Linux/Bash Terminal Essentials

A handy cheat sheet for navigating and manipulating the Linux/Bash terminal environment, covering essential commands, shortcuts, and scripting tips.

Basic Navigation & File Management

Navigation Commands

`pwd`	Print working directory (current directory).
`cd <directory>`	Change directory. Use `cd ..` to go up one level.
`ls`	List files and directories in the current directory.
`ls -l`	List files with detailed information (permissions, size, modification date, etc.).
`ls -a`	List all files, including hidden files (files starting with `.`).
`ls -t`	List files sorted by modification time (newest first).

File Operations

`mkdir <directory>`	Create a new directory.
`touch <file>`	Create an empty file or update the modification timestamp of an existing file.
`cp <source> <destination>`	Copy a file or directory. Use `cp -r` for recursive copying of directories.
`mv <source> <destination>`	Move or rename a file or directory.
`rm <file>`	Remove a file. **Warning**: This is permanent! Use `rm -r` for directories, and `rm -rf` to force removal.
`rmdir <directory>`	Remove an empty directory. Use `rm -r <directory>` to remove non-empty directories.

File Content Viewing

`cat <file>`	Display the entire content of a file.
`less <file>`	View file content page by page. Use `q` to quit.
`head <file>`	Display the first few lines of a file (default 10 lines).
`tail <file>`	Display the last few lines of a file (default 10 lines).
`tail -f <file>`	Display the last few lines and follow the file as it grows. Useful for log files.
`wc <file>`	Word count - displays number of lines, words, and characters in a file.

Searching & Text Manipulation

Searching

`grep <pattern> <file>`	Search for a pattern within a file. Use `grep -i` for case-insensitive search.
`grep -r <pattern> <directory>`	Recursively search for a pattern within all files in a directory.
`find <directory> -name <filename>`	Find files by name within a directory.
`find <directory> -type f`	Find all files within a directory.
`find <directory> -type d`	Find all directories within a directory.
`locate <filename>`	Find files by name using a pre-built database. Requires `updatedb` to update the database.

Text Manipulation

`sed 's/<old>/<new>/g' <file>`	Replace all occurrences of `<old>` with `<new>` in a file using stream editor.
`awk '{print $1}' <file>`	Print the first column of each line in a file using AWK.
`sort <file>`	Sort the lines of a file.
`uniq <file>`	Remove duplicate lines from a file (usually used with `sort`).
`cut -d '<delimiter>' -f <field>` `	Cut out specific fields from a file based on a delimiter.
`tr '[:lower:]' '[:upper:]' <file>`	Convert all lowercase characters to uppercase in a file.

Piping and Redirection

`|`	Pipe the output of one command to the input of another. **Example:** `ls -l | grep .txt` (list files and filter for .txt files)
`>`	Redirect the output of a command to a file, overwriting the file if it exists. **Example:** `ls > files.txt`
`>>`	Append the output of a command to a file. **Example:** `echo 'New entry' >> logfile.txt`
`2>`	Redirect standard error to a file. **Example:** `command 2> error.log`
`&>`	Redirect both standard output and standard error to a file. **Example:** `command &> output.log`
`<`	Redirect the content of a file to the input of a command. **Example:** `wc -l < file.txt`

System Information & Process Management

System Information

`uname -a`	Display kernel information.
`hostname`	Display the system's hostname.
`df -h`	Display disk space usage in a human-readable format.
`du -sh <directory>`	Display the disk usage of a directory in a human-readable format.
`free -m`	Display memory usage in megabytes.
`uptime`	Show how long the system has been running.

Process Management

`ps aux`	Display all running processes.
`top`	Display a dynamic real-time view of running processes.
`kill <PID>`	Terminate a process with the given PID (Process ID).
`kill -9 <PID>`	Forcefully terminate a process (use with caution).
`bg`	Put a stopped process in the background.
`fg`	Bring a background process to the foreground.

User Management

`whoami`	Display the current username.
`id`	Display user and group IDs.
`passwd`	Change the password for the current user.
`sudo <command>`	Execute a command with superuser privileges.
`su <username>`	Switch to another user.
`groups`	Display the groups the current user belongs to.

Bash Scripting Basics

Script Structure

All bash scripts should start with a shebang line, which tells the system which interpreter to use:

```
#!/bin/bash
```

Comments are denoted by `#`:

```
# This is a comment
```

Variables

Setting a variable:	`variable_name="value"` (no spaces around `=`) **Example:** `NAME="John Doe"`

Accessing a variable:	`$variable_name` or `${variable_name}`
	
	Example:
	`echo "Hello, $NAME"`

Environment Variables:	Variables that are available system-wide (e.g., `PATH`, `HOME`). Access them the same way as regular variables.
Read-only variables:	`readonly variable_name`

Conditional Statements

`if` statement:

```
if [ condition ]; then
  commands
elif [ condition ]; then
  commands
else
  commands
fi
```

`case` statement:

```
case variable in
  pattern1)
    commands
    ;;
  pattern2)
    commands
    ;;
  *)
    commands # Default
    ;;
esac
```

Looping

`for` loop:

```
for variable in item1 item2 ...;
do
  commands
done
```

Example:

```
for i in {1..5}; do echo $i; done
```

`while` loop:

```
while [ condition ]; do
  commands
done
```

`until` loop:

```
until [ condition ]; do
  commands
done
```

Functions

Defining a function:

```
function_name () {
  commands
}
```

Or:

```
function function_name {
  commands
}
```

Calling a function:

`function_name`

Example:

```
greet () {
  echo "Hello, $1"
}

greet John
```

Returning a value: Use `return` to return an exit status (0-255). Use `echo` to output a string value.