Linux file system

Files in Linux are organized in directories (analogous to folders in Windows). The root directory is simply "/". Users have their files in their home directories in "/home/". For example, my home directory (user name "jan") is "/home/jan/".

Special directory names: "./" refers to the current directory; "../" refers to the directory one level above the current directory; "~/" refers to your home directory.

File names: unlike Windows, Linux differentiates upper case and lower case letters in file names. That is, the file names "MyFile", "Myfile", "myfile", and "MYFILE" relate to four different files.

Hidden files: filenames that begin with "." (period) are hidden files. These are usually system files that don't show up when you list directory content. Under normal situations you don't have to worry about the hidden files. Just remember not to use "." at the start of a file name.

Files are assigned "permissions" that define who has access to them and what kinds of access. Basic types of access are read, write and execute. Read access allows you to read the content (e.g., make your own copy) of a file. Write access allows you to delete, modify or overwrite files.

Execute access is required to execute programs or for directories to be able to access their content. You have write access only to your home directory, that is, unless specifically given access you will not be able to store files elsewhere on the file system. At the same time, you have read and execute (but not write) access to all system files and programs that you will need. File permissions can be changed with the "chmod" command but it is unlikely that you will need it in this course.

Linux does not use extensions to recognize the type of a file (like Windows) but you can include "." in file names. I often give files Windows-like extensions (like .txt, .pdf, .bat, etc.) in order to remind myself what kind of a file it is and also that I don't have to change the extension when I transfer the file to Windows.

I recommend that you use directories to keep your files organized.

Wildcards: Many Linux commands allow wildcards in the file names. Most useful are "*" (matches any text) and "?" (matches any single character). For example, "*" matches all files in a directory, "a*" matches all files that start with "a", "a*z" matches all files that start with "a" and end with "z", "a?z" will match things like "atz", "a2z", "a.z" but not "a2.z", ????? will mach all files with names exactly 5 characters long. You can use it with directories, too, e.g., the command "ls ~/*/*.txt" (see also below) will display file names of all files ending with ".txt" in directories one level inside your home directory.

Linux Shell

Linux commands have none, one or more parameters. For example, "ls /home/jan" will list the content of my home directory ("/home/jan" is a parameter). "ls" with no parameters will list content of the current directory. "cp file1 file2" will read "file1" and make a copy named "file2". As you see, the order of the parameters is usually significant.

In addition to parameters, most Linux commands have options. Options start with "-". For example, "ls -l /home/jan" will list additional information about each file (without the -l it will only list the file names). Options modify behavior of the command.

There are several types of shells that have some minor differences. The shell we use is called "bash".

List of Basic Linux shell commands

Following is just the very basic list of some useful commands. More can be found on the internet (e.g., type "linux shell commands" in Google). Check http://linux.org.mt/article/terminal for a beginner's tutorial including some more advanced tricks.

Ctrl-C: pressing ctrl-C will stop the running program or command

Output redirection: ">" will redirect the output normally going on the screen into a file. E.g., "ls > List" will list all files in the directory but store it in a file named "List" instead of displaying it on the screen. This can be used with any command or program.

Files and directories

- pwd: shows current directory.
- cd directoryname: makes directoryname your current directory. cd with no parameters
- switches to your home directory
- ls directoryname: lists contents of directories. Use ls -l for more information about the files. You can limit the list with wildcards (e.g., "ls /home/mydirectory/*.txt")
- mkdir directoryname: creates a new directory.
- cp source destination: makes a copy of a file named "source" to "destination".
- cp –r source destination: copies a directory and its content
- mv source destination: moves a file or directory.
- rm filenamelist: removes/deletes file(s). Be careful with wildcards.
- rm –r directory: removes directory (-ies) including its content

Viewing files

- cat file: prints the whole file on the screen. Can be used with redirection, e.g., "cat Small1 Small2 Small3 > Big" will concatenate files Small1, 2 and 3 into a single file named Big. You can also use wildcards, e.g., "cat *.txt > VeryBigFile".

- more file: shows a file page by page. When viewing the files press “Enter” to move forward one line, “Space” to move forward one page, “b” to move one page backward, “q” to end viewing the file, type “/patern” to jump to the next occurrence of the text “pattern”. These are just few things you can do with more.

Getting more information about commands

- man command: will show the manual page (complete reference) for the command. Scroll up and down as in more.
- info command: similar to man but contains more verbose information.
- Many commands will show basic description when run with -h or --help option.

Other

- passwd: change your password
- exit: close the terminal
- chmod: change permissions for a file/directory. This allows you to set access to your files for other users or turn a text file into an executable. You will hardly need it in this course.
- gcc program.c –o ~/bin/program: compiles a program written in C language and creates an executable file in your bin directory. ~/bin/ is a standard place for storing executable (binary) files. –lm option has to be included if the program uses math library. –O option will optimize the program (make it run faster).

Editing

You can use command-line editors like vi, emacs or ed to modify contents of text files. However, they take a while to get used to and unless you already know them you may be better off using sftp to open the file in Notepad on your (Windows) computer. Just make sure that the sftp is set to ASCII (text) mode when you do this.

