SSH (SecureSHell) and SFTP (Secure FTP)

Rev: 2020
David Nichols

Contents

Purpose of the Document
Section I: SSH & SFTP
Section II: Using SFTP Clients
Section III: Using SSH Clients and Connections
Appendix – Useful Linux Commands

Purpose of the Document

The purpose of this guide is to provide step-by-step instructions for connecting remotely to various systems in the Sociolinguistics Laboratory, including the desktop computers (Peladon, Astrid, Chesterton) as well as the file servers (Zeos). This entails using SSH, FileZilla (for Macs) or MobaXterm (for Windows), and a remote shell account.

This document has been prepared to allow lab access at a time when physical laboratory access has been restricted, as part of the lab's COVID-19 Return to Research plan.

Note: This document assumes users already have permissions, logins and passwords for accessing the lab computing environment. The examples below, therefore, do not use active login credentials, for the sake of the security of the lab's computing system. If you do not have a login or password for a particular project, remember that you can use the generic lab user account for all of the lab's desktop computers. Please contact the lab director, computing specialist or lab SA if you have questions.

When should I think of using this document?

- To access files on the lab computers instead of coming to campus
- To backup my generals paper or dissertation research to Zeos
- To better understand SSH and SFTP before running p2fa on Peladon
- To remotely run a Praat script over a large amount of audio data

Section I: SSH & SFTP

What are SSH & SFTP?

Secure Shell or SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices. It is used primarily on Linux and Unix based systems to access shell accounts. A shell account is a personal account that gives a user access to a Unix (or Linux) shell on a remote server. A shell account can be used for many different purposes because many different programs can be run on the shell. Access a shell account via SSH in order to run Unix or Linux commands (such as changing file permissions). By using a client such as FileZilla or MobaXterm for SFTP in order to transfer files, you're already using SSH – SFTP is secure FTP (File Transfer Protocol), or FTP over SSH.

SFTP uses the non-secure method of FTP over the secure SSH channel. Without the encryption of SSH (the "secure" portion of secure shell), any files can be read simply by listening to the traffic between systems. This includes username / password pairs, which could give bad actors control an exposed account.

A brief word of warning: **deleting files in Linux, whether via SFTP or SSH, is permanent**. There are no do-overs.

Programs for SFTP:

The apps recommended for lab users are listed below. Ensure that you have installed the one appropriate for your operating system.

- Windows: MobaXterm
  - https://mobaxterm.mobatek.net/
- Mac: FileZilla
  - https://filezella-project.org/

Section II: Using SFTP Clients

Windows

The recommended client for using SFTP in Windows is MobaXterm. It can serve as both a terminal (for SSH) and as a graphical SFTP client.

