DevOps external course

Basic shell

Lecture 4.2

Module 4 **Linux Essentials**

Serge Prykhodchenko

Agenda

- SSH in Virtualbox
- Linux file system
- FS commands
- Q&A

dnf

https://blog.eldernode.com/dnf-command-on-centos-8/

```
dnf search httpd
dnf install httpd
dnf reinstall httpd
dnf info httpd

dnf check-update
dnf update
dnf update httpd

dnf repolist all
dnf repolist
```

```
dnf remove httpd
dnf autoremove
dnf clean all
dnf groupremove 'System Tools'
```

SSH TO VIRTUALBOX

SSH Realization. Prerequisites

- VirtualBox ver. 5+
- Installed Ubuntu 16.04 (CentOS 7) with SSH server with
“Network”-> “Adapter” set to “Bridged Adapter”
- Clone(-s) of Installed Ubuntu 16.04 with SSH server (server edition) with
“Network”-> “Adapter” set to “Bridged Adapter” (full clone with reinitialized MAC-addresses)

SSH client :

- https://mobaxterm.mobatek.net/download.html (for Windows)

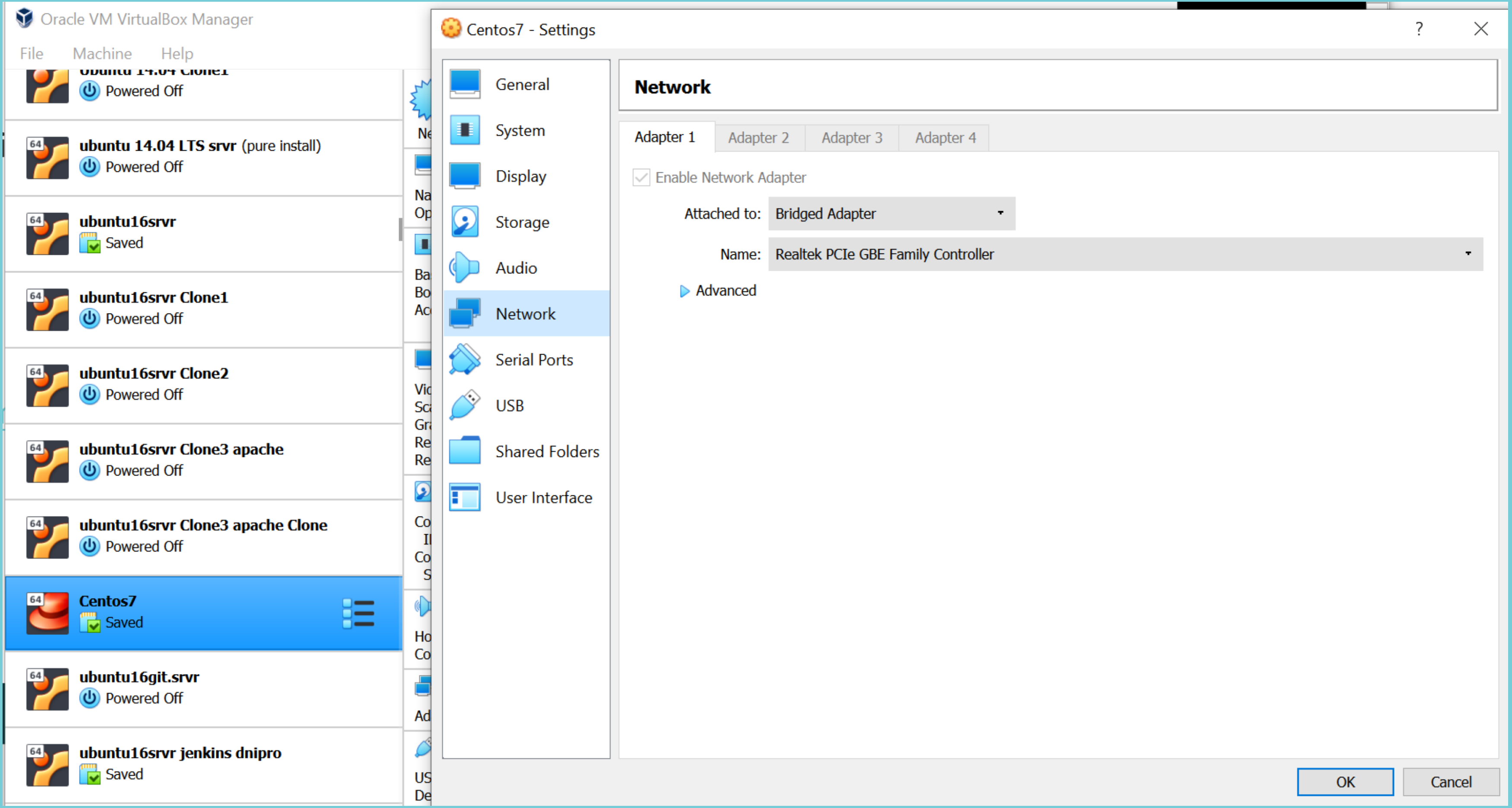
or

- https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html (Windows, Linux, but simple)

or

- https://www.termius.com/
(Windows, Linux, Mac)

“Network”-> “Adapter” set to “Bridged Adapter”



