Navigating Linux System Commands

A guide for beginners to the Linux Shell and GNU coreutils

Sayan Ghosh

May 13, 2024

IIT Madras
BS Data Science and Applications

Disclaimer
This document is a companion activity book for the System Commands (BSSE2001) course taught by **Prof. Gandham Phanikumar** at **IIT Madras BS Program.** This book contains resources, references, questions and solutions to some common questions on Linux commands, shell scripting, grep, sed, awk, and other system commands.

This was prepared with the help and guidance of the course instructors:

Santhana Krishnan and **Sushil Pachpinde**

Colophon
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.
The source code of this book is available at:
`https://github.com/sayan01/se2001-book`
(You are welcome to contribute!)

Edition
Compiled on May 13, 2024

UNIX is basically a simple operating system, but you have to be a genius to understand the simplicity.

– Dennis Ritchie

Preface

Through this work I have tried to make learning and understanding the basics of Linux fun and easy. I have tried to make the book as practical as possible, with many examples and exercises. The structure of the book follows the structure of the course *BSSE2001 - System Commands*, taught by **Prof. Gandham Phanikumar** at **IIT Madras BS Program.** .

The book takes inspiration from the previous works done for the course,

- Sanjay Kumar's Github Repository
- Cherian George's Github Repository
- Prabuddh Mathur's TA Sessions

The book covers basic commands, their motivation, use cases, and examples. The book also covers some advanced topics like shell scripting, regular expressions, and text processing using `sed` and `awk`.

This is not a substitute for the course, but a companion to it. The book is a work in progress and any contribution is welcome at `https://github.com/sayan01/se2001-book`

Sayan Ghosh

Contents

Preface **v**

Contents **vii**

1 Basic Commands in Linux **1**
1.1 Navigating the File System . . . 1
1.2 File Permissions . . . 2
1.3 Inodes and Links . . . 2
1.4 System Management and Information . . . 3
1.5 Reading and Writing Files using cat . . . 4
1.6 Types of Files . . . 5
1.7 Types of Commands . . . 5
1.8 File Manipulation . . . 6

2 Shell Variables **9**
2.1 Creating Variables . . . 9
2.2 Printing Variables to the Terminal . . . 10
2.3 Bash Flags . . . 11
2.4 Signals . . . 11
2.5 Variable Manipulation . . . 12
2.6 Arrays . . . 14

3 Process Management **17**
3.1 What are processes? . . . 17

Appendix **19**

Notation **21**

Alphabetical Index **23**

List of Figures

List of Tables

Basic Commands in Linux 1

1.1 Navigating the File System

The file system can be navigated in the Linux command line using the following commands:

- **pwd**: Print the current working directory
- **ls**: List the contents of the current directory
- **cd**: Change the current working directory
- **mkdir**: Create a new directory
- **rmdir**: Remove a directory
- **touch**: Create a new file
- **rm**: Remove a file
- **pushd**: Push the current directory to a stack
- **popd**: Pop the current directory from a stack[1]

More details about these commands can be found in their respective man pages. For example, to find more about the **ls** command, you can type `man ls`

Question 1.1.1 What is the command to list the contents of the current directory?

Answer 1.1.1 `ls`

Question 1.1.2 What is the command to list the contents of the current directory including hidden files?

Answer 1.1.2 `ls -a`

Question 1.1.3 What is the command to list the contents of the current directory in a long list format? (show permissions, owner, group, size, and time)

Answer 1.1.3 `ls -l`

Question 1.1.4 What is the command to list the contents of the current directory in a long list format and also show hidden files?

Answer 1.1.4 `ls -al` or `ls -la` or `ls -l -a` or `ls -a -l`

Question 1.1.5 The output of `ls` gives multiple files and directories

1.1 Navigating the File System 1
1.2 File Permissions 2
1.3 Inodes and Links 2
1.4 System Management and Information 3
1.5 Reading and Writing Files using cat 4
1.6 Types of Files 5
1.7 Types of Commands 5
1.8 File Manipulation 6

1: **pushd** and **popd** are useful for quickly switching between directories in scripts.

in a single line. How can you make it print one file or directory per line?

Answer 1.1.5 `ls -1`
This can also be done by passing the output of `ls` to `cat` or storing the output of `ls` in a file and then using `cat` to print it. We will see these in later weeks.[2]

2: that is a one, not an L

1.2 File Permissions

Question 1.2.1 How to list the permissions of a file?

Answer 1.2.1 `ls -l`
The permissions are the first 10 characters of the output.
`stat -c %A filename` will list only the permissions of a file.
There are other format specifiers of `stat` to show different statistics which can be found in `man stat`.

