The command line

* 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.
* You can use helper commands to make navigation easier:
  + clear clears the terminal
  + tab autocompletes the name of a file or directory
  + ↑ and ↓ allow you to cycle through previous commands

What can we generalize so far?

* Options modify the behaviour of commands:
  + ls -a lists all contents of a directory, including hidden files also known as dot(.) files and directories
  + ls -l lists all contents in long format, as well as the file permissions
  + ls -t orders files and directories by the time they were last modified
  + ls -alt lists all contents, including hidden files and directories, in long format, ordered by the date and time they were last modified.

A screenshot of a computer

Description automatically generated with medium confidence

Access rights, number of hard links, username of the file’s owner, name of the group that owns the file, size of the file in bytes, date & time that the file was last modified, name of the file or directory.

* From the command line, you can also copy, move, and remove files and directories:
  + cat command outputs the contents of a specified file:
    - * 1. cat genres.txt, cat action/batman.txt
  + cp copies files contents of a source file into a destination file:
    - * 1. cp source.txt destination.txt
        2. cp terminator.txt terminator.bak
        3. cp source.txt directory - copies a file to a destination directory
        4. cp the-office.txt slapstick/the-office-us.txt - rename a file inside a directory
        5. cp file1.txt file2.txt my\_directory - copy multiple files into a directory
  + mv moves and renames files:
    - * 1. mv source.txt my\_directory moves a source file from the current directory into a destination directory
        2. mv my\_file\_1.txt my\_file\_2.txt my\_directory moves multiple files from the current directory into a destination directory
        3. mv file\_original.txt file\_renamed.txt renames a file from the current directory
  + rm removes files:
    - * 1. rm unwanted\_file.txt removes a file from the current directory
        2. rm -r removes directories
* Wildcards are useful for selecting groups of files and directories
  + cp \* my\_directory to copy all files in the current working directory into another directory
  + cp \*.txt my\_directory to copy just the ‘.txt’ files into another directory
  + cp w\*.txt my\_directory to copy all files in the working directory starting with “w” (prefix) and ending with “.txt” (suffix) into another directory

Standard input, standard output, and standard error:

1. standard input, abbreviated as stdin, is information inputted into the terminal through the keyboard or input device.
2. standard output, abbreviated as stdout, is the information outputted after a process is run.
3. standard error, abbreviated as stderr, is an error message outputted by a failed process.

The > command redirects the standard output to a file, e.g. *echo "Hello" > hello.txt*

*cat deserts.txt > forests.txt* where > command takes the standard output of the command on the left, and redirects it to the file on the right.

*cat deserts.txt >> forests.txt* where >> command takes the standard output of the command on the left and appends (adds) it to the file on the right.

*cat* < *deserts.txt* < takes the standard input from the file on the right and inputs it into the command on the left.  It does the same thing as *cat deserts.txt*.

The | takes the standard output of the command on the left, and pipes it as standard input to the command on the right, e.g. *cat volcanoes.txt | wc* counts the words in volcanoes.txt using the word count command.

In *cat volcanoes.txt | wc | cat > count.txt* the standard output of *cat volcanoes.txt* is “piped” to the *wc* command. The standard output of *wc* is then “piped” to *cat*. Finally, the standard output of *cat* is redirected to *count.txt*.

In *sort continents.txt* the *sort* commandstakes the standard input and orders it alphabetically for the standard output (it doesn’t change the file itself).

In *cat glaciers.txt | sort > sorted-glaciers.txt* the *sort* command takes the standard output from *cat glaciers.txt* and “pipes” it to *sort*. The standard output of *sort* is redirected to a new file named *sorted-glaciers.txt*.

In *uniq deserts.txt* filters out adjacent duplicates. In *sort deserts.txt | uniq* the *sort* command alphabetized a file, and then “pipe” the standard output to *uniq*.

In *sort deserts.txt | uniq > uniq-deserts.txt* the *sort* command alphabetized a file, and then “pipe” the standard output to uniq. Finally, the standard output of *uniq* is redirected to *uniq-deserts.txt*.

In *grep America continents.txt*, *grep* stands for “global regular expression print.” It searches files for lines that match a pattern and then returns the results. Here *grep* searched for anything that matched “America” in *continents.txt*.

In *grep -i America continents.txt*, *grep -i* enables the command to be case insensitive. Here, *grep* searched for capital or lowercase strings that match “America” in *continents.txt*.

In *grep -R Arctic /home/ccuser/workspace/geography*, *grep -R* searches all files in a directory and outputs *filenames and lines* containing matched results and *-R* stands for “recursive”.

In *grep -Rl Arctic /home/ccuser/workspace/geography*, *grep -Rl* searches all files in a directory and outputs *only filenames* with matched results (so no lines).

In *sed 's/snow/rain/' forests.txt*, the *sed* command accepts standard input and modifies it based on an expression, and displays output data. In the expression *'s/snow/rain/'*:

* *s*: stands for “substitution.” It is always used when using sed for substitution.
* *snow*: the search string, or the text to find.
* *rain*: the replacement string, or the text to add in place.

In *sed 's/snow/rain/' forests.txt*, *sed* searches *forests.txt* for the word *“snow*” and replaces it with *“rain”*. Importantly, the command will only replace the *first instance* of *“snow”* on a line.