Oracle VM VirtualBox Manager

File Machine Help

ubuntu 14.04 Clone1 — Powered Off

ubuntu 14.04 LTS srvr (pure install) — Powered Off

ubuntu16srvr — Saved

ubuntu16srvr Clone1 — Powered Off

ubuntu16srvr Clone2 — Powered Off

ubuntu16srvr Clone3 apache — Powered Off

ubuntu16srvr Clone3 apache Clone — Powered Off

Centos7 — Saved

ubuntu16git.srvr — Powered Off

ubuntu16srvr jenkins dnipro — Saved

Centos7 - Settings

General

System

Display

Storage

Audio

Network

Serial Ports

USB

Shared Folders

User Interface

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

☑ Enable Network Adapter

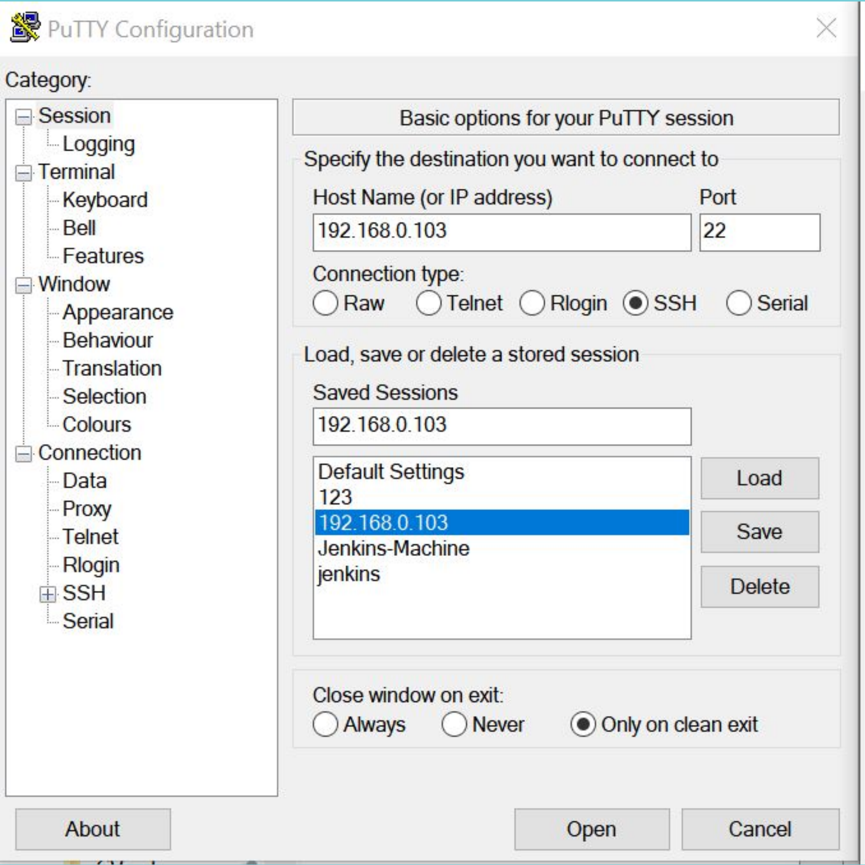
Attached to: Bridged Adapter

Name: Realtek PCIe GBE Family Controller

▷ Advanced

OK Cancel

Realization



Specify the destination you want to connect to

Host Name (or IP address): 192.168.0.103

Port: 22

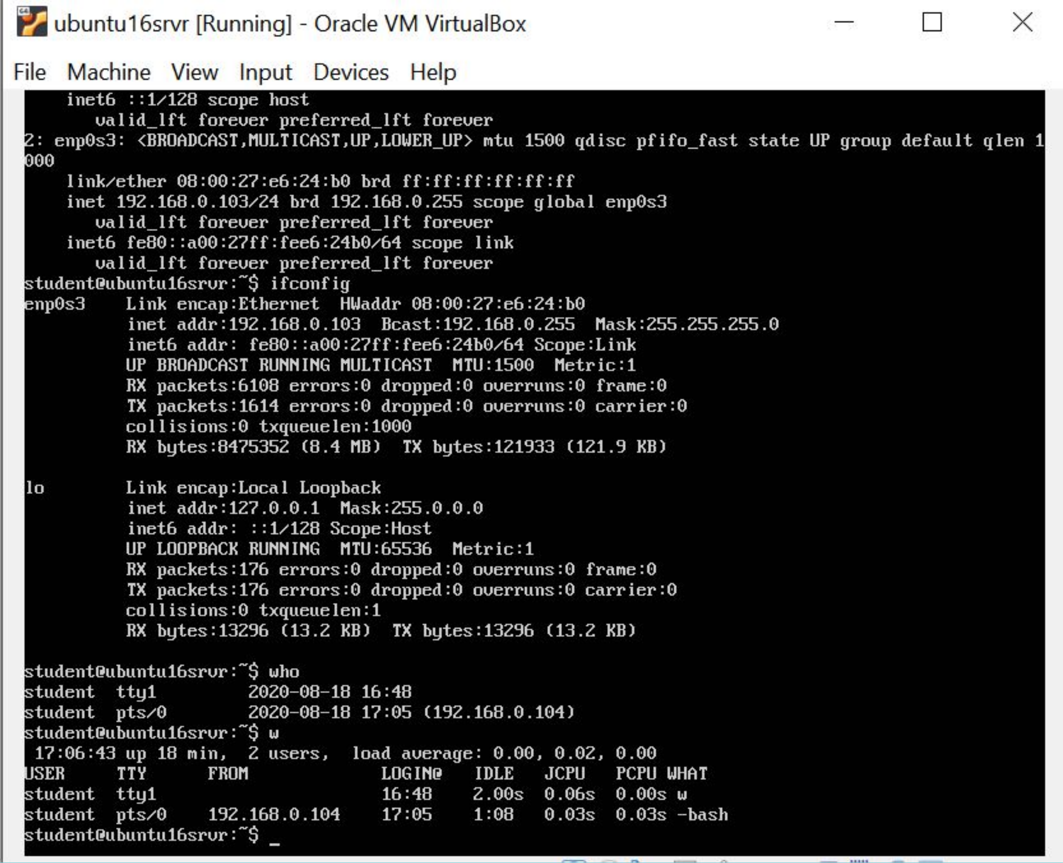
Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

Saved Sessions: 192.168.0.103

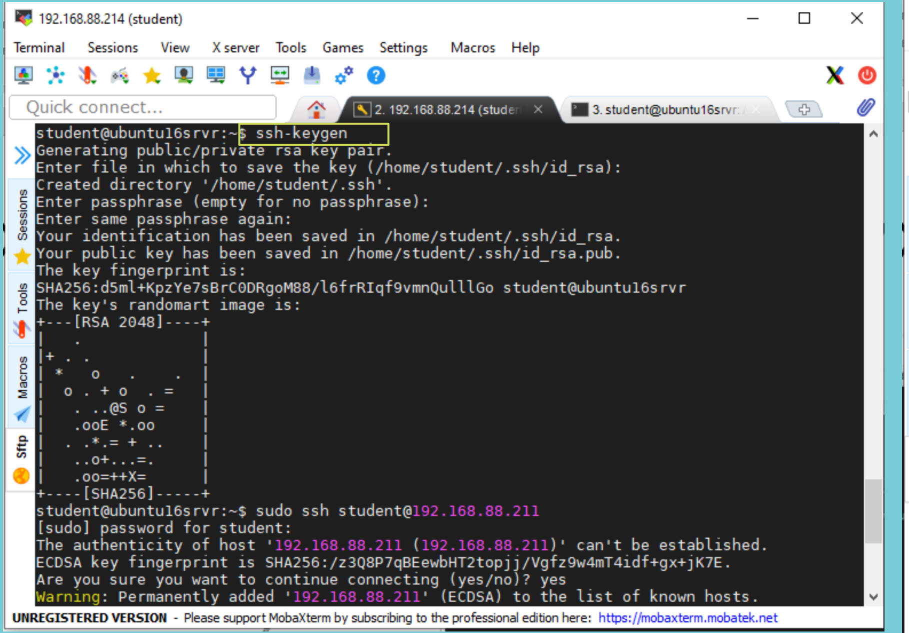
Default Settings
123
192.168.0.103
Jenkins-Machine
jenkins