1. Open MobaXterm and click the "Start local terminal" button to start a new session (Fig. 1).

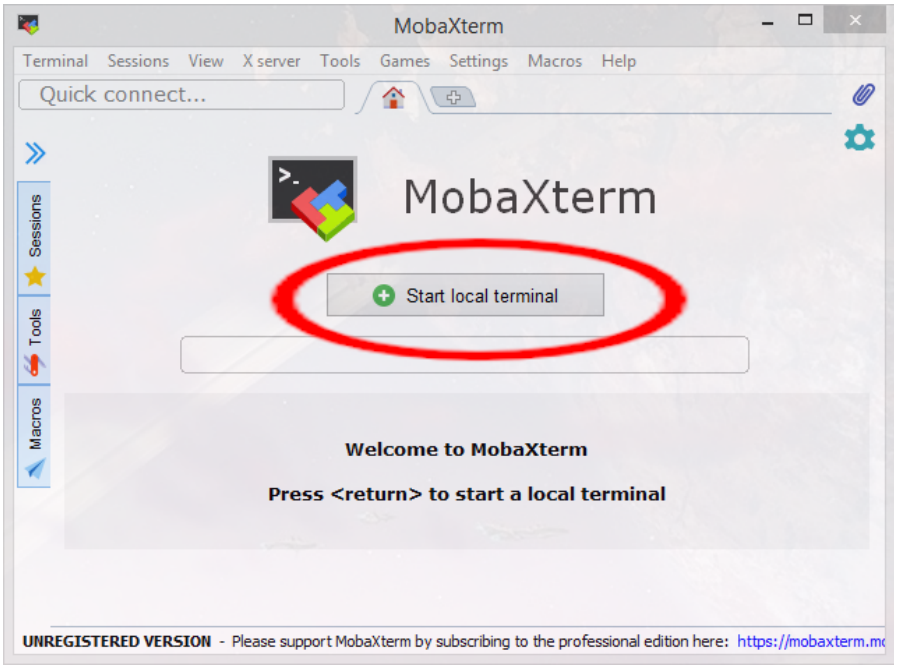

Figure 1: MobaXterm Start-up Window

2. Use SSH to connect to a remote host (Fig. 2).

   i. The command is:

   ssh <username>@<hostname>

   ii. By default, the username is the username for the (local) computer account on which MobaXterm is being run. (In Fig. 2 the default username is ulfgard.) **-l p2fa** selects the username p2fa; **-l <username> <hostname>** is functionally equivalent to **<username>@<hostname>**.

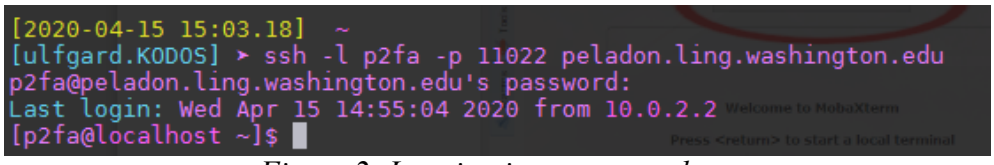

Figure 2: Logging into a remote host

   iii. Flag **-p** specifies the port to be used for the SSH connection. The default port for ssh is 22. This will rarely change, but in some cases, the ssh server will be

configured to use a different port. In the above example (Fig. 2), **-p 11022** selects port 11022.

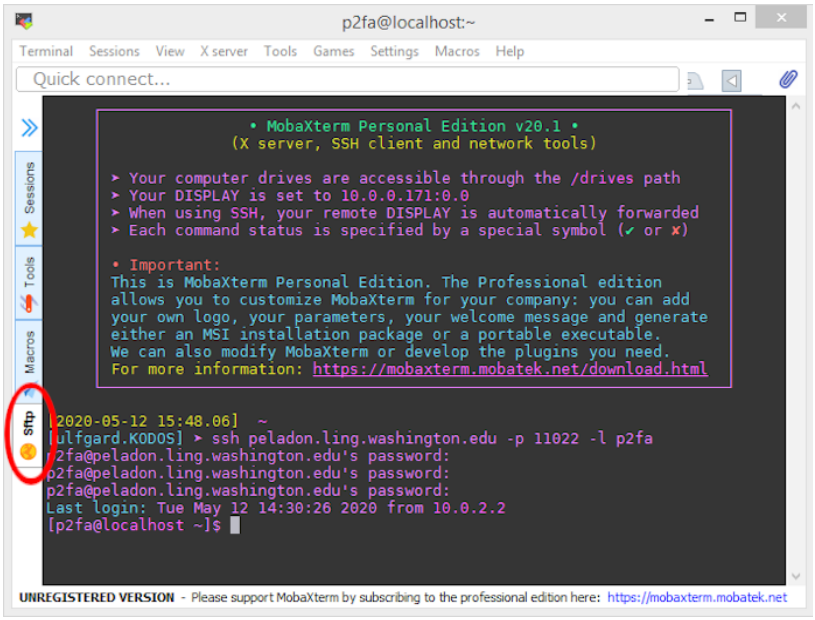

Figure 3: MobaXterm SFTP Tab

3. Select the SFTP tab (in the red ellipse, Fig. 3). It typically opens automatically upon login. If it only works as a mouse-over, then click on the double arrow at the top of the sidebar. This will show the file structure for the account.