Running programs

Once created, executable programs can be run as any regular command just by typing the program name and any parameters and/or options. In fact, the shell commands are programs. Executable program files can be created in different ways (included for your reference):

- When written in a language like C, C++, or Fortran, the “source code” (human-readable text file prepared by a programmer in an appropriate syntax) can be converted into an “executable” (machine-readable binary file) by a special program called compiler (see cc command above). Compiling complex program packages may be complicated and programmers often create “make” files, which contain all instructions how the program should be compiled and assembled in a single file and are executed with the “make”command. Program packages are increasingly often distributed as rpm files where the installation is completely automatic making it very easy for the user (generally only the system administrator can install such packages).

- You can create shell scripts (text files containing lists of commands to be executed) and give them executable permission. That will effectively turn them into executable programs.
- Interpreted languages such as Perl differ from compiled ones in that you do not prepare an executable file but instead each command is "interpreted" at the time of execution. This makes such programs slower to run but eliminates need for compilation. Perl programs (scripts) are run by typing "perl program" followed by parameters and options.

File and path names

There are two ways to reference a file in Linux: using a relative path name or using an absolute path name. An absolute path name always begins with a / and names every directory on the path from the root to the file in question. For example, in the figure above, the konsole file has the absolute path name /usr/bin/konsole. A relative path names a path relative the current working directory.

The current working directory is set using the cd command. For example, if the current working directory is /usr, then the konsole file could be referenced with the name bin/konsole. Note that there is no leading /. If the current working directory were /usr/share, then konsole could be referenced with ../bin/konsole. The special name ".." is used to reference the directory above the current working directory.

File system permissions

Linux, as many other modern operating systems have methods of administrating permissions or access rights to individual or groups of users. These permissions affect how users can make changes to the file system.

There are three groups of permissions on Linux type systems: "user", "group" and "others". The "user" group grants permissions for the owner of a file or directory. The "group" group grants permissions for members of the file or directory's group and the "others" group grants permissions for all other users. Each group can have three main permission bits set "read", "write" or "executable".

The read bit (r bit) grants permission to read a file or directory tree. The write bit (w bit) grants permission to modify a file. If this is set for a directory it grants permission to modify the directory tree, including creating or (re)moving files in the directory or changing their permissions. Finally, the executable bit (x bit) grants permission to execute a file. The x bit must be set in order for any file to be executed or run on a system (even if the file is a executable binary). If the x bit is set on a directory, it grants the ability to traverse the directory tree.

To list the permissions of a file or directory, use the ls command with the -l option (to enable long file listing; see the man page for ls). For example, to see the permissions set for the file "foobar" in the current directory has, write:

% ls -l foobar

-rwxr-xr-- 1 john users 64 May 26 09:55 foobar

Each group of permissions is represented by three characters in the leftmost column of the listing. The very first character indicates the type of the file, and is not related to permissions. The next three characters (in this case rwx) represent user permissions. The following three (in this case r-x) represent group permissions and the final three represent permissions for others (in this case r--).

The owner and group of the file are given by the third and fourth column, respectively (user john and group users in this example). In this example the owner, "john", is allowed to read, write and execute the file (rwx). Users belonging to the group "users" are allowed to read and execute the file (r-x), but cannot write to it. All other users are allowed to read foobar (r--), but not write or execute it.

File types

The first character, the type field, indicates the file type. In the example above the file type is "-", which indicates a regular file. Other file types include: d for directory, l (lower case ell) for symbolic link, s for linux domain socket, p for named pipe, c for character device file and b for block device file.

Manipulating access rights

The chmod and chown commands are used to manipulating permissions. chmod is used to manipulate individual permissions. Permissions can be specified using either "long" format or a numeric mode (all permission bits together are called the files mode). The long format takes a string of permission values (r, w or x) together with a plus or minus sign. For example, to prevent any user from changing foobar we would do as follows to disable write permission, then verify that the change has taken place:

% chmod -w foobar

% ls –l foobar

-r-xr-xr-x 1 john users 81 May 26 10:43 foobar

In numeric mode, each permission is treated as a single bit value. The read permission has value 4, write value 2 and execute value 1. The mode is a three character octal string where the first digit contains the sum of the user permissions, the second the sum of the group permissions and the third the sum of the others permissions. For example, to set the permission string "-rwxrw-r--" (user may do anything, group may read or write, but not execute and all others may read) for a file, you would calculate the mode as follows:

User: 4+2+1= 7 (rwx)

Group: 4+2 = 6 (rw-)

Others: 4 = 4 (r--)

Together with chmod the string “764” can then be used to set the file permissions:

```
% chmod 764 foobar

% ls -l foobar

-rwxrw-r-- 1 john users 81 May 26 10:43 foobar
```

Numeric combinations are generally quicker to work with once you learn them, especially when making more complicated changes to files and directories. Therefore, you are encouraged to use them. It is useful to learn a few common modes by heart:

755 Full rights to user, execute and read rights to others. Typically used for executables.

644 Read and write rights to user, read to others. Typically used for regular files.

777 Read, write and execute rights to everybody. Rarely used.

chown

chown is used to change the owner and group for a file. To change the user from “john” to “mike” and the group from “users” to “wheel” issue:

```
% chown mike:wheel foobar
```

Note that some linux systems do not support changing the group with chown. On these systems, use chgrp to change tile file’s group. Changing owner of a file can only be done by privileged users such as root. Unprivileged users can change the group of a file to any group they are a member of. Privileged users can alter the group arbitrarily.

