codecademy

Navigating the File System

Print Working Directory `pwd`

The shell command `pwd` displays the file path from the root directory to the current working directory.

```
$ pwd
/Users/sonny/Downloads
```

Make Directory `mkdir`

The shell command `mkdir` can be used to make a new directory in the filesystem according to its argument. If a file path is given, the new directory will be placed at the end. Otherwise, it will create a new directory in the current working directory with the name given.

```
$ mkdir new-directory
$ ls
old-directory    new-directory
```

List `ls`

The shell command `ls` is used to list the contents of directories. If no arguments are given, it will list the contents of the current working directory.

```
$ ls Desktop
resume.pdf
photo.png
```

`cd` Change Directory

The shell command `cd` can be used to move throughout the filesystem of a computer. It accepts a variety of arguments:

- Full file paths.
- Names of children of the current directory.
- `..` the parent of the current directory.

```
$ cd some-directory
$ cd ..
```

Filesystem Structure

A computer's filesystem organizes the data stored by a computer, so that it can be easily retrieved by the user. Files are typically represented by a tree-like structure, in which any parent directory can have any number of children.
The *root directory* is then found at the base of the tree.

`touch` Create New File

The shell command `touch` creates a new file in the current working directory with the name provided.

```
$ touch secret-file.txt
```

The Command Line

The *command line* allows a user to navigate the filesystem and run built-in programs or custom scripts. In Unix, the command line interface is called Bash, and the shell prompt is the `$`.

```
$
```

Helper Commands

Helper commands for the command line include:

- `clear` to clear the terminal
- tab to autocomplete the line
- ↑ and ↓ to cycle through your previous commands

codecademy

Viewing and Changing the File System

`cp` Copy

The shell command `cp` is used to copy files or directories.

The basic argument structure is `cp source destination`, where the `source` is the file/directory to copy to the `destination` file/directory.

```
$ cp file1 file1_copy
$ cp file1 file2 destination_folder
```

Command Options

Options can be used to modify the behavior of shell commands. Shell command *options* are commonly represented by a single letter preceded by a `-`. For example, `-l`, `-a`, and `-d` could all be options that follow a shell command.

`mv` Move

The shell command `mv` is used 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.

```
$ mv index.html website/
```

`rm` Remove

The shell command `rm` is used to delete files and directories. The `-r` flag deletes a directory and all of its files and directories (`rm -r`).

```
$ rm -r bad_selfies
```

`ls` List Command Options

The shell command `ls` is used to list the contents in a directory. It can be combined with the following command options:

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

```
$ ls -a
$ ls -l
$ ls -t
```

Redirecting Input and Output

Append Redirect shell command

The `>>` shell command is used to redirect the standard output of the command on the left and append (add) it to the end of the file on the right.

```
# This command will append "Hello World!" to greetings.txt
echo "Hello World!" >> greetings.txt
```

Pipe shell command

The `|` command is called a *pipe*. It is used to *pipe*, or transfer, the standard output from the command on its left into the standard input of the command on its right.

```
# First, echo "Hello World" will send Hello World to the
standard output.
# Next, pipe | will transfer the standard output to the next
command's standard input.
# Finally, wc -w will count the number of words from its
standard input, which is 2.
echo "Hello World" | wc -w
```

Redirecting Output

The `>` symbol is used to redirect output by taking the output from the command on the left and passing as input to the file on the right.

```
echo "Hello" > hello.txt
```

`cat` Display

The shell command `cat` displays the contents of one or more files to the terminal.

```
$ cat poem.txt
$ cat poem.txt kitties.txt
```

`grep` Search

The shell command `grep` is used to search files for lines that match a pattern and returns the results. Various options can be specified along with the `grep` command to specify the search.
In the provided example, the lines in the file **names.txt** which contain “sonny” will be returned.

```
grep 'sonny' names.txt
```

Case insensitive search

The shell `grep` command searches files for a particular pattern. The `grep` command with the `-i` option can be used to search files for lines that match a pattern, case insensitive, and returns the results.

grep -R shell command

The shell command `grep` has a `-R` option (`grep -R`) that searches all files in a directory, including its subdirectories, and outputs filenames and lines containing matched results.