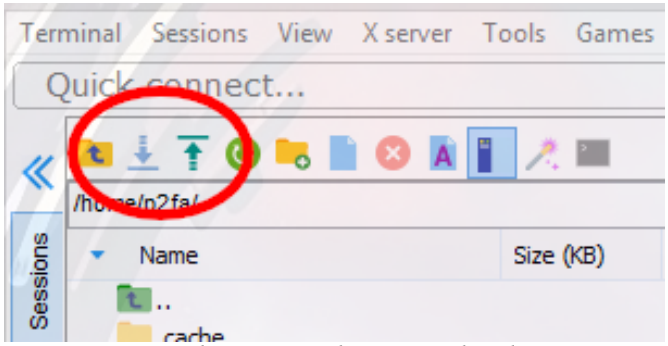

Figure 4 MobaXterm Blue Download & Green Upload Arrows

4. To download a file: select the desired file(s) in the SFTP tab. Click the blue arrow (Fig. 4) “Download selected files” in the sidebar to choose the local destination, or drag and drop to your desired window. A download destination menu will pop up in order to choose a location for the download (Fig. 5).

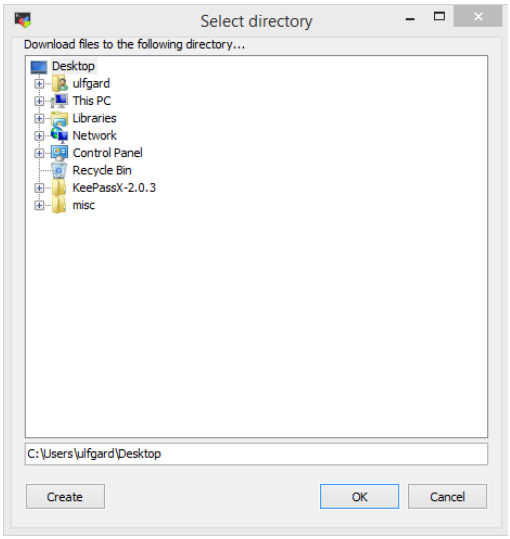

Figure 5 MobaXterm Upload Menu

5. To upload a file: Navigate to the desired destination in the SFTP tab. Click the green arrow “Upload to current folder” in the sidebar and select the local file(s) which you would like to upload.

6. It is also possible to drag and drop between the File Explorer window and the SFTP window. Note that the SFTP window can be expanded (Fig. 6).

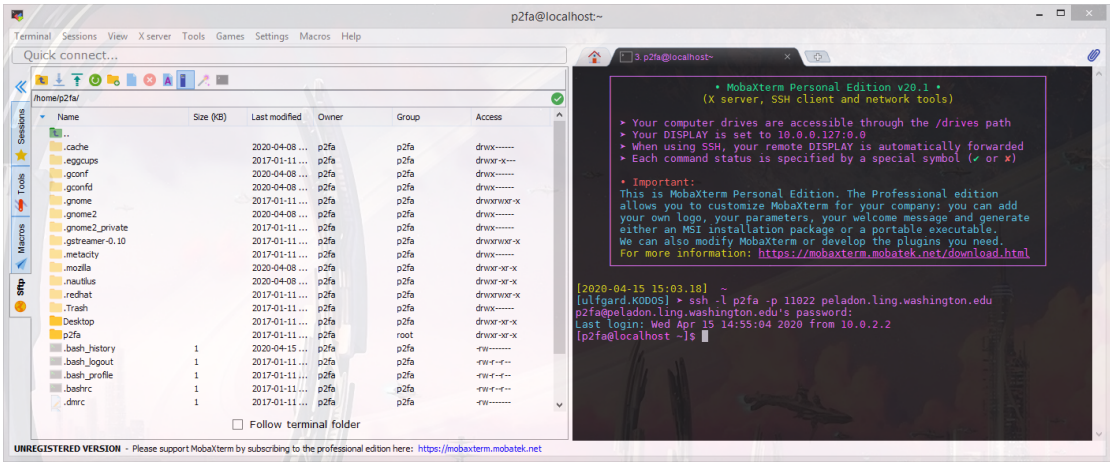

Figure 6: MobaXterm: Expanded SFTP Tab

Mac:

The recommended client for using SFTP on Mac is FileZilla. It serves as a graphical SFTP client.

