ASP2018 LINUX TUTORIAL

Lecturer: Eli Kasai

Tutorial Outcomes

By the end of the tutorial, you will be able to navigate, access, and modify files and folders on your computer—all without a mouse!

Why learn the command line?

We use our mouse and fingers to click images of icons and access files, programs, and folders on our devices. However, this is just one way for us to communicate with computers. The command line is a quick, powerful, text-based interface developers use to more effectively and efficiently communicate with computers to accomplish a wider set of tasks. Learning how to use it will allow you to discover all that your computer is capable of!

NAVIGATION

Your First Command

The **command line** is a text interface for your computer. It's a program that takes in commands, which it passes on to the computer's operating system to run.

From the command line, you can navigate through files and folders on your computer, just as you would with Finder on Mac OS or Windows Explorer on Windows. The difference is that the command line is fully text-based.

The advantage of using the command line is its power. You can run programs, write scripts to automate common tasks, and combine simple

commands to handle difficult tasks – making it an important programming tool.

This tutorial is for unix-based systems such as Linux and Mac OS X.

To access the command line, we use a terminal emulator, often just called the *terminal*.

In the terminal, after the `$`, type:

```
ls
```

and press `enter`.

You should see three items print out below the command.

ls

```
$ ls 2014 2015 hardware.txt
```

1. In the terminal, first you see `$`. This is called a *shell prompt*. It appears when the terminal is ready to accept a command.
2. When you type `ls`, the command line looks at the folder you are in, and then "lists" the files and folders inside it. The directories **2014**, **2015**, and the file **hardware.txt** are the contents of the current directory.

`ls` is an example of a *command*, a directive to the computer to perform a specific task.

Filesystem

A filesystem organizes a computer's files and directories into a tree structure:

1. The first directory in the filesystem is the *root directory*. It is the parent of all other directories and files in the filesystem.
2. Each parent directory can contain more child directories and files. Here **blog/** is the parent of **2014/**, **2015/**, and **hardware.txt**.
3. Each directory can contain more files and child directories. The parent-child relationship continues as long as directories and files are nested.

Let's see how to navigate the filesystem from the command line. In the terminal, after the shell prompt, type:

```
pwd
```

and press enter

pwd

```
$ pwd /Users/Kunwiji/LinuxTutorial/workspace/blog
```

`pwd` stands for "print working directory". It outputs the name of the directory you are currently in, called the *working directory*.

Here the working directory is **blog/**. Together with `ls`, the `pwd` command is useful to show where you are in the filesystem.

1.

Let's continue with more commands. In the terminal, print the working directory.

2.

List all files and directories in the working directory.

3.

Then type:

```
cd 2015
```

Again, print the new current working directory.

List all files and directories in the working directory.

cd I

```
$ cd 2015
```

1. `cd` stands for "change directory". Just as you would click on a folder in Windows Explorer or Finder, `cd` switches you into the directory you specify. In other words, `cd` changes the working directory.
2. The directory we change into is `2015`. When a file, directory or program is passed into a command, it is called an *argument*. Here the 2015 directory is an argument for the `cd` command.

The `cd` command takes a directory name as an argument, and switches into that directory.

Make sure you are in the directory **/Users/Kunwiji/LinuxTutorial/workspace/blog/2015** by using the command `pwd`.

1.

Once you are in the directory **/ Users/Kunwiji/LinuxTutorial /workspace/blog/2015**

Then type:

```
cd jan/memory/
```

Print the working directory to see the new location.

2.

Then type:

```
cd ..
```

Print the working directory again to see the new location.

cd II

```
$ cd jan/memory
```

To navigate directly to a directory, use `cd`with the directory's path as an argument. Here, `cd jan/memory/` command navigates directly to the **jan/memory** directory.

```
$ cd ..
```

To move up one directory, use `cd ..`. Here, `cd ..` navigates up from **jan/memory/** to **jan/**.

Make sure you are in the directory **/Users/Kunwiji/LinuxTutorial /workspace/blog/2015/jan** by using the command `pwd`.

1.

Once you are in **/Users/Kunwiji/LinuxTutorial /workspace/blog/2015/jan**, change the directory to the **2015/feb/** directory using:

```
cd ../feb
```

List all files and directories in the working directory.

2.

Type:

```
mkdir media
```

Again, list all files and directories in the working directory. You'll see that there is now a new directory named **media/**.

mkdir

```
$ mkdir media
```

The `mkdir` command stands for "make directory". It takes in a directory name as an argument, and then creates a new directory in the current working directory.