Close window on exit: ○ Always ○ Never ● Only on clean exit



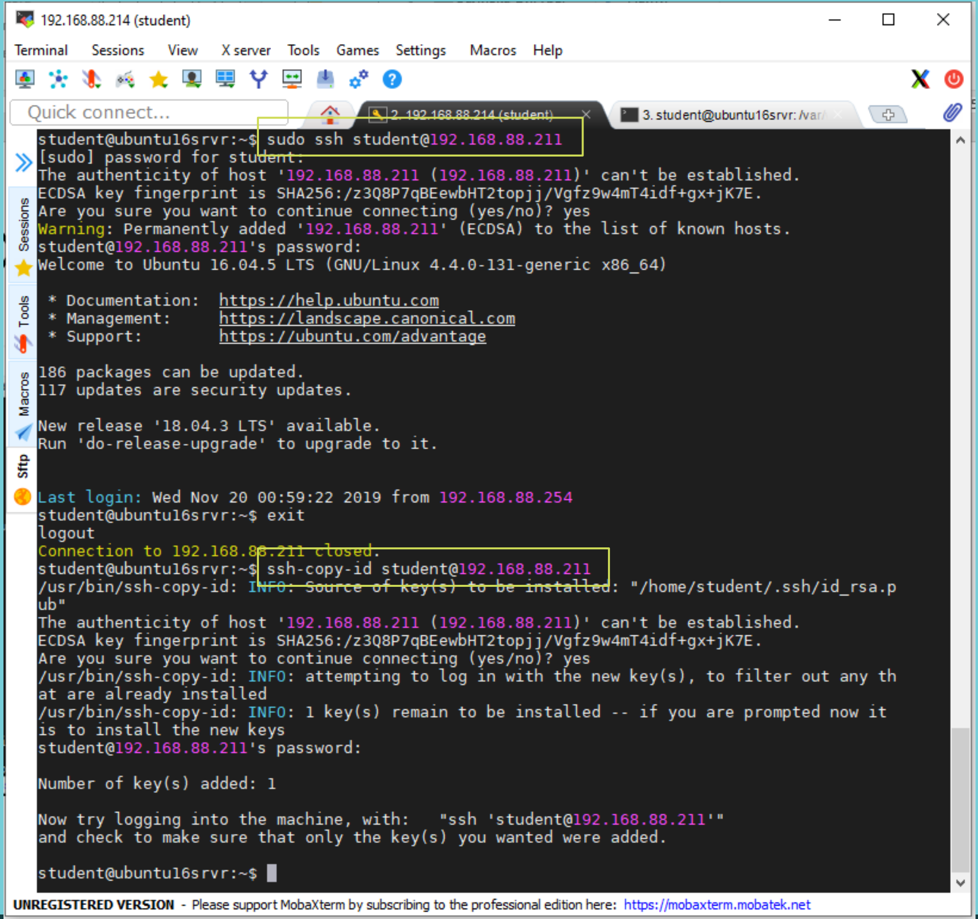
```
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1
000
    link/ether 08:00:27:e6:24:b0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.103/24 brd 192.168.0.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fee6:24b0/64 scope link
       valid_lft forever preferred_lft forever
student@ubuntu16srvr:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:e6:24:b0
          inet addr:192.168.0.103  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fee6:24b0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6108 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1614 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8475352 (8.4 MB)  TX bytes:121933 (121.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:176 errors:0 dropped:0 overruns:0 frame:0
          TX packets:176 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:13296 (13.2 KB)  TX bytes:13296 (13.2 KB)

student@ubuntu16srvr:~$ who
student  tty1         2020-08-18 16:48
student  pts/0        2020-08-18 17:05 (192.168.0.104)
student@ubuntu16srvr:~$ w
 17:06:43 up 18 min,  2 users,  load average: 0.00, 0.02, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
student  tty1                      16:48    2.00s  0.06s  0.00s w
student  pts/0    192.168.0.104    17:05    1:08   0.03s  0.03s -bash
student@ubuntu16srvr:~$ _
```

Establish SSH connection without login/password (using MobaXTerm)


192.168.88.214 (student)
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect...
2. 192.168.88.214 (studen
3. student@ubuntu16srvr:
Sessions
Tools
Macros
Sftp

```
student@ubuntu16srvr:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/student/.ssh/id_rsa):
Created directory '/home/student/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/student/.ssh/id_rsa.
Your public key has been saved in /home/student/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:d5ml+KpzYe7sBrC0DRgoM88/l6frRIqf9vmnQulllGo student@ubuntu16srvr
The key's randomart image is:
+---[RSA 2048]----+
|   .             |
|+ . .            |
| *   o   .    .  |
|  o . + o  . =   |
|   . ..@S o =    |
|   .ooE *.oo     |
|  . .*.= + ..    |
|   ..o+...=.     |
|   .oo=++X=      |
+----[SHA256]-----+
student@ubuntu16srvr:~$ sudo ssh student@192.168.88.211
[sudo] password for student:
The authenticity of host '192.168.88.211 (192.168.88.211)' can't be established.
ECDSA key fingerprint is SHA256:/z3Q8P7qBEewbHT2topjj/Vgfz9w4mT4idf+gx+jK7E.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.88.211' (ECDSA) to the list of known hosts.
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net


192.168.88.214 (student)
Terminal Sessions View X server Tools Games Settings Macros Help
Quick connect...
2. 192.168.88.214 (student)
3. student@ubuntu16srvr: /var/
Sessions
Tools
Macros
Sftp

```
student@ubuntu16srvr:~$ sudo ssh student@192.168.88.211
[sudo] password for student:
The authenticity of host '192.168.88.211 (192.168.88.211)' can't be established.
ECDSA key fingerprint is SHA256:/z3Q8P7qBEewbHT2topjj/Vgfz9w4mT4idf+gx+jK7E.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.88.211' (ECDSA) to the list of known hosts.
student@192.168.88.211's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

186 packages can be updated.
117 updates are security updates.

New release '18.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Wed Nov 20 00:59:22 2019 from 192.168.88.254
student@ubuntu16srvr:~$ exit
logout
Connection to 192.168.88.211 closed.
student@ubuntu16srvr:~$ ssh-copy-id student@192.168.88.211
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/student/.ssh/id_rsa.p
ub"
The authenticity of host '192.168.88.211 (192.168.88.211)' can't be established.
ECDSA key fingerprint is SHA256:/z3Q8P7qBEewbHT2topjj/Vgfz9w4mT4idf+gx+jK7E.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any th
at are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it
is to install the new keys
student@192.168.88.211's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'student@192.168.88.211'"
and check to make sure that only the key(s) you wanted were added.

student@ubuntu16srvr:~$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

LINUX FILE SYSTEM

man

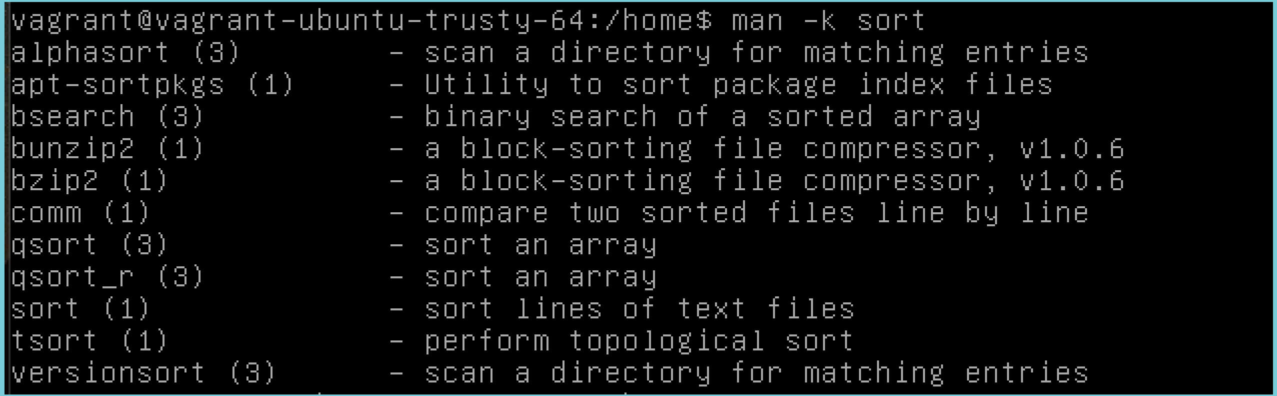
To view, for example, manual page concerns on ***ls*** command, type next:

$ man ls.

man-pages containing glossary with examples and cross-references.

For searching by keyword you'd use parameter **-k**:

$ man -k sort



```
vagrant@vagrant-ubuntu-trusty-64:/home$ man -k sort
alphasort (3)          - scan a directory for matching entries
apt-sortpkgs (1)       - Utility to sort package index files
bsearch (3)            - binary search of a sorted array
bunzip2 (1)            - a block-sorting file compressor, v1.0.6
bzip2 (1)              - a block-sorting file compressor, v1.0.6
comm (1)               - compare two sorted files line by line
qsort (3)              - sort an array
qsort_r (3)            - sort an array
sort (1)               - sort lines of text files
tsort (1)              - perform topological sort
versionsort (3)        - scan a directory for matching entries
```

Result contains page name, section number and a brief info.