1. Open FileZilla (1).

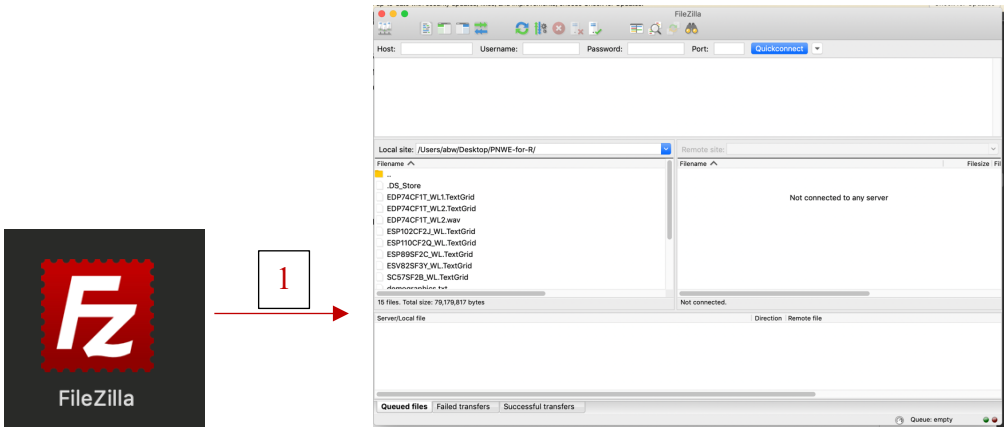

Figure 7. FileZilla logo and launch screen.