Here we used `mkdir` to create a new directory named **media/** inside the **feb/**directory.

Use `pwd` to make sure you are in the directory **/Users/Kunwiji/LinuxTutorial /workspace/blog/2015/feb** before moving to the checkpoints.

1.

Navigate to the **2014/dec/** directory.

```
cd ../../2014/dec
```

List all files and directories in the working directory.

2.

Then type:

```
touch keyboard.txt
```

Again, list all files and directories in the working directory. You'll see that there is now a new file named **keyboard.txt**.

touch

```
touch keyboard.txt
```

The `touch` command creates a new file inside the working directory. It takes in a filename as an argument, and then creates an empty file in the current working directory.

Here we used `touch` to create a new file named **keyboard.txt** inside the **2014/dec/**directory.

Instructions

The commands we've covered so far are commonly used to navigate the filesystem. There are more commands you can use to master the command line, and we'll cover them in the next session.

Let's summarize what we've done so far.

Generalizations

You've learned five commands commonly used to navigate the filesystem from the command line. What can we generalize so far?

- The *command line* is a text interface for the computer's operating system. To access the command line, we use the terminal.
- A *filesystem* organizes a computer's files and directories into a tree structure. It starts with the *root directory*. Each parent directory can contain more child directories and files.
- From the command line, you can navigate through files and folders on your computer:
    - `pwd` outputs the name of the current working directory.
    - `ls` lists all files and directories in the working directory.
    - `cd` switches you into the directory you specify.
    - `mkdir` creates a new directory in the working directory.
    - `touch` creates a new file inside the working directory.

MANIPULATION

ls, revisited

So far we've used the command line to navigate the filesystem.

We can do more with the command line to view directories and files. We can also use the command line to copy, move, and remove files and directories. Let's see how this is accomplished.

1.

In the terminal after the shell prompt, type

```
ls
```

2.

Then type

```
ls -a
```

Do you see the differences between the outputs of both commands?

ls -a

```
$ ls -a . .. .preferences action drama comedy genres.txt
```

1. The `ls` command lists all files and directories in the working directory.
2. The `-a` modifies the behavior of the `ls`command to also list the files and directories starting with a dot (`.`). Files started with a dot are hidden, and don't appear when using `ls` alone.

The `-a` is called an *option*. Options modify the behavior of commands. Here we used `ls -a`to display the contents of the working directory in more detail.

In addition to `-a`, the `ls` command has several more options. Here are three common options:

- `-a` - lists all contents, including hidden files and directories
- `-l` - lists all contents of a directory in long format
- `-t` - order files and directories by the time they were last modified.

Let's practice using these options below.

Instructions

1.

In the terminal, type

```
ls -l
```

ls -l

```
$ ls -l
drwxr-xr-x 5 Kunwiji staff 4096 Jun 24 16:51 action
drwxr-xr-x 4 Kunwiji staff 4096 Jun 24 16:51 comedy
drwxr-xr-x 6 Kunwiji staff 4096 Jun 24 16:51 drama
-rw-r--r-- 1 Kunwiji staff 0 Jun 24 16:51 genres.txt
```

The `-l` option lists files and directories as a table. Here there are four rows, with seven columns separated by spaces. Here's what each column means:

1. Access rights. These are actions that are permitted on a file or directory.
2. Number of hard links. This number counts the number of child directories and files. This number includes the parent directory link (`..`) and current directory link (`.`).
3. The username of the file's owner. Here the username is `Kunwiji`.
4. The name of the group that owns the file. Here the group name is `staff`.
5. The size of the file in bytes.
6. The date & time that the file was last modified.
7. The name of the file or directory.

1.

Let's try out another option for the `ls`command.

Navigate to the **comedy/** directory.

2.

Then type

```
ls -alt
```

ls -alt

```
$ ls -alt
drwxr-xr-x 4 cc eng 4096 Jun 29 12:22 .
-rw-r--r-- 1 cc eng 0 Jun 29 12:22 .gitignore
```

```
drwxr-xr-x 5 cc eng 4096 Jun 30 14:20 ..
drwxr-xr-x 2 cc eng 4096 Jun 29 12:22 satire
drwxr-xr-x 2 cc eng 4096 Jun 29 12:22 slapstick
-rw-r--r-- 1 cc eng 14 Jun 29 12:22 the-office.txt
```

The `-t` option orders files and directories by the time they were last modified.

