав на каталоги и файлы, редактирование системных параметров	Утилита должна реализовывать следующий функционал: – создавать системного пользователя; – создавать системную группу; – назначать права и ACL на каталоги и файлы, в том числе посредством указания uid и gid; – задавать и редактировать имя хоста; – справочная информация (функция usage()) по использованию утилиты.
4. Установка и конфигурирование сетевой службы Samba (серверная часть)	Утилита должна реализовывать следующий функционал: – установка сетевой службы samba. Которая заканчивается созданием резервного файла конфигурации samba; • формирование конфигурации глобальных параметров по умолчанию: • **workgroup** - рабочая группа, как уже говорилось должна одинакова на всех машинах • **netbios name** - имя компьютера, которое будет отображаться в Windows; • **log file** - адрес файла, куда будут складываться сообщения об ошибках и другая информация; • **security** - по умолчанию выполнять аутентификацию на уровне пользователя; • **name resolve order** - очередность разрешения IP адресов по NetBIOS имени. bcast - означает отправить в локальную сеть широковещательный

	запрос. Если все компьютеры между которыми планируется взаимодействие находятся в одной сети этот вариант оптимальный; • **passdb backend** - способ хранения паролей пользователей; • **unix password sync** - синхронизация паролей пользователей samba с локальными паролями Unix; • **map to guest** - указывает, когда пользователю будет предоставляться гостевой доступ. Доступно три значения - **never** - никогда, **bad user** - когда такого пользователя не существует, **bad password** - когда пароль введен неверно, – задание конфигурации для сетевого доступа к каталогу; – создание каталога для сетевого доступа и назначение ему прав; – справочная информация (функция usage()) по использованию утилиты.
5. Установка и конфигурирование СУБД PostgreSQL	Утилита должна реализовывать следующий функционал: – установка СУБД PostgreSQL; – конфигурирование файла pg_hba.conf; – создание базы данных и формирование ее бэкапа; – создание схем и таблиц по имени БД; – создание таблиц должно предполагать задание столбцов и их типов данных; – создание роли и назначение прав; – справочная информация (функция usage()) по использованию утилиты.
6. Конфигурирование сетевых настроек операционной системы	Утилита должна реализовывать следующий функционал: – задание адресации, маски и шлюза на каждый сетевой интерфейс (работа с файлом /etc/network/interfaces); – задание статических маршрутов (при перезагрузке машины должны продолжать работать); – создание сетевого моста на сетевой интерфейс; – включение и отключение сетевых интерфейсов и сетевых мостов; – справочная информация (функция usage()) по использованию утилиты.
7. Создание iso-образа по заданной директории	Утилита должна реализовывать следующий функционал: – прием пути до папки, где находятся файлы для установки; – возможность менять атрибуты каталогов и файлов (время создания, права и т.д.); – справочная информация (функция usage()) по использованию утилиты.

8. Генерация HTML-страницы и подключение к ней мультимедийных ресурсов	Утилита должна реализовывать следующий функционал: – содержать в своем составе блоки заголовков, текстов, форм и т.д.; – возможность компоновать блоки в заданном порядке и формировать файл index.html; – устанавливать заданный текст; – формировать папки ./css, ./img, ./js и размещать в них таблицу разметки, изображения и javascript соответственно; – справочная информация (функция usage()) по использованию утилиты.
9. Управление компоновкой и компиляцией на основе утилиты make	Утилита должна реализовывать следующий функционал: – установка утилиты make, компилятора gcc и build-essential; – формирование make-файла; – возможность задать формат целевого файла исполнения; – справочная информация (функция usage()) по использованию утилиты.
10. Формирование файлового хранилища и метаданных к каталогам и файлам	Утилита должна реализовывать следующий функционал: – создание файла с правилами формирования директорий; – обход файлового хранилища и создания справочных файлов без расширения о файлах и каталогах; – включение в файл с правилами возможность задать типовые и тематические директории с шаблоном; – осуществление вывода информации по справочным файлам; – справочная информация (функция usage()) по использованию утилиты.
11. Установка и управление конфигурацией системы контроля версий исходных кодов git	Утилита должна реализовывать следующий функционал: – установка git; – создание репозитория с заданным набором пользователей; – создание ветки для каждого пользователя; – организация репозитория по модели git flow; – справочная информация (функция usage()) по использованию утилиты.
12. Генерация документов	Утилита должна реализовывать следующий функционал: – на основе утилиты find и ее аналогов разработать перечень функций по созданию, объединению, удалению, редактированию (расширений в том числе) файлов; – справочная информация (функция usage()) по использованию утилиты.

13. Регистрация команд, программ и служб в качестве сервиса	Утилита должна реализовывать следующий функционал: – создание сервиса systemd на основе скрипта .sh; – создание сервиса systemd на основе утилит, команд и служб; – справочная информация (функция usage()) по использованию утилиты.
14. Создание и управление lvm-томами	Утилита должна реализовывать следующий функционал: – инициализация физических lvm разделов; – создание группы разделов lvm; – создание логических томов lvm; – удаление логических томов lvm; – изменение размера lvm тома; – справочная информация (функция usage()) по использованию утилиты.
15. Конфигурирование кластеров ISCSI	Утилита должна реализовывать следующий функционал: – установка targetcli; – создание блочных устройств; – создание таргета; – создание LUN; – сохранение и применение конфигурации targetcli; – справочная информация (функция usage()) по использованию утилиты.
16. Установка и управление системой управления проектами на базе программного обеспечения redmine	Утилита должна реализовывать следующий функционал: – реализовать утилиту установки конфигурирования redmine на базе сценария приведенного во второй лабораторной работе; – справочная информация (функция usage()) по использованию утилиты.
17. Установка и управление конфигурацией web-сервера Nginx	Утилита должна реализовывать следующий функционал: – установку web-сервера; – конфигурирование виртуального хоста; – подключение модулей расширения; – размещение символических ссылок директорий сайтов в директорию /var/www; – рассмотреть вариант автоматизации сборки их исходных кодов (необязательно); – справочную информацию (функция usage()) по использованию утилиты.
18. Установка и управление сетевой службой NetworkManager	Утилита должна реализовывать следующий функционал: – задание адресации, маски и шлюза на каждый сетевой интерфейс; – задание статических маршрутов (при перезагрузке машины должны продолжать работать); – создание сетевого моста на сетевой интерфейс;

	– включение и отключение сетевых интерфейсов и сетевых мостов; – справочную информацию (функция usage()) по использованию утилиты.
19. Конфигурирование и управление системной службой управления заданиями	Утилита должна реализовывать следующий функционал: – гибкое задание периодов времени выполнения задания; – возможность поставить в задание как скрипт, так и отдельную команду; – задание пользователя, от имени которого будет выполняться задание; – справочную информацию (функция usage()) по использованию утилиты.
20. Конфигурирование и управление DHCP и DNS серверами	Утилита должна реализовывать следующий функционал: – настройка dhcp и dns должна быть возможна как в варианте со службой networking, так и NetworkManager; – справочную информацию (функция usage()) по использованию утилиты.

Подготовка к защите лабораторной работы

По результатам выполнения лабораторной работы должен быть оформлен отчет, описывающий следующие положения.

1. Назначение разработанной утилиты.
2. Список зависимостей утилиты и как их установить.
3. Детальное описание разработанных функций.
4. Расширенный вариант справки по утилите с описанием флагов, параметров и подпрограмм.
5. Исходный код утилиты.