Command man types first founded page. To read man about file **/etc/passwd** (not about command **passwd**), you'd use section number:

$ man 5 passwd

The structure of the Linux OS file system. Working with files and directories. Linux devices. File types.

Objective: To understand the Linux OS file system, its directory structure, basic commands for working with the file subsystem. Master the skills of navigating the operating system directories and working with devices.

The content of the lecture:

1. Organization of the file system. Directory tree.
2. Types of files and devices in Linux
3. Basic commands for navigating the file system (pwd, cd, dir, ls).
3. Basic commands for working with files (directories) (rm, mv, cp, mkdir, cat, less, more).
4. Soft and hard links (ln).
5. Organization of search (find, locate, grep).
6. Standard I / O and their redirection.
7. Working with disks (mount, umount).
8. Midnight Commander

The structure of the Linux OS file system.

File system - a method of storing and accessing data in the information carrier or its section. The classic file system has a hierarchical structure in which a file is uniquely identified by its full path to it.

Classification of file systems. By design, file systems can be classified into the following categories:

1) For media with random access (for example, a hard disk): FAT32, HPFS, ext2, etc. Since access to disks is several times slower than access to RAM, asynchronous writing of changes to disk is used to increase performance in many file systems ... For this purpose any logging, for example ext3, ReiserFS, JFS, NTFS, XFS, or soft_updates mechanism and others. Logging is widespread in Linux, used in NTFS. Soft_updates - on BSD systems.

2) For sequential access media (e.g., magnetic tape): QIC et al.

3) For optical media - CD and DVD: ISO9660, ISO9690, HFS, UDF, etc.

4) Virtual file systems: AEFS, etc.

5) Network file systems: NFS, SMBFS, SSHFS, GmailFS, etc

The structure of the Linux OS file system.

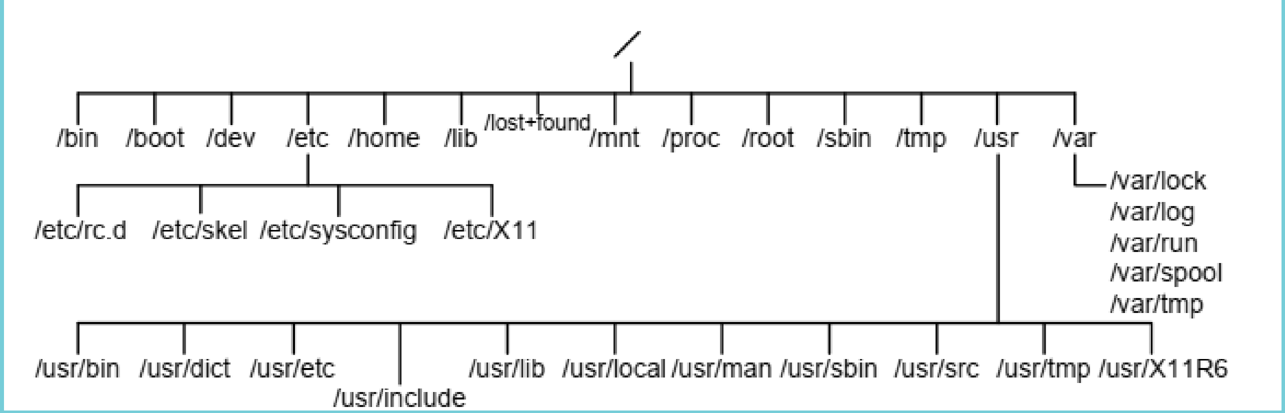
Catalog - a list of links to files or other directories. It is commonly said that a directory contains files or other directories, although in reality it only refers to them, the physical layout of data on disk usually has nothing to do with the location of the directory. A directory that is referenced in a given directory is called a subdirectory or subdirectory. The directory in the file system most of all resembles a library directory containing links to books and other sections of the directory (files and subdirectories) united by some characteristics.

The link to the same file can be contained in several directories at the same time – this makes access to the file more convenient. In the Ext2 file system, each directory is a separate file of a special type ("d", from the English "directory"), which differs from a regular data file: it can only contain links to other files and directories.

File and directory names. The main distinguishing features of files and directories are their names. On Linux, file and directory names can be up to 256 characters long, and can contain any characters other than "/". The reason for this limitation is obvious: this character is used as a name separator in the path, so it should not appear in the names themselves. And Linux always distinguishes between uppercase and lowercase letters in file and directory names, so "one", "One" and "ONE" will be three different words.

The structure of the Linux OS file system.

Directory structure. The concept of a catalog allows you to organize all objects located on a storage medium (for example, on a disk). Most modern file systems use a hierarchical data organization model: there is one directory that unites all data in the file system - this is the "root" of the entire file system, the root directory. The root directory can contain any objects of the file system, and in particular, subdirectories (directories of the first nesting level). Thus, everything that is written on the disk - files, directories and special files - necessarily "belongs" to the root directory: either directly (contained in it), or at some level of nesting. The structure of the file system can be visualized as a tree. The "root" of which is the root directory, and all other directories are located at the vertices.



TYPES OF FILES AND DEVICES IN LINUX

The structure of the Linux OS file system.

File types. From the operating system's point of view, a file is a stream of bytes. This approach allows you to extend the concept of a file to physical devices and some other objects. This makes it easier to organize and exchange data because data is written to a file, transferred to physical devices, and exchanged between processes in a similar way. All of these cases use the same approach based on the idea of a byte stream. Therefore, along with ordinary files and directories, files from the point of view of Linux are also:

- files of physical devices;
- named pipes;
- sockets;
-symbolic links (symlinks).

Physical device files. As already mentioned, from the point of view of Linux OS, all devices connected to the computer (hard and removable drives, terminal, printer, modem, etc.) are represented as files. If, for example, you need to display some information on the screen, then the system, as it were, writes to the file / dev / tty01

The names of the most commonly used special files (physical devices)

Name	Meaning
/dev/console /dev/hd	The system console, which is a monitor and keyboard that are physically connected to a computer. IDE hard drives. The /dev/hda1 device corresponds to the first partition on the first hard disk (/dev/hda), i.e. on the disk connected as Primary Master
/dev/sd	SCSI hard drives
/dev/fd /dev/tty /dev/pty	Floppy disk drive files. The first drive is /dev/fd0, the second is /dev/fd1. Support files for custom consoles. The name has survived since the days when teletypes were connected to UNIX as terminals. On Linux, these device files provide virtual consoles (you can switch between them with <Alt> + <F1> - <Alt> + <F6>). Pseudo-terminal support files. Used for remote work sessions using telnet
/dev/ttS	Files providing work with serial corresponds to COM1 in MS-DOS, /dev/ttS1 - COM2. If your mouse is connected via a serial port, then /dev/mouse is a symbolic link to the corresponding /dev/ttSN
/dev/null	This device is a "black hole". Anything written to / dev / null is lost forever. You can redirect the output of unnecessary messages to this device. If / dev / null is used as an input device, then it behaves like a zero-length file.

The structure of the Linux OS file system.