In addition to using each option separately, like `ls -a` or `ls -l`, multiple options can be used together, like `ls -alt`.

Here, `ls -alt` lists all contents, including hidden files and directories, in long format, ordered by the date and time they were last modified.

1.

Let's move on to copying, moving, and removing files and directories from the command line.

Navigate to the **drama/biopic/**directory.

```
cd ../drama/biopic/
```

List all files and directories in the working directory.

2.

Then type

```
cp frida.txt lincoln.txt
```

Let's learn about this command.

cp l

```
cp frida.txt lincoln.txt
```

The `cp` command copies files or directories. Here, we copy the contents of **frida.txt** into **lincoln.txt**.

1.

Navigate to the **drama/** directory.

```
cd ..
```

List all files and directories in the working directory.

2.

Type

```
cp biopic/cleopatra.txt historical/
```

3.

Navigate to the **historical/** directory.

List all files and directories in the working directory. You should see a new copy of **cleopatra.txt** in this directory.

4.

Here's one more way to use `cp`.

Navigate up one directory from **drama/historical/** to **drama/**.

5.

Then type

```
cp biopic/ray.txt biopic/notorious.txt historical/
```

6.

Change directories into **historical/**.

List all files and directories in the working directory. You should see a new copy of **ray.txt** and **notorious.txt**in this directory.

cp II

```
cp biopic/cleopatra.txt historical/
```

To copy a file into a directory, use `cp` with the source file as the first argument and the destination directory as the second argument. Here, we copy the file **biopic/cleopatra.txt** and place it in the **historical/** directory.

```
cp biopic/ray.txt biopic/notorious.txt historical/
```

To copy multiple files into a directory, use `cp` with a list of source files as the first arguments, and the destination directory as the last argument. Here, we copy the files **biopic/ray.txt** and **biopic/notorious.txt** into the **historical/** directory.

1.

Let's look at two more ways to use `cp`.

Navigate to the **comedy/** directory.

```
cd ../../comedy
```

2.

In this directory, create a new file named **shrek.txt**.

3.

Then type

```
cp * satire/
```

4.

Navigate to the **satire/** directory.

List all files and directories in the working directory.

You should see a copy of the files **the-office.txt** and **shrek.txt** in this directory. I'll explain how this works in the next exercise.

5.

Here's another way to use `cp`.

Navigate to the **action/** directory. Type

```
cd ../../action/
```

Here we navigate up two directories, and then into the **action/** directory.

6.

Type

```
cp m*.txt scifi/
```

7.

Change directories into **scifi/**.

List all files and directories in the working directory.

You should see a copy of all text files starting with "m": **matrix.txt**, **matrix-reloaded.txt**, and **matrix-revolutions.txt**.

Let's learn how this works.

Wildcards

```
cp * satire/
```

In addition to using filenames as arguments, we can use special characters like `*` to select groups of files. These special characters are called *wildcards*. The `*` selects all files in the working directory, so here we use `cp` to copy all files into the **satire/** directory.

```
cp m*.txt scifi/
```

Here, `m*.txt` selects all files in the working directory starting with "m" and ending with ".txt", and copies them to ***scifi/***.

1.

In addition to copying files, we can move files from the command line.

Change directories into the **action/**directory.

```
cd ../
```

2.

Type

```
mv superman.txt superhero/
```

3.

Navigate to the **superhero/** directory.

List all files and directories in the working directory. You should see **superman.txt** in it.

4.

Here's another way to use `mv`.

Navigate up one directory from **action/superhero/** to **action/**.

5.

Then type

```
mv wonderwoman.txt batman.txt superhero/
```

6.

Navigate to **superhero/** again.

List all files and directories in the working directory. You should see **wonderwoman.txt** and **batman.txt** in it.

7.

Here's one more way to use `mv`.

Type

```
mv batman.txt spiderman.txt
```

8.

List all files and directories in the working directory. You should see the file **batman.txt** has been renamed as **spiderman.txt**.

Let’s learn how these commands work.

mv

The `mv` command moves files. It's similar to `cp` in its usage.

```
mv superman.txt superhero/
```

To move a file into a directory, use `mv` with the source file as the first argument and the destination directory as the second argument. Here we move **superman.txt** into **superhero/**.

```
mv wonderwoman.txt batman.txt superhero/
```

To move multiple files into a directory, use `mv`with a list of source files as the first arguments, and the destination directory as the last argument. Here, we move **wonderwoman.txt** and **batman.txt** into **superhero/**.

```
mv batman.txt spiderman.txt
```

To rename a file, use `mv` with the old file as the first argument and the new file as the second argument. By
moving **batman.txt** into **spiderman.txt**, we rename the file
as **spiderman.txt**.