In *sed 's/snow/rain/g'* *forests.txt*, the *sed* command uses the *g* expression, meaning “global.” Here *sed* searches *forests.txt* for the word *“snow”* and replaces *all instances* of *“snow”* with *“rain”*.

*sed* as used above will only rewrite the command line output and the actual file won’t be changed.

In *sed -i 's/snow/rain/g' forests.txt*, use *-i* at the beginning of the command in order to rewrite the actual file.

The above command will rewrite *forests.txt* and replace all instances of *“snow”* with *“rain”*.

Using the command line to redirect standard input and standard output

* Redirection reroutes standard input, standard output, and standard error.
* The common redirection commands are:
  + > redirects standard output of a command to a file, overwriting previous content.
  + >> redirects standard output of a command to a file, appending new content to old content.
  + < redirects standard input to a command.
  + | redirects standard output of a command to another command.
* A number of other commands are powerful when combined with redirection commands:
  + sort: sorts lines of text alphabetically.
  + uniq: filters duplicate, adjacent lines of text.
  + grep: searches for a text pattern and outputs it.
  + sed : searches for a text pattern, modifies it, and outputs it.

*nano hello.txt*, nano is a command line text editor. It works the same way as a desktop text editor like TextEdit or Notepad, except that it is accessible from the command line and only accepts keyboard input.

In *nano hello.txt*:

* The command *nano hello.txt* opens a new text file named *hello.txt* in the nano text editor.
* *Hello, I am nano* is a text string entered in *nano*.
* The menu of keyboard commands at the bottom of the window allow us to save changes to *hello.txt* and exit nano.

Thee following shortcuts are useful in *nano*:

* *Ctrl + O* saves a file. O stands for output. Again, not case sensitive.
* *Ctrl + X* exits the nano program. X stands for exit.
* Ctrl + G opens a help menu.

*nano .bash\_profile* is used to open and edit the bash profile in the directory.

*source .bash\_profile* is used to activate the changes made in *.bash\_profile* for the current session.

In *alias pd="pwd"* , the *alias* command creates the *alias pd* for the *pwd* command, which is then saved in the bash profile.

The *pd alias* will be available each time a new terminal session is opened, and the output of *pd* will be the same as the *pwd* command.

In *alias hy="history"*, *hy* command is set as an *alias* for the history command in the bash profile. The *alias* is then made available in the current session through *source*.

In *alias ll="ls -la"*, sets *ll* as an *alias for ls -la.* Once the *alias* is made available in the current session through *source*, the command *ll* now executes *ls -la* and outputs all contents and directories in long format, including all hidden files.

*Environment* variables are variables that can be used across commands and programs and hold information about the environment.

In *export USER="Jane Doe"* :

* The line *USER="Jane Doe"* sets the environment variable *USER* to a name *“Jane Doe”.* Usually the *USER* variable is set to the name of the computer’s owner.
* The line *export* makes the variable to be available to all child sessions initiated from the current session.
* At the command line, the command *echo $USER* returns the value of the variable. Note that *$* is always used when returning a variable’s value. Here, the command *echo $USER* returns the name set for the variable.

*PS1* is an environment variable that defines the makeup and style of the command prompt.

In *export PS1=">> "* :

* *export PS1=">> "* sets the command prompt variable and exports the variable. Here the default command prompt is changed from *$ to >>*.
* After using the *source* command, the command line displays the new command prompt.

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

The *echo $HOME* command displays the path of the home directory as output on the terminal.

*PATH* is an environment variable that stores a list of directories separated by a colon.

The command *echo $PATH* will print a list of directories separated by a colon. Each directory contains scripts for the command line to execute. The *PATH* variable simply lists which directories contain scripts.

The *env* command stands for “environment,” and returns a list of the environment variables for the current user.

The *env* command returns a number of variables, including *PATH, PWD, PS1*, and *HOME*.

The command *env | grep PATH* selects the value of a *PATH* environment variable. Here the standard output of *env* is “piped” to the *grep* command. *grep* searches for the value of the variable *PATH* and outputs it to the terminal. Note that this is the same output as echo *$PATH*.

Bash profile is used to configure the environment.

* The environment refers to the preferences and settings of the current user.
* The nano editor is a command line text editor used to configure the environment.
* **.bash\_profile** is where environment settings are stored. You can edit this file with nano.
* Environment variables are variables that can be used across commands and programs and hold information about the environment.
  + export VARIABLE="Value" sets and exports an environment variable.
  + USER is the name of the current user.
  + PS1 is the command prompt.
  + HOME is the home directory. It is usually not customized.
  + PATH returns a colon : separated list of file paths. It is customized in advanced cases.
  + env returns a list of environment variables. You can redirect the output, using grep to select the variable you want to see.

Here’s a quick table of Bash commands and their Windows Command Prompt equivalents if you want to try them out on Windows!

**Navigation**

A picture containing text, software, multimedia software, graphics software

Description automatically generated

**Manipulation**

A picture containing text, software, multimedia software, font

Description automatically generated

**Redirection**

A screenshot of a computer

Description automatically generated with medium confidence

**Environment**

A screenshot of a computer program

Description automatically generated with low confidence