Extensions. Many users are familiar with the concept of extension - the part of a file name after a period, usually limited to a few characters and indicating the type of data contained in the file. In the Linux file system, there are no prescriptions about the extension: the file name can have any number of dots (including none), and after the last dot can be any number of characters. Although extensions are not required or imposed by technology in Linux, they are widely used: an extension allows a person or program, without opening a file, to determine what type of data it contains, only by its name. However, keep in mind that an extension is just a set of naming conventions for different types of files. Strictly speaking, the data in the file may not correspond to the declared extension for one reason or another, so you cannot completely rely on the extension. You can also determine the type of file content based on the data itself. Many formats provide for an indication at the beginning of the file how further information should be interpreted: as a program, source data for a text editor, HTML page, sound file, image, or something else. A Linux user always has the file utility at his disposal, which is designed specifically to determine the type of data contained in a file:

```
[globus@fedora ~]$ file test
test: ASCII English text
[globus@fedora ~]$ file /home/globus
/home/globus: directory
```

The structure of the Linux OS file system.

Named pipes. Another type of special file is named pipes, or First In - First Out (FIFO) buffers. Files of this type are mainly used to organize the exchange of data between different applications. A pipe is a very convenient and widely used means of exchanging information between processes. Anything that one process puts into a pipe, another can read from there. If two processes exchanging information are spawned by the same parent process (and this is most often the case), the pipe may be unnamed. Otherwise, you need to create a named pipe, which can be done using the ***mkfifo*** program.

Domain sockets. Sockets are connections between processes that allow them to communicate without being influenced by other processes. In general, sockets (and the interaction of programs using sockets) play a very important role in all Unix systems, including Linux: they are a key concept of TCP / IP and, accordingly, the Internet is built entirely on them. From a filesystem perspective, sockets are virtually indistinguishable from named pipes: they are simply labels that allow multiple programs to be linked together. After the connection is established, the programs communicate without the participation of the socket file: data is transferred by the OS kernel directly from one program to another. Examples of standard tools using sockets: X Window System, Printing System, and Syslog System

Links.

Symbolic links. A file in Linux can have multiple names or "hard links". A hard link is just another name for the original file. It is written in the inode of the source file. Once a hard link is created, it is impossible to distinguish between where the original file name is and where the link is. If you delete one of these files (more precisely one of these names), then the file is still saved on disk (as long as it has at least one link name).

Example:

[globus@fedora lab1]$ ln lab1.txt link_on_lab1

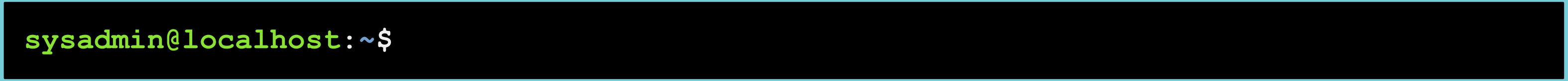
There is another type of links in Linux, so called **symbolic links**. These links can also be considered as additional filenames, but at the same time they are represented as separate files - files such as symbolic links. Unlike hard links, symbolic links can point to files located in a different file system, such as a mountable medium, or even on another computer. If the original file is deleted, the symbolic link is not removed, but is useless.

To create a symbolic link, use the **ln** command with the additional **-s** option:

ln -s name_of_file_or_directory_link_name

Shell

- When a terminal application is run, and a shell appears, displaying an important part of the interface – the prompt.
- Typically the prompt contains information about the user and the system. Below is a common prompt structure:



```
sysadmin@localhost:~$
```

- The prompt shown contains the following information:
  - Username (`sysadmin`)
  - System name (`localhost`)
  - Current Directory (`~`)

Commands

- Some commands require additional input to run correctly.

- This additional input comes in two forms: *options* and *arguments*.
  - Options are used to modify the core behavior of a command.
  - Arguments are used to provide additional information (such as a filename or a username).

- The typical format for a command is as follows:

```
command [options] [arguments]
```

Commands

```
command [options] [arguments]
```

- An argument can be used to specify something for the command to act upon.

- If the `ls` command is given the name of a directory as an argument, it lists the contents of that directory:

```
sysadmin@localhost:~$ ls /etc/ppp
ip-down.d  ip-up.d
```

- Some commands (such as `ls`) accept multiple arguments:

```
sysadmin@localhost:~$ ls /etc/ppp /etc/ssh
```

Commands

```
command [options] [arguments]
```

- Options can be used with commands to expand or modify the way a command behaves.
- For example, using the `-l` option of the `ls` command results in a *long listing*, providing additional information about the files that are listed.

```
sysadmin@localhost:~$ ls -l
total 0
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Desktop
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29  2015 Documents
Output Omitted...
```

- Often the character is chosen to be mnemonic for its purpose, like choosing the letter *l* for *long* or *r* for *reverse*.

Commands

- Options can be used in conjunction with other options:

```
sysadmin@localhost:~$ ls -lr
```

- Options are often single letters; however, sometimes they are words or phrases as well.
- Typically, older commands use single letters while newer commands use complete words for options.
  - Single-letter options are preceded by a single dash – character, like the `-h` option.
  - Full-word options are preceded by two dash -- characters like the full-word form of the `-h` option, the `--human-readable` option

Basic commands for navigating the file system (pwd, cd, dir, ls)

Each executable program "runs" in a strictly defined directory of the file system. This directory is called the current directory. Depending on the current directory, the behavior of the program can change: often the program will work by default with files located in the current directory - it will "reach" them first. Any program has the current directory, including the user's shell. Since the user's interaction with the system is necessarily mediated by the command shell, we can say that the user is "located" in the directory that is currently the current directory of his command shell. All commands issued by the user using the shell inherit the current shell directory, that is, they "work" in the same directory. For this reason, it is important for the user to know the current shell directory. The pwd utility is used for this:

```
[globus@fedora ~]$ pwd
/home/globus
```

The pwd (print working directory) command returns the full path of the current shell directory – of course, exactly the shell with which the pwd command was executed. In this case, globus learned that at this point (on this virtual console) the current directory is "/ home / globus".

Basic commands for navigating the file system (pwd, cd, dir, ls)

A **relative path** is a path to a file system object that does not begin at the root directory. Each Linux process has a specified current directory from which the system starts the relative path when performing file operations. A relative path is constructed in the same way as a full one - by listing through "/" all the directory names encountered when moving to the desired directory or file. There is only one significant difference between a full path and a relative path: a relative path starts from the current directory, while a full path always starts from the root directory.

Directory information (ls and dir). To be able to navigate the file system, you need to know what is contained in each directory. It is impossible and unnecessary to remember the entire structure of the file system: at any time you can view the contents of any directory using the ls utility (abbreviated from the English "list" - "list"):

```
[root@fedora globus]# ls
BUILD build.log
```

Using this command without parameters (keys) allows you to display the contents of the current directory

The ls Command

- Useful options for the “**ls**” command:
  - **ls -a** List all files, including hidden files beginning with a period “**.**”
  - **ls -ld *** List details about a directory and not its contents
  - **ls -F** Put an indicator character at the end of each name
  - **ls –l** Simple long listing
  - **ls –lR** Recursive long listing
  - **ls –lh** Give human readable file sizes
  - **ls –lS** Sort files by file size
  - **ls –lt** Sort files by modification time (very useful!)

Basic commands for working with files (directories) (rm, mv, cp, mkdir, touch; cat, less, more, tail, head).