1.

Change directory to **comedy/slapstick**.

List all files and directories in the working directory.

```
cd ../../comedy/slapstick/
```

2.

Type

```
rm waterboy.txt
```

3.

List all files and directories in the working directory. You should see that **waterboy.txt** has been removed.

4.

Navigate up one directory from **comedy/slapstick/** to **comedy/**.

5.

Type

```
rm -r slapstick
```

6.

List all files and directories in the working directory. You should see that the **slapstick/** directory has been removed.

Let's learn how the `rm`command works.

rm

```
rm waterboy.txt
```

The `rm` command deletes files and directories. Here we remove the file **waterboy.txt** from the filesystem.

```
rm -r comedy
```

The `-r` is an option that modifies the behavior of the `rm` command. The `-r` stands for "recursive," and it's used to delete a directory and all of its child directories.

Be careful when you use `rm`! It deletes files and directories permanently. There isn't an undelete command, so once you delete a file or directory with `rm`, it's gone.

The commands we've covered so far are commonly used to view and change the filesystem.

Let's summarize what we've done so far.

Generalizations

You learned how to use the command line to view and manipulate the filesystem. What can we generalize so far?

- Options modify the behavior of commands:
    - `ls -a` lists all contents of a directory, including hidden files and directories
    - `ls -l` lists all contents in long format

    - `ls -t` orders files and directories by the time they were last modified
    - Multiple options can be used together, like `ls -alt`
- From the command line, you can also copy, move, and remove files and directories:
    - `cp` copies files
    - `mv` moves and renames files
    - `rm` removes files
    - `rm -r` removes directories
- Wildcards are useful for selecting groups of files and directories

APPENDIX

BACKGROUND

The command line is a text interface for your computer. It's a program that takes in commands, which it passes on to the computer's operating system to run.

From the command line, you can navigate through files and folders on your computer, just as you would with Windows Explorer on Windows or Finder on Mac OS. The difference is that the command line is fully text-based.

Here's an appendix of commonly used commands.

COMMANDS

>

```
$ cat oceans.txt > continents.txt
```

`>` takes the standard output of the command on the left, and redirects it to the file on the right.

>>

```
$ cat glaciers.txt >> rivers.txt
```

`>>` takes the standard output of the command on the left and *appends*(adds) it to the file on the right.

<

```
$ cat < lakes.txt
```

`<` takes the standard input from the file on the right and inputs it into the program on the left.

|

```
$ cat volcanoes.txt | wc
```

`|` is a "pipe". The `|` takes the standard output of the command on the left, and *pipes* it as standard input to the command on the right. You can think of this as "command to command" redirection.

~/.BASH_PROFILE

```
$ nano ~/.bash_profile
```

~/.bash_profile is the name of file used to store environment settings. It is commonly called the "bash profile". When a session starts, it will load the contents of the bash profile before executing commands.

ALIAS

```
alias pd="pwd"
```

The `alias` command allows you to create keyboard shortcuts, or aliases, for commonly used commands.

CD

```
cd Desktop/
```

`cd` takes a directory name as an argument, and switches into that directory.

```
$ cd jan/memory
```

To navigate directly to a directory, use `cd` with the directory's path as an argument. Here, `cd jan/memory/` command navigates directly to the **jan/memory** directory.

CD ..

```
$ cd ..
```

To move up one directory, use `cd ...`. Here, `cd ..` navigates up from **jan/memory/** to **jan/**.

CP

```
$ cp ada_lovelace.txt historical/
```

`cp` copies files or directories. Here, we copy the file **ada_lovelace.txt** and place it in the **historical/** directory

WILDCARDS (*)

```
$ cp * satire/
```

The wildcard `*` selects all of the files in the current directory. The above example will copy all of the files in the current directory to the directory called **satire**. There are other types of wildcards, too, which are beyond the scope of this glossary.

```
$ cp m*.txt scifi/
```

Here, m*.txt selects all files in the working directory starting with "m" and ending with ".txt", and copies them to scifi/.

ENV

```
env
```

The `env` command stands for "environment", and returns a list of the environment variables for the current user.

ENV | GREP VARIABLE

```
env | grep PATH
```

`env | grep PATH` is a command that displays the value of a single environment variable.