Symbolic links

In Linux, it is possible to create a special file called a symbolic link that points to another file, the target file, allowing the target file to be accessed through the name of the special file. Similar functions exist in other operating systems, under different names (e.g. “shortcut” or “alias”). For example, to make it possible to access /etc/init.d/myservice as /etc/rc2.d/S98myservice, you would issue the following command:

```
% ln –s /etc/init.d/myservice /etc/rc2.d/S98myservice
```

Symbolic links can point to any type of file or directory, and are mostly transparent to applications. Linux also supports a concept called “hard linking”, which makes it possible to give a file several different names (possibly in different directories) that are entirely equal (i.e. there is no concept of “target”, as all names are equally valid for the file).

The Linux boot process

When the Linux kernel loads, it starts a single user process: init. The init process in turn is responsible for starting all other user processes. This makes the Linux boot process highly configurable since it is possible to configure the default init program, or even replace it with something entirely different.

The init process reacts to changes in run level. Run levels define operating modes of the system. Example include “single user mode” (only root can log in), “multi-user mode with networking”, “reboot” and “power off”. In Debian/Gnu Linux, the default run level is run level 2. Other Linux distributions and other Unix-like systems may use different default run levels.

The actions taken by init when the run level changes are defined in the /etc/inittab file. The default configuration in most Linux distributions is to use something called “System V init” to manage user processes. When using System V init, init will run all scripts that are stored in a special directory corresponding to the current run level, named /etc/rcN.d, where N is the run level. For example, when entering run level 2, init will run all scripts in /etc/rc2.d.

Scripts are named SNNservice or KNNservice. Scripts whose names start with K are kill scripts and scripts whose names start with S are start scripts. When entering a run level, init first runs all kill scripts with the single argument stop, and then all start scripts with the single argument start. For example if the directory /etc/rc5.d contains the following scripts: K10nis, K20nfs and S10afs, init would first execute /etc/rc5.d/K10nis stop, then /etc/rc5.d/K20nfs stop, then /etc/rc5.d/S10afs start.

When Linux boots it start by changing to run level S (single user mode), then to run level 2. This implies that all scripts in /etc/rcS.d and in /etc/rc2.d are run when the system boots, and more importantly that all services that are started are started by scripts in one of these directories.

In Debian/Gnu Linux, all the scripts in /etc/rcN.d are actually symbolic links to scripts in /etc/init.d. For example, /etc/rc2.d/S20ssh is a symbolic link pointing to /etc/init.d/ssh. This is so that changes to the scripts need to be made in a single file, not in one file per run level. It also means that if you want to start or stop a service manually, you can use the script in /etc/init.d rather than try to remember its exact name in any particular run level.

/etc/init.d/SERVICE start

Start the service named SERVICE (e.g. ssh, nis, postfix).

/etc/init.d/SERVICE stop

Stop the service named SERVICE.

/etc/init.d/SERVICE restart

Restart SERVICE (roughly equivalent to stopping then starting).

/etc/init.d/SERVICE reload

Reload configuration for SERVICE (does not work with all services).

Sometimes it is useful to see exactly how a service is started or stopped (e.g. when startup fails). To see all the commands run when a service starts, run the script using the sh –x command (works for nearly all startup scripts, but is not guaranteed to always work).

sh –x /etc/init.d/SERVICE start

Start SERVICE, displaying each command that is executed.

Debian/Gnu Linux includes a command named update-rc.d that can be used to manipulate start scripts.

System logging

System logs are some of the most important source of information when troubleshooting a problem, or when testing a system. Most Unix services print diagnostic information to the system logs.

Logging is managed by the syslogd process, which is accessed through a standard API. By default, the syslog process outputs log messages to various log files in /var/log, but it is also possible to send log messages over the network to another machine. It is also possible to configure exactly which log messages are sent to which files, and which are simply ignored.

For the purpose of this course, the default configuration is sufficient. It creates a number of log files, the most important of which are: /var/log/auth.log for log messages related to authentication (e.g. logins and logouts); /var/log/syslog and /var/log/messages contain most other messages;

mail.log contains log messages from the mail subsystem. For details on what goes where, see /etc/syslog.conf. Since log files grow all the time, there needs to be a facility to remove old logs.

In Debian/Gnu Linux, a service called logrotate is commonly used. It “rotates” log files regularly, creating a series of numbered log files, some of which are compressed. For example, you may see the files /var/log/auth.log, /var/log/auth.log.0, /var/log/auth.log.1.gz and /var/log/auth.log.2.gz on a system. /var/log/auth.log is the current log file. /var/log/auth.log.0 is the next most recent and so forth.