Creating directories. The user, of course, does not have to keep all of his files in one directory. In the home directory, as in any other, you can create as many as you like, in them - your subdirectories, etc. In other words, the user owns a fragment (subtree) of the file system, the root of which is his home directory. To organize such a subtree of directories **/** subdirectories will need to create directories inside the home. The ***mkdir*** utility is used for this. It is used with one required parameter: the name of the directory to be created. By default, a new directory will be created in the current directory:

```
[globus@fedora ~]$ mkdir lab1
[globus@fedora ~]$ ls -F
BUILD/ build.log lab1/
```

Copying and moving files. The mv utility is used to move files and directories.

```
[globus@fedora ~]$ mv lab1.txt /home/globus/lab1
```

Files are copied using the cp utility, which requires the presence of two required parameters: the first is the file or directory to be copied, the second is the destination file or directory

```
[globus@fedora lab1]$ cp lab1.txt ../
```

Some Useful File Commands

- **cp** [file1] [file2] copy file
- **mkdir** [name] make directory
- **rmdir** [name] remove (empty) directory
- **mv** [file] [destination] move/rename file
- **rm** [file] remove (-r for recursive)
- **file** [file] identify file type
- **less** [file] page through file
- **head -n** [file] display first n lines
- **tail -n** [file] display last n lines
- **ln** –s [file] [new] create symbolic link
- **cat** [file] [file2…] display file(s)
- **tac** [file] [file2…] display file in reverse order
- **touch** [file] update modification time
- **od** [file] display file contents, esp. binary

Basic commands for working with files (directories)

Inodes. Since a file can have multiple names thanks to hard links, it is clear that all the essential information about a file in the file system is not tied to the name. In Linux file systems, all the information needed to work with a file is stored in an inode. There is an inode for each file: not only for regular files, but also for directories, and so on. Each file has one inode.

An inode is a description of a file that contains:

- file type (regular file, directory, etc.);
- file access rights;
- information about who owns the file;
- stamps about the time of creation, modification, last access to the file;
- file size;
- pointers to physical blocks on disk belonging to this file - these blocks store the "contents" of the file.

All inodes are numbered, so the inode number is a unique identifier for a file in the file system – as opposed to a file name (hard link to it), which can be multiple. You can find out the inode number of any file using the same ls utility with the -i switch:

```
[globus@fedora lab1]$ ls -i
559767 lab1.txt 559767 link_to_lab1
```

Searching

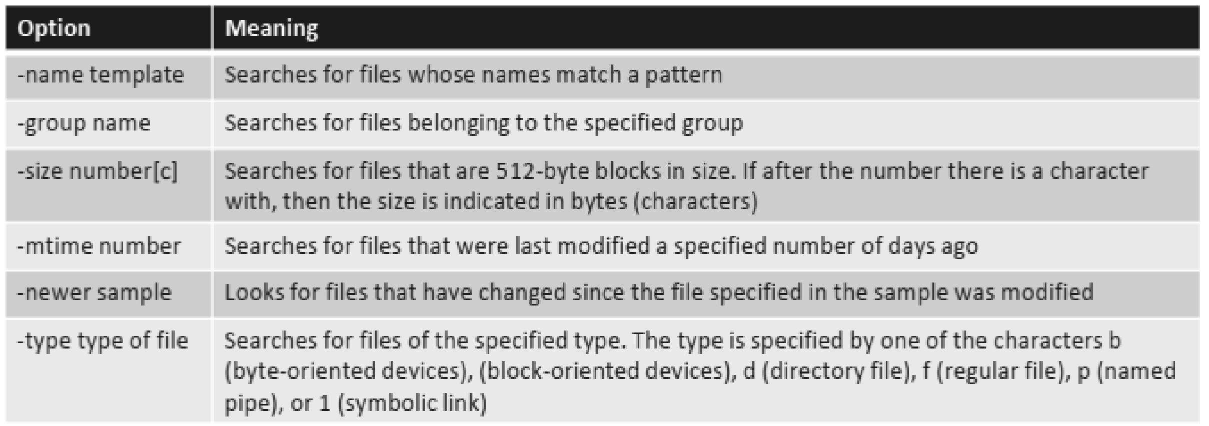
Organization of search. Another commonly used command for working with files on Linux is find. The find command can search for files by name, size, creation or modification date, and some other criteria. The general syntax for the find command is:

```
find [directory_list] search_criteria
```

The directory_list parameter determines where to find the file you want. The easiest way is to set the root directory / as the initial search directory, however, in this case, the search can take a very long time, since the entire directory structure will be scanned, including mounted file systems (including network files, if any). You can reduce the search volume if instead of one root directory you specify a list of several directories (of course, those in which the desired file may be located):

```
[globus@fedora ~]$ find /usr/share/doc /usr/doc /usr/locale/doc -name instr.txt
```

Examples of simple search criteria



Option	Meaning
-name template	Searches for files whose names match a pattern
-group name	Searches for files belonging to the specified group
-size number[c]	Searches for files that are 512-byte blocks in size. If after the number there is a character with, then the size is indicated in bytes (characters)
-mtime number	Searches for files that were last modified a specified number of days ago
-newer sample	Looks for files that have changed since the file specified in the sample was modified
-type type of file	Searches for files of the specified type. The type is specified by one of the characters b (byte-oriented devices), (block-oriented devices), d (directory file), f (regular file), p (named pipe), or 1 (symbolic link)

Searching

Locate command. Searches by filename. Before using this command, you must run the updated command as root, which will create an index of files on the disk; it is also recommended to run this command from time to time to update information about the file system. For more information, see the help system.

The grep utility. Utilities from the * grep family — fgrep, grep, and egrep — search for substrings in text strings. The first, fgrep (fast grep), does not use regular expressions; it simply reads text from standard input or from a file and prints lines containing the substring pattern specified on the command line to standard output. The grep utility interprets its first parameter as a base regular expression and prints out lines that contain the substring that matches this RV. The egrep utility works with extended PBs and is otherwise similar to grep. The grep utility searches text files for a pattern and prints out all lines containing that pattern. It uses a compact non-deterministic matching algorithm.

/usr/bin/grep [-bchilnsvw] restricted regex [filename ...]

Standard I / O. I / O streams.

When a program is launched for execution, it has three threads (or channels) at its disposal:

1. Standard input (standard input or stdin). On this channel, data is transmitted to the program;
2. Standard output (standard output or stdout). The program displays the results of its work through this channel;
3. The standard error message stream (standard error or stderr). Through this channel, programs issue information about errors.

The program can only read from standard input, and the other two streams can be used by the program only for writing. By default, the input stream is associated with the keyboard, and the output stream and error message stream are directed to the user's terminal.

Echo command. The echo command prints the character string given as an argument to standard output. After that, it issues a line feed signal and exits:

```
[root@fedora /]# echo "Test with echo"
Test with echo
```

Cat command. By default, the output of the cat command is directed to the output stream. To ensure that this command accepts input by default, run cat with no arguments. As a result, the cursor will move to a new line, and nothing more will happen. During this time, the command is waiting for the arrival of characters in the input stream. Enter any character, and you will see that it immediately appears on the screen, which indicates that the program immediately sent it to the output stream.

I/O redirection, pipes and filters.

Although usually, as mentioned, program I/O is associated with standard streams, there are special facilities in the shell to redirect I/O.

Operators **>**, **<** and **>>**. The symbols ">", "<" and ">>" are used to indicate redirection. The most common use is to redirect the command output to a file. Here's a relevant example:

```
[globus@fedora lab1]$ ls -l > test.out
```

This command will save a list of files and subdirectories of the directory that was current at the time of the ls command in the file /home/globus/lab1/test.out; in this case, if the specified file did not exist, then it will be created; if it existed, it will be overwritten; if you want the output of the command to be appended to the end of an existing file, you must use >> instead of the> symbol. In this case, the presence of spaces before or after the symbols> or >> is unimportant and serves only for the convenience of the user.

You can send output not only to a file, but also to the input of another command or to a device (such as a printer or terminal). This is quite convenient, for example, when debugging the system, the error code can be sent to another terminal:

```
[root@fedora ~]# top > /dev/tty6
```

Working with disks. Creation and mounting of file systems.