EXPORT

```
export USER="Jane Doe"
```

`export` makes the variable to be available to all child sessions initiated from the session you are in. This is a way to make the variable persist across programs.

GREP

```
$ grep "Mount" mountains.txt
```

`grep` stands for "global regular expression print". It searches files for lines that match a pattern and returns the results. It is case sensitive.

GREP -I

```
$ grep -i "Mount" mountains.txt
```

`grep -i` enables the command to be case insensitive.

GREP -R

```
$ grep -R Arctic /home/ccuser/workspace/geography
```

`grep -R` searches all files in a directory and outputs filenames and lines containing matched results. `-R` stands for "recursive".

GREP -RL

```
$ grep -Rl Arctic /home/ccuser/workspace/geography
```

`grep -Rl` searches all files in a directory and outputs only filenames with matched results. `-R` stands for "recursive" and `l` stands for "files with matches".

HOME

```
$ echo $HOME
```

The `HOME` variable is an environment variable that displays the path of the home directory.

LS

```
$ ls
2014  2015  hardware.txt
```

`ls` lists all files and directories in the working directory

ls -a

```
ls -a
.  ..  .preferences  action  drama comedy  genres.xt
```

`ls -a` lists all contents in the working directory, including hidden files and directories

ls -l

```
ls -l
drwxr-xr-x 5  cc  eng  4096 Jun 24 16:51  action
drwxr-xr-x 4  cc  eng  4096 Jun 24 16:51  comedy
drwxr-xr-x 6  cc  eng  4096 Jun 24 16:51  drama
-rw-r--r-- 1  cc  eng     0 Jun 24 16:51  genres.txt
```

`ls -l` lists all contents of a directory in long format. Here's what each column means.

ls -t

`ls -t` orders files and directories by the time they were last modified.

MKDIR

```
$ mkdir media
```

`mkdir` takes in a directory name as an argument, and then creates a new directory in the current working directory. Here we used mkdir to create a new directory named **media/**.

MV

```
$ mv superman.txt superhero/
```

To move a file into a directory, use mv with the source file as the first argument and the destination directory as the second argument. Here we move superman.txt into superhero/.

NANO

```
$ nano hello.txt
```

nano is a command line text editor. It works just like a desktop text editor like TextEdit or Notepad, except that it is accessible from the the command line and only accepts keyboard input.

PATH

```
$ echo $PATH

/home/ccuser/.gem/ruby/2.0.0/bin:/usr/local/sbin:/usr/local/bin:/usr
/bin:/usr/sbin:/sbin:/bin
```

`PATH` is an environment variable that stores a list of directories separated by a colon. Each directory contains scripts for the command line to execute. PATH lists which directories contain scripts.

PWD

```
$ pwd
/home/ccuser/workspace/blog
```

`pwd` prints the name of the working directory

RM

```
$ rm waterboy.txt
```

`rm` deletes files. Here we remove the file waterboy.txt from the file system.

RM -R

```
$ rm -r comedy
```

`rm -r` deletes a directory and all of its child directories.

SED

```
$ sed 's/snow/rain/' forests.txt
```

`sed` stands for "stream editor". It accepts standard input and modifies it based on an *expression*, before displaying it as output data.
In the expression `'s/snow/rain/'`:

- `s`: stands for "substitution".
- `snow`: the search string, the text to find.
- `rain`: the replacement string, the text to add in place.

SORT

```
$ sort lakes.txt
```

`sort` takes a filename or standard input and orders each line alphabetically, printing it to standard output.

STANDARD ERROR

standard error, abbreviated as `stderr`, is an error message outputted by a failed process.

SOURCE

```
source ~/.bash profile
```

`source` activates the changes in **~/.bash_profile** for the current session. Instead of closing the terminal and needing to start a new session, `source`makes the changes available right away in the session we are in.

STANDARD INPUT

standard input, abbreviated as `stdin`, is information inputted into the terminal through the keyboard or input device.

STANDARD OUTPUT

standard output, abbreviated as `stdout`, is the information outputted after a process is run.

TOUCH

```
$ touch data.txt
```

`touch` creates a new file inside the working directory. It takes in a file name as an argument, and then creates a new empty file in the current working directory. Here we used touch to create a new file named keyboard.txt inside the 2014/dec/ directory.

If the file exists, touch is used to update the modification time of the file

UNIQ

```
$ uniq lakes.txt
```

`uniq`, short for "unique", takes a filename or standard input and prints out every line, removing any exact duplicates.