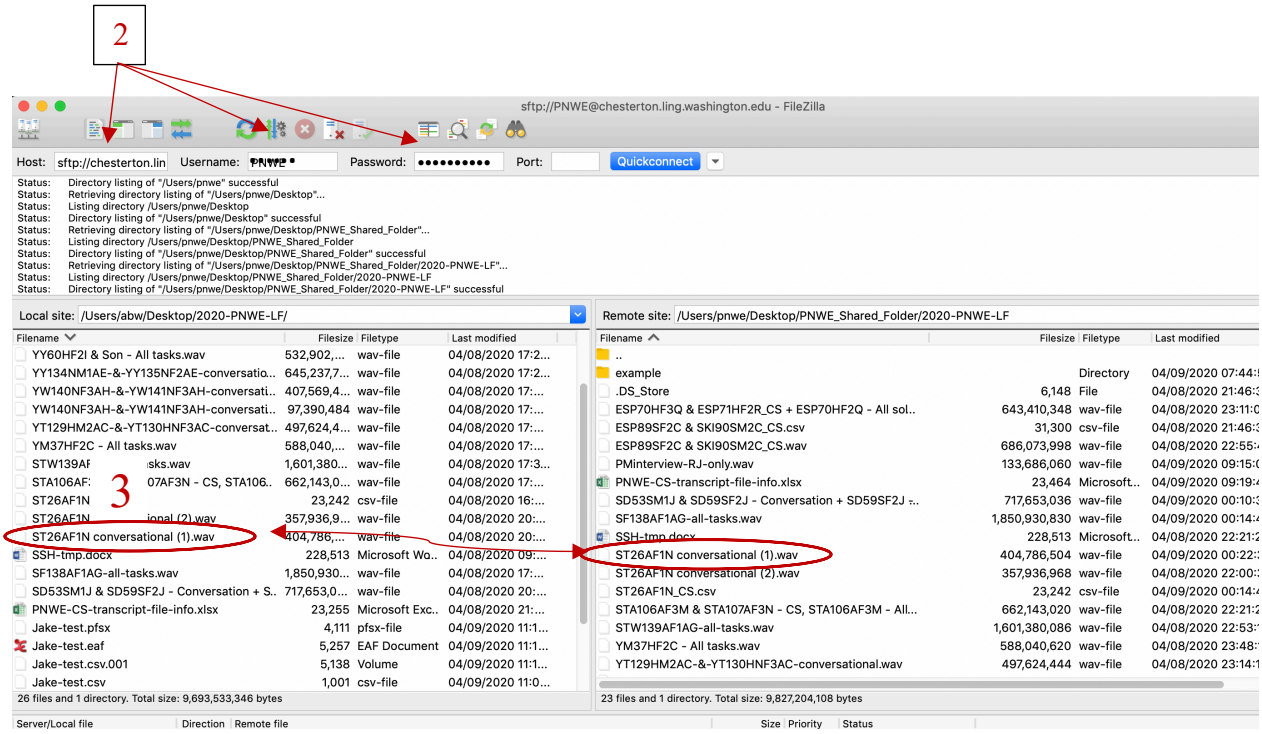

Figure 8: SFTP using drag and drop in FileZilla

2. Enter the hostname preceded by SFTP:// (sftp://<hostname>), as shown by the arrows labeled (2) in the screenshot.
    - Username: enter the username for the account.
    - Password: enter the password for the account.
    - Port: This is 22 by default, and generally does not need to be changed. If you experience difficulties connecting, try manually entering 22. In some cases, however, the remote SSH server is configured with a different port, which can be entered in this field.
    - This should provide the remote account's file structure, shown in the lower right panel.
    - The local file structure is in the lower left panel.

3. To upload/download a file: Select the file(s) which you would like to copy from the local structure or remote file structure. To transfer, simply drag and drop the file at its desired destination (3).

4. You can also perform a limited number of file functions inside the user interface.

- With a file selected in the remote server pane, CTRL + Click to display a dropdown menu. Select “rename” to rename, delete, or view/edit files.

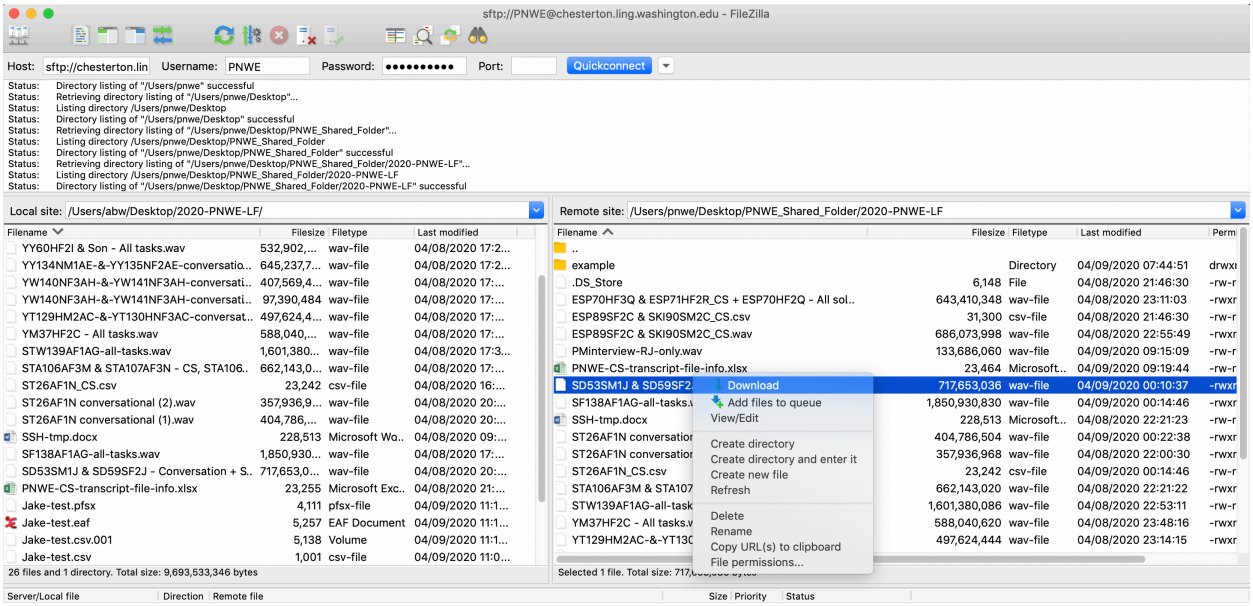

Figure 9: FileZilla: Download Files

Section III: Using SSH Clients and Connections

Shell accounts are useful as more than storage containers; they may also allow users to run programs on the remote system, such as P2FA, and have acted as development environments for decades. UW student webspace is accessed via SSH and SFTP, as are several systems in the Sociolinguistics Lab.