Question 1.2.2 How to change permissions of a file? Let's say we want to change `file1`'s permissions to `rwxr-xr-` What is the octal form of that?

Answer 1.2.2 `chmod u=rwx,g=rx,o=r file1` will change the permissions of `file1`
The octal form of `rwxr-xr-` is 754.
So we can also use `chmod 754 file1`
Providing the octal is same as using `=` to set the permissions.
We can also use `+` to add permissions and `-` to remove permissions.

1.3 Inodes and Links

Question 1.3.1 How to list the inodes of a file?

Answer 1.3.1 `ls -i` will list the inodes of a file. The inodes are the first column of the output of `ls -i` This can be combined with other flags like `-l` or `-a` to show more details.

Question 1.3.2 How to create soft link of a file?

Answer 1.3.2 `ln -s sourcefile targetfile` will create a soft link of `sourcefile` named `targetfile`. The soft link is a pointer to the original file.

Definition 1.3.1 (Soft Links) Soft Links are special kinds of files that just store the path given to them. Thus the path given while making soft links should either be an absolute path, or relative **from** the location of the soft link **to** the location of the original file. It should not be relative from current working directory.[a]

[a] This is a common mistake.

Question 1.3.3 How to create hard link of a file?

Answer 1.3.3 `ln sourcefile targetfile` will create a hard link of `sourcefile` named `targetfile`. The hard link is same as the original file. It does not depend on the original file anymore after creation. They are equals, both are `hardlinks` of each other. There is no parent-child relationship. The other file can be deleted and the original file will still work.

Definition 1.3.2 (Hard Links) Hard Links are just pointers to the same inode. They are the same file. They are not pointers to the path of the file. They are pointers to the file itself. They are not affected by the deletion of the other file. When creating a hard link, you need to provide the path of the original file, and thus it has to be either absolute path, or relative from the current working directory, not relative from the location of the hard link.

Question 1.3.4 How to get the real path of a file?
Assume three files:

- **file1** is a soft link to **file2**
- **file2** is a soft link to **file3**
- **file3** is a regular file

Real path of all these three should be the same. How to get that?

Answer 1.3.4 `realpath filename` will give the real path of `filename`. You can also use `readlink -f filename` to get the real path.

1.4 System Management and Information

Question 1.4.1 How to print the current date and time in some custom format?

Answer 1.4.1 `date -d today +%Y-%m-%d` will print the current date in the format `YYYY-MM-DD`. The format can be changed by changing the format specifiers. The format specifiers are given in the `man date` page. The `-d today` can be dropped, but is mentioned to show that

the date can be changed to any date. It can be strings like '2024-01-01' or '5 days ago' or 'yesterday', etc.

Question 1.4.2 How to print the kernel version of your system?

Answer 1.4.2 `uname -r` will print the kernel version of your system. `uname` is a command to print system information. The `-r` flag is to print the kernel release. There are other flags to print other system information.
We can also run `uname -a` to get all fields and extract only the kernel info using commands taught in later weeks.

Question 1.4.3 How to see how long your system is running for? What about the time it was booted up?

Answer 1.4.3 `uptime` will show how long the system is running for. `uptime -s` will show the time the system was booted up.
The `-s` flag is to show the time of last boot.

Question 1.4.4 How to see the amount of free memory? What about free hard disk space? If we are unable to understand the big numbers, how to convert them to human readable format? What is difference between MB and MiB?

Answer 1.4.4 `free` will show the amount of free memory.
`df` will show the amount of free hard disk space.
`df -h` and `free -h` will convert the numbers to human readable format.
MB is Megabyte, and MiB is Mebibyte.
1 MB = 1000 KB, 1 GB = 1000 MB, 1 TB = 1000 GB, this is SI unit.
1 MiB = 1024 KiB, 1 GiB = 1024 MiB, 1 TiB = 1024 GiB, this is 2^{10} unit.

1.5 Reading and Writing Files using cat

Question 1.5.1 Can we print contents of multiple files using a single command?

Answer 1.5.1 `cat file1 file2 file3` will print the contents of `file1`, `file2`, and `file3` in the order given. The contents of the files will be printed one after the other.

Question 1.5.2 Can `cat` also be used to write to a file?

Answer 1.5.2 Yes, `cat > file1` will write to `file1`. The input will be taken from the terminal and written to `file1`. The input will be written to `file1` until the user presses `Ctrl+D` to indicate end of input. This is `redirection`, which we see in later weeks.

Question 1.5.3 `ls` can only show files and directories in the **cwd**[3], not subdirectories. True or False?

3: **cwd** means **Current Working Directory**

Answer 1.5.3 False. `ls` can show files and directories in the cwd, and also in subdirectories. The `-R` flag can be used to show files and directories in subdirectories, recursively.