In the previous sections, we briefly reviewed the basic commands for working with an already formed file system. Now we need to dwell on the question of how to create a file system and modify it.

The general tree of files and directories of a Linux system is formed from separate "branches“ corresponding to different physical media. In UNIX there is no concept of "formatting a disk“ (and formatting command), but rather the concept of "creating a file system"

fd0	First floppy drive
fd1	Second floppy drive
hda	IDE hard drive / CD-ROM on the first IDE port (master)
hdb	IDE hard drive / CD-ROM on the first IDE port (slave)
hdc	IDE hard disk / CD-ROM on the second IDE port (master)
hdd	IDE hard disk / CD-ROM on the second IDE port (slave)
hda1	The first partition on the first IDE hard drive
hdd15	Fifteenth partition on the fourth IDE hard drive
sda	SCSI hard disk with the lowest SCSI ID (i.e. 0)
sdb	SCSI hard drive with next highest SCSI ID (i.e. 1)
sdc	SCSI hard drive with next highest SCSI ID (i.e. 2)
sda1	First partition on the first SCSI hard drive
sdd10	The tenth partition on the fourth hard disk SCSI
sr0	SCSI CD-ROM with the lowest SCSI ID
sr1	SCSI CD-ROM with the next highest SCSI ID
ttyS0	Serial port 0, COM1 under MS-DOS
ttyS1	Serial port 1, COM2 under MS-DOS
cdrom	Symbolic link to the CD-ROM drive
mouse	Symbolic link to the mouse device file

Working with disks. Creation and mounting of file systems.

In order to start working with a new file system (for example, to rewrite some files to a new medium), you need to connect it to a common directory tree, which is done using the mount command. The mount command requires at least a device and a "mount point" as parameters. A mount point is a directory in an existing and known directory tree that will now serve as the root directory for the mounted file system. Example:

```
[root]# mount /dev/hdb1 /mnt/disk2
```

connects the filesystem of the first partition on the second hard disk to the / mnt / disk2 directory (this directory must exist!).

Once the file system is mounted to / mnt / disk2, the previous contents of this directory will become inaccessible to you (as well as information about the previous owner and access rights to this directory) until you unmount the newly mounted file system. The old content is not destroyed, but simply becomes temporarily unavailable. Therefore, it is better to use empty directories (prepared in advance) as mount points.

By using the mount command without parameters, you can determine which filesystems are mounted.

Variables

- A variable is a feature that allows the user or the shell to store data.

- Variables are given names and stored temporarily in memory.

- There are two types of variables used in the Bash shell, *local* and *environment*.

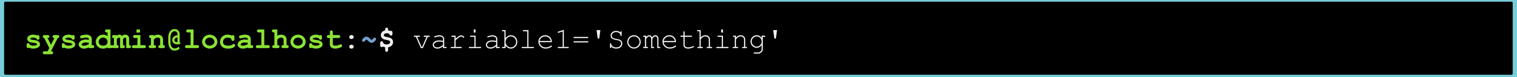
Local Variables

- Local or *shell*, variables exist only in the current shell. When the user closes a terminal window or shell, all of the variables are lost.

- To set the value of a variable, use the following assignment expression.

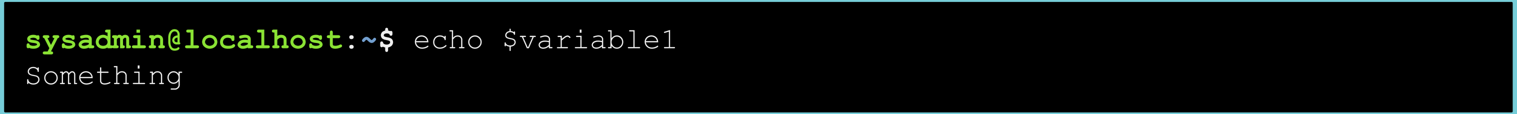
```
variable=value
```

- The following example creates a local variable named `variable1` and assigns it a value of `Something`:



```
sysadmin@localhost:~$ variable1='Something'
```

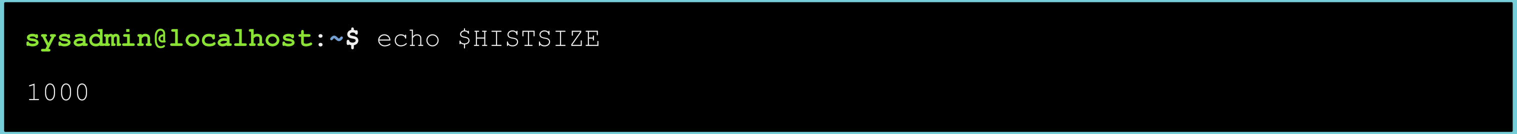
- To display the value of the variable, use a dollar sign `$` character followed by the variable name as an argument to the `echo` command:



```
sysadmin@localhost:~$ echo $variable1
Something
```

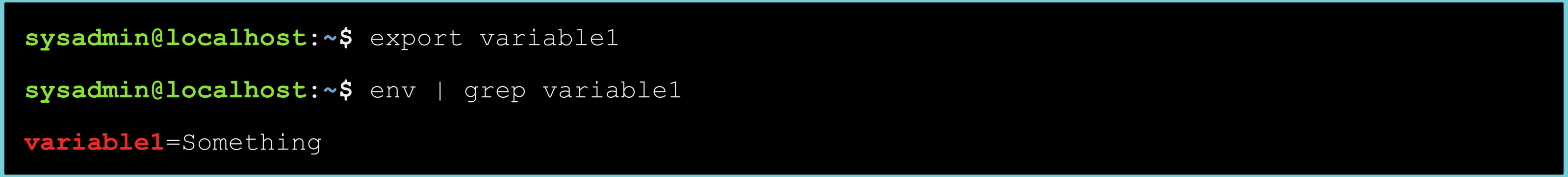
Environment Variables

- *Environment variables*, also called *global variables*, are available system-wide.
- Examples include the `PATH`, `HOME`, and `HISTSIZE` variables.
- The command in the example below displays the value of the `HISTSIZE` variable:



```
sysadmin@localhost:~$ echo $HISTSIZE
1000
```

- The `env` command outputs a list of the environment variables.
- The `export` command is used to turn a local variable into an environment variable.

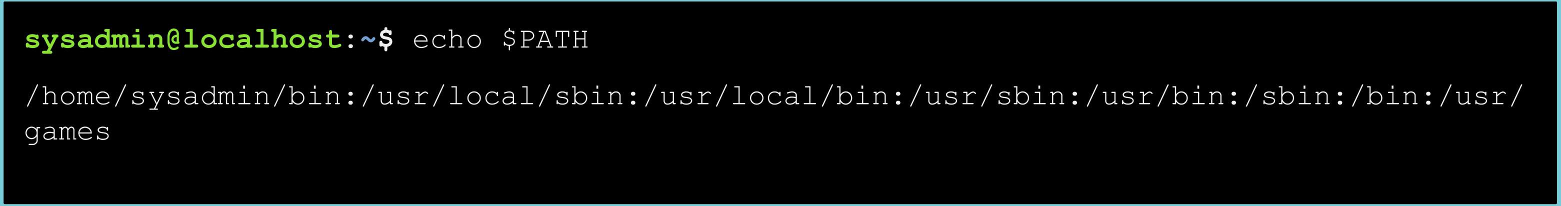


```
sysadmin@localhost:~$ export variable1
sysadmin@localhost:~$ env | grep variable1
variable1=Something
```

- Exported variables can be removed using the `unset` command:

Environment Variables

- One of the most important Bash shell variables to understand is the `PATH` variable.
- The `PATH` variable lists all the places that the system can look for programs to execute.
- The following command displays the path of the current shell:



```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games
```

- If the command is not found in any directory listed in the `PATH` variable, then the shell returns a `command not found` error.