To use SSH to reach a remote server, you need a terminal emulator, which is the SSH client itself. On a Windows machine, MobaXterm may be used. The same connection which provides a graphical SFTP interface also opens a terminal to the remote host. On Macs, this is the Terminal application found in Utilities.

- Windows: MobaXterm
- Mac: go to Applications → Utilites → Terminal.

Use the terminal to connect to the remote host:

ssh <username>@<hostname>

Breaking down the command:

- The primary command is **ssh**, which opens the SSH connection to the remote host (<hostname>).
- The general syntax for SSH is: **ssh [flags] remotehost [flags]**

- In general, the order of the flags and the position (before or after <hostname>) does not matter.
- To find more information on the ssh command, use the man pages. Enter: man ssh at the command prompt.

Now what?

This depends on the purpose of the account. For many use cases in the lab, a server's remote function is purely SFTP, so SSH is not necessary. However, if there is work to be done, it will generally be in order to use a specific program. In that case, the best practice is to create a working directory for whatever project you are working on (unless one has been provided for you).

Create a directory for your working files. Use the **pwd** command to check your current directory. When you first log into an account, it is the "root" of the account's file structure (e.g., **/home/myaccount)**. You can get to the root directory from anywhere using the change directory command, **cd**, with no other arguments (i.e., without anything following it on the command line). To create a directory, use the **mkdir** command to create your directory: e.g.,

mkdir /home/myaccount/<directory name>

It is best to choose a name which is easily identifiable; if desired <directory name> is my_directory, the command would be:

mkdir /home/p2fa/my_directory

All of the above commands are further detailed in **Section III**.

- **pwd** to display the current working directory.
- **cd /directory/path** to change to /directory/path.
- **mkdir /home/<username>/<directory name>** to create the directory /home/<username>/<directory name>.

A brief word about file and directory names

Linux tracks whitespace, and interprets a break between words as the end of a particular command or flag. Thus file and directory names with blank spaces in them can create problems. It is usually best to use a dash or underscore to replace whitespace characters. E.g., cool filename.txt would be better as cool_filename.txt or cool-filename.txt. To get around this issue with existing files, enclose the filename in '', so that the system will interpret 'cool filename.txt' as intended.

Running Programs

There is no specific recipe for running a program or command in Linux. Each is context specific. For example, a command like **cd** is accessible system-wide, while a specific home-brew Python program may only work from a specific directory. Running the former command

just requires the command itself. Running the latter requires that you actually be in the directory where the command is, or that you use the full path to the command. For example, to run a Python program, the **python** command is invoked. In most systems with python installed, this is a system-wide command. Running a specific Python program, mypython.py, requires not only the python command, but also the argument /full/path/to/mypython.py, so that the command would be:

python /full/path/to/mypython.py

To run scripts which are not recognized by the OS system-wide, it is necessary to either include the full path or, if you are already in the directory of the script, to put ./ before the program name. E.g., for the program dothis.go, you must either be in its directory **/this/script/directory** and use **./dothis.go** or use **/this/script/directory/dothis.go**. (For a program to run, it must have the 'x' (execute) permission for the user, group, or everyone. For more on permissions, see the File Permissions section of the Appendix.)

SFTP

Another use of the command line is as an sftp client, using the sftp command. Generally, this is sftp <username>@<hostname>. While you remain in a shell, the sftp command opens a secure ftp session, allowing you to upload and download files without a gui. The two primary commands are **put** and **get**. **put** uploads files, while **get** downloads them. For example, **put star.wav** would upload the file star.wav, while **get light.wav** would download light.wav. In SFTP, the **ls**, **cd**, and **pwd** commands all work as they would in a shell account (see Appendix for more on these commands).

Generally, a graphical interface is easier, but there are times when this is not true: e.g., if there is a directory with numerous files of various file types, it is often easier to use the command line and the sftp **get** or **put** commands with the wildcard: *.<file extension> (e.g., **put *.wav** will upload all of the files with the extension .wav from the local system to the remote system).