Command Line Redirection

On a command line, `redirection` is the process of using the input/output of a file or command to use it as an input for another file. It is similar but different from pipes, as it allows reading/writing from files instead of only commands.

Redirection can be done by using the operators `>` and `>>` .

```
ls > directories_list.txt
ls >> directories_list.txt
```

Configuring the Environment

Command line environment

The *environment* of the command line refers to the settings and preferences of the current user. It enables users to set greetings, `alias` commands, variables, and much more.

Shell Command env

For Unix-based systems like Mac OS and Linux (not Windows), the shell command `env` returns a list of environment variables for the current user.

Alias

The shell command `alias` is used to assign commonly used commands to shortcuts (or aliases). The assigned commonly used command should be wrapped in double quotes.

```
# The following command creates an alias `pd` for the
command `pwd`

alias pd="pwd"
```

Environment Variables

Variables that can be used across terminal commands are called *environment variables*. They also hold information about the shell's environment.

Source Bash Profile

All the commands in **~/.bash_profile** are executed with the shell command `source ~/.bash_profile`. So when changes are made to **~/.bash_profile**, run this command to activate the changes in the current session.

`history` command in Unix Systems

The `history` shell command is used to get a history of commands (also known as "events") that were executed in the current session. The command also allows us to perform operations on this list of commands that have been executed, such as selecting or manipulating a command in the history.

Export command

The `export` command makes a given variable available to all child sessions initiated from the current session.

```
# This command will make the environment variable USER
available
# to all child sessions with the value "Jane Doe".
export USER="Jane Doe"
```

`HOME` Environment Variable in Unix Systems

`HOME` is an environment variable present in command line environments. It is used to get the path to the current user's home directory. This makes it easy for programs to access the home directory when needed.

```
# To show the path of the home directory use the following
command:

echo $HOME
```

Bash Scripting

Bash Script Arguments

Arguments can be added to a bash script after the script's name. Once provided they can be accessed by using `$(position in the argument list)`. For example, the first argument can be accessed with `$1`, the second with `$2`, the third with `$3`, etc.

```
#!/bin/bash
# For a script invoked by saycolors red green blue
# echoes red
echo $1

# echoes green
echo $2

# echoes blue
echo $3
```

Bash script variables

Variables in a bash script can be set using the `=` sign, and accessed using `$`.

```
greeting="Hello"

echo $greeting
```

bash script `read` keyword

The `read` command can be used to prompt the user for input. It will continue to read user input until the Enter key is pressed.

Some prompt text can also be specified using `-p` with the `read` command.

```
#!bin/bash
echo "Press Enter to continue"
read
read -p "Enter your name: " name
```

Bash Shebang

Bash script files start with `#!/bin/bash`. This special line tells the computer to use *bash* as the intepreter.

Bash Aliases

Aliases can be created using the keyword `alias`. They are used to create shorter commands for calling bash scripts. They can also be used to call bash scripts with certain arguments.

```
# For example, to create an alias that invokes the saycolor
# script with the argument "green", the following syntax is
used:
alias saygreen='./saycolors.sh "green"'
```

Bash Scripts

Reusuable sets of *bash* terminal commands can be created using *bash scripts*. *Bash scripts* can run any command that can be run in a terminal.

Bash script comparison operators

In bash scripting, strings are compared using the `==` (Equal) and `!=` (Not equal) operators.

```
#!bin/bash
word1="Hello"
word2="Hello"
word3="hello"

if [ $word1 == $word2 ]
then
  echo "Strings are equal"
fi

if [ $word1 != $word3 ]
then
  echo "Strings are not equal"
fi
```