Read-Only Variables

Including the keyword readonly before the command assignment prevents you from changing the variable afterwards
For example: `readonly phone="123-4567"`

After a variable is set, it can be protected from changing by using the readonly command

```
Syntax:         readonly variable
For example:    readonly phone
```

If no variable name is supplied a list of defined read only variables will be displayed

Removing Variables

For example:

```
course=
```

OR

```
unset course
```

Read-only variables cannot be removed – you must log out for them to be cleared

Variable Substitution

- Whenever you wish to use the value of a variable (its contents), use the variable name preceded by a dollar sign ($)
- This is called variable substitution
  - Example: name=Bob

    echo $name

Shell Configuration Files

Shell configuration files are scripts that are run when you log in, log out, or start a new shell

/etc/profile belongs to the root user and is the first start-up file that executes when you log in, regardless of shell

User-specific config files are in the user's home directory:

~/.bash_profile runs when you log in

~/.bashrc runs when you start an interactive subshell

~/.bash_logout runs when you log out

The start-up files can be used, for example, to:

- Set the prompt and screen display
- Create local variables
- Create temporary Linux commands (aliases)

Common Shell Variables

- Shell environment variables shape the working environment whenever you are logged in
- Common shell variables include:
  - PS1 – primary prompt
  - PWD – present working directory
  - HOME – absolute path to user's home
  - PATH – list of directories where executables are
  - HOST – name of the host
  - USER – name of the user logged in
  - SHELL – current shell
- The set command will display all available variables

Command Types

- The `type` command can be used to determine information about command type.

```
type command
```

- There are several different sources of commands within the shell of your CLI:
  - Internal commands
  - External commands
  - Aliases
  - Functions

Internal Commands

- Also called *built-in* commands, these commands are built into the shell itself.
- A good example is the `cd` (change directory) command as it is part of the Bash shell.
- The `type` command identifies the `cd` command as an internal command:

```
sysadmin@localhost:~$ type cd
cd is a shell builtin
```

External Commands

- *External commands* are stored in files that are searched by the shell.

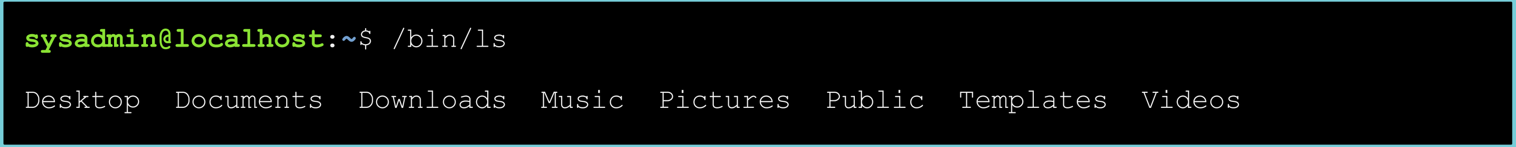
- It can be beneficial to know where the shell is finding the command or which version it is using.

- The `which` command searches for the location of a command by searching the `PATH` variable.

```
sysadmin@localhost:~$ which ls
/bin/ls
sysadmin@localhost:~$ which cal
/usr/bin/cal
```

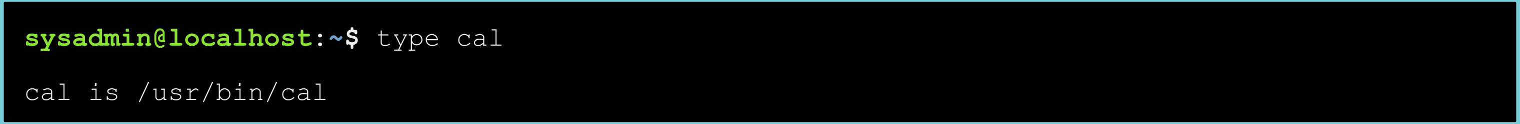
External Commands

- External commands can be executed by typing the complete path to the command.



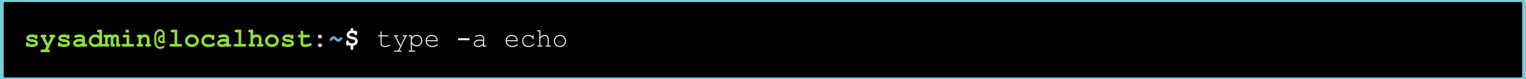
```
sysadmin@localhost:~$ /bin/ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

- For external commands, the `type` command displays the location of the command:



```
sysadmin@localhost:~$ type cal
cal is /usr/bin/cal
```

- To display all locations that contain the command name, use the `-a` option to the `type` command:



```
sysadmin@localhost:~$ type -a echo
```

Aliases

- An *alias* can be used to map longer commands to shorter key sequences.
- For example, the command `ls -l` is commonly aliased to `l` or `ll`.
- To determine what aliases are set on the current shell use the `alias` command:

```
sysadmin@localhost:~$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
Output Omitted...
```

- The `type` command can identify aliases to other commands:

```
sysadmin@localhost:~$ type ll
ll is aliased to `ls -alF'
```

Double Quotes

- Double quotes stop the shell from *interpreting* some metacharacters, including glob characters.

> Glob characters, also called wild cards, are symbols that have special meaning to the shell (i.e, `*`, `?`).

- This is useful when you want to display something on the screen that is normally a special character to the shell.
- In the example below, the Bash shell doesn't convert the glob pattern into filenames that match the pattern (like it normally does):

```
sysadmin@localhost:~$ echo "The glob characters are *, ? and [ ]"
The glob characters are *, ? and [ ]
```

- Double quotes still allow for *command substitution*, *variable substitution,* and permit some other shell metacharacters (i.e., the `PATH` variable)

Single Quotes

- Single quotes prevent the shell from doing any interpreting of special characters, including globs, variables, command substitution and other metacharacters.

```
sysadmin@localhost:~$ echo The car costs $100

The car costs 00

sysadmin@localhost:~$ echo 'The car costs $100'

The car costs $100
```

Backslash Character

- A technique to essentially single quote a single character is to use the backslash character `\`.
- If the phrase below is placed in single quotes, `$1`and `$PATH` are not variables:

```
sysadmin@localhost:~$ echo "The service costs $1 and the path is $PATH"

The service costs  and the path is
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
:/usr/games
```

- What if you want to have `$PATH` treated as a variable and `$1` not?
- In this case, use a backslash `\` character in front of the dollar sign `$` character to prevent the shell from interpreting it:

```
sysadmin@localhost:~$ echo The service costs \$1 and the path is $PATH

The service costs $1 and the path is
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
n:/usr/games
```

Backquotes

- Backquotes, or backticks, are used to specify a command within a command, a process called *command substitution*.

- Note the output of the `echo Today is date` command line:

```
sysadmin@localhost:~$ echo Today is date
Today is date
```

- To execute the `date` command so the output of that command is sent to the `echo` command, put the `date` command inside of two backquotes:

```
sysadmin@localhost:~$ echo Today is `date`
Today is Mon Nov 4 03:40:04 UTC 2030
```

Links

Advanced Bash-Scripting Guide - https://tldp.org/LDP/abs/html/

Advanced Bash-Scripting Guide (ru) - https://www.opennet.ru/docs/RUS/bash_scripting_guide/

QUESTIONS & ANSWERS

THANK YOU!