Section V: Appendix

Linux commands are somewhat arcane. Even so, most are well-documented through **man** (manual) pages that the user can access via the command line (by typing **man <command_name>**), as well as online. For most commands, a quick Google search will no doubt provide hundreds of examples of its use.

General Linux commands

cd

cd [/full/path/to/directory]

Change directory. This must be followed by the directory path. It is an indispensable command. Some examples:

- Change directly to the temp directory in your home directory:
  - **cd /home/my_directory/temp**
- Change to one level up in the directory structure (i.e., from /home/**my_directory**/temp to /home/**my_directory**):
  - **cd ../**
- Change to the home directory from anywhere (i.e., from anywhere to /home/**my_directory**):
  - cd
- **lcd** (**l**ocal **cd)** is used with **sftp** to change directories on the local system, while **cd** changes directories on the remote system.

exit

Exit the shell. This ends the login session to login shells.

pwd

Print working directory, i.e., display the name of my current directory on the screen.

- Use the command: **pwd**
  - sample output: /home/**my_directory**/myproject
- **lpwd** (**l**ocal **pwd)** is used with **sftp** to check the local wording directory, while **pwd** checks the working directory on the remote system.

cp

cp file1 file2 or **cp file1 /full/path/to/directory/** (Note the trailing '/'.)
Copy a file of one file name to another file name, or copy one or more files to the same names under directory. If the destination is an existing file, the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten). If the directory does not exist, then there the command will fail (i.e., if /full/path/to/directory does not exist, file1 cannot be copied there – it will not create a new directory structure). Some examples:

- To make a copy of file1 and name it file2:
  - **cp file1 file2**
- To copy file1 to the temp directory:
  - **cp file1 temp/**
  - Note: the trailing '/' is necessary for the command

man

Display the contents of the system manual pages (help) on the topic. Try **man man** first. This will provide a viewer with general information about the manual. Press "**q**" to quit the viewer; use the terminal scroll bar to view previous screen output. The command **info <topic>** works similar and may contain more up-to-date information. Manual pages can be hard to read, but can be a valuable tool. Linux is often learned by reading the manual pages. Try **any_command --help** for short, easy to digest help on a command. Some examples:

- To find the manual page for ssh:
  - **man ssh**
- To get help on ssh:
  - **ssh --help**

apropos

apropos topic

Give me the list of the commands that have something to do with my topic. This list can be quite long.

- E.g., **apropos python** lists 40+ files relevant to the topic (search string) “python”.

ls

ls [flags] [directory]

List the content of the current directory. Optionally lists the contents of the specified directory (e.g., /some/directory). Additional usage(with various flags):

- List all content of the current directory including hidden content.
  - **ls -a**
- Long list – list all files and folders including permissions, ownership, size, etc.
  - **ls -l**
- Combines the -a and -l flags.
  - **ls -al**
- **lls** (local ls) is used with **sftp** to list the contents of the local system, **ls** works as usual on the remote system.

mv

mv source destination

Move or rename files. The same command is used for moving and renaming files and directories. As with **cp**, the destination directory must exist. Some examples:

- Rename a file; FileNameA → FileNameB:
  - **mv FileNameA FileNameB**

- Move a file; (for file FileName with the full path /this/directory/FileName to /other/location):
  - **mv FileName /other/location/FileName**

rm

rm files

Remove (delete) files. You must own the file in order to be able to remove it. Remember: there is **NO** undo in Linux – once a file is deleted, it is gone for good. Be careful using wildcards (such as *.wav) as this could lead to losing multiple files. Some examples:

- Remove File1 and File2:
  - **rm File1 File2**
- Remove files of type .txt:
  - **rm *.txt**

mkdir

mkdir directory

Make a new directory. If the full path is not specified, the new directory is made in the current working directory. E.g., creating a directory from /home/p2fa will create /home/p2fa/directory (**mkdir directory**). The full path of a new directory can be specified, but every directory in its tree must already exist. E.g., creating the directory /home/p2fa/fantastic/new/directory cannot happen unless /home/p2fa/fantastic/new already exists (**mkdir /home/p2fa/fantastic/new/directory**).

- It can be useful to create a temporary directory to store working files so that the /home/p2fa/p2fa directory does not become cluttered. It is also easier to delete working files if they are stored in another directory.
  - E.g., **mkdir /home/p2fa/myWorkingFiles.**

rmdir

rmdir directory

Remove an empty directory. To remove a non-empty directory, use:

- **rm -rf directory**
  - The **-r** flag is recursive, so that it will delete all files in a given tree.
  - The **-f** flag forces the **rm** command. This is necessary in order to remove directories with the **rm** command.

Again, there is **NO** undo in Linux – once a file or directory is deleted, it is gone for good.

chmod *(see File Permissions)*

chmod [options] mode files
Change the access mode (permissions) of one or more files. Important flags:

- Print help message and then exit.
  - **--help**
- Traverse subdirectories recursively, applying changes.
  - **-R**

File Permissions

What are file permissions?

Every file or folder in Linux has access permissions. Often, the reason a program fails to execute properly is because of incorrect file permissions. There are three types of permissions (what allowed to do with a file):

read access l write access l execute access

Permissions are defined for three types of users:

- the owner of the file
- the group that the owner belongs to
- other users

Thus, Linux file permissions are nine bits of information (3 types x 3 type of users), each of them may have just one of two values: allowed or denied. Simply put, for each file it can be specified who can read or write from/to the file. For programs or scripts it also can be set if they are allowed to be executed.

File permissions notation

Textual representation like "-rwxr--r--" is used in Linux long directory listings. It consists of 10 characters. The first character shows the file type. The next 9 characters are permissions, consisting of three groups: owner, group, others. Each group consists of three symbols: **rwx** (in this order), if some permission is denied, then a dash "-" is used instead. Example:

-rwxr--r--

- Symbol in the position 0 (the first "-") is the type of the file. It is either "d" if the item is a directory, or "l" if it is a link, or "-" if the item is a regular file.
- Symbols in positions 1 to 3 ("rwx") are permissions for the owner of the file.
- Symbols in positions 4 to 6 ("r--") are permissions for the group.
- Symbols in positions 7 to 9 ("r--") are permissions for others.

Changing File Permissions

The **chmod** command

We use the **chmod** command to **ch**ange the access **mode** of a file. This command comes in many flavors, but we'll be talking primarily about one of them here.

chmod who=permissionsfilename

specified permissions for a given filename.

Who:

- **Letter**
  - u g o a
- **Meaning**
  - **u**ser who owns the file (usually you)
  - **g**roup the file belongs to
  - **o**ther users
  - **a**ll of the above

Permissions:

- r **r**ead
- w **w**rite (or delete)
- x e**x**ecute (search in case of directory)

Operator Code(Opcode):

- removes permissions + adds permissions

= assigns permissions

(removing those not present)

Using the command:

Type:

chmod *{who}***[*]***{permissions}* *<filename>*

where:

{who} is one or more of: **u**, **g**, **o**, or **a**

[*] is the operator code (opcode); one of **-**, **+**, or **=**

{permissions} is one or more of **r**, **w**, or **x**

<filename> is the name of the file for which you wish to change permissions

The-R flag

Use this flag to change all of the permissions within a directory tree:

chmod -R *{who}***[*]***{permissions}* *<directoryname>*
This prevents having to change every file within a directory tree individually.

Examples:

- Remove execute permission for other users.
  - Before: -rwxr-xr-x tmp.txt
  - Command: **chmod o=r tmp.txt**
  - After: -rwxr-xr-- tmp.txt
- Take away all permissions for the group for temp.inf We do this by leaving the permissions part of the command empty.
  - Before: -rw-r----- temp.inf
  - Command: **chmod g= temp.inf**
  - After: -rw------- temp.inf
- Open up greetings.html for reading and writing by anyone.
  - Before: -rw-r--r-- greetings.html
  - Command: **chmod og=rw greetings.html**
  - After: -rw-rw-rw- greetings.html