1.6 Types of Files

Question 1.6.1 What types of files are possible in a linux file system?

Answer 1.6.1 There are 7 types of files in a linux file system:

- Regular Files (starts with `-`)
- Directories (starts with `d`)
- Symbolic Links (starts with `l`)
- Character Devices (starts with `c`)
- Block Devices (starts with `b`)
- Named Pipes (starts with `|`)
- Sockets (starts with `s`)

Question 1.6.2 How to know what kind of file a file is? Can we determine using its extension? Can we determine using its contents? What does *MIME mean?* How to get that?

Answer 1.6.2 The `file` command can be used to determine the type of a file.
The extension of a file does not determine its type.
The contents of a file can be used to determine its type.
MIME stands for Multipurpose Internet Mail Extensions.
It is a standard that indicates the nature and format of a document.
`file -i filename` will give the MIME type of `filename`.

1.7 Types of Commands

Question 1.7.1 How to create aliases? How to make them permanent? How to unset them?

Answer 1.7.1 `alias name='command'` will create an alias.
`unalias name` will unset the alias.
To make them permanent, add the alias to the `~/.bashrc` file.
The `~/.bashrc` file is a script that is executed whenever a new terminal is opened.

Question 1.7.2 How to run the normal version of a command if it is aliased?

Answer 1.7.2 `\command` will run the normal version of `command` if it is aliased.

Question 1.7.3 What is the difference between `which`, `whatis`, `whereis`, `locate`, and `type`?

Answer 1.7.3 Each of the commands serve a different purpose:

- `which` will show the path of the command that will be executed.
- `whatis` will show a short description of the command.
- `whereis` will show the location of the command, source files, and man pages.
- `locate` is used to find files by name.
- `type` will show how the command will be interpreted by the shell.

Exercise 1.7.1 Find the path of the `true` command using `which`. Find a short description of the `true` command using `whatis`. Is the executable you found actually the one that is executed when you run `true`? Check using `type true`

Definition 1.7.1 (Types of commands) A command can be an alias, a shell built-in, a shell function, keyword, or an executable. The `type` command will show which type the command is.

- **alias**: A command that is an alias to another command defined by the user or the system.
- **builtin**: A shell built-in command is a command that is built into the shell itself. It is executed internally by the shell.
- **file**: An executable file that is stored in the file system. It has to be stored somewhere in the **PATH** variable.
- **function**: A shell function is a set of commands that are executed when the function is called.
- **keyword**: A keyword is a reserved word that is part of the shell syntax. It is not a command, but a part of the shell syntax.

1.8 File Manipulation

Question 1.8.1 How to list only first 10 lines of a file? How about first 5? Last 5? How about lines 105 to lines 152?

Answer 1.8.1 `head filename` will list the first 10 lines of `filename`.
`head -n 5 filename` will list the first 5 lines of `filename`.
`tail -n 5 filename` will list the last 5 lines of `filename`.
`head -n 152 filename | tail -n 48` will list lines 105 to 152 of `filename`. This uses `|` which is a pipe, which we will see in later weeks.

Question 1.8.2 Do you know how many lines a file contains? How can we count it? What about words? Characters?

Answer 1.8.2 `wc filename` will count the number of lines, words, and characters in `filename`.
`wc -l filename` will count the number of lines in `filename`.
`wc -w filename` will count the number of words in `filename`.
`wc -c filename` will count the number of characters in `filename`.

Question 1.8.3 How to delete an empty directory? What about non-empty directory?

Answer 1.8.3 `rmdir dirname` will delete an empty directory.
`rm -r dirname` will delete a non-empty directory.

Question 1.8.4 How to copy an entire folder to another name? What about moving?
Why the difference in flags?

Answer 1.8.4 `cp -r sourcefolder targetfolder` will copy an entire folder to another name.
`mv sourcefolder targetfolder` will move an entire folder to another name.
The difference in flags is because `cp` is used to copy, and `mv` is used to move or rename a file or folder. The `-r` flag is to copy recursively, and is not needed for `mv` as it is not recursive and simply changes the name of the folder (or the path).

Shell Variables 2

2.1 Creating Variables 9
2.2 Printing Variables to the Terminal 10
2.3 Bash Flags 11
2.4 Signals 11
2.5 Variable Manipulation . . 12
2.6 Arrays 14

2.1 Creating Variables

Definition 2.1.1 (Environment Variables) An environment variable is a variable that is accessbile to all processes running in the environment. It is a key-value pair. It is created using the `export` command. It can be accessed using the $ followed by the name of the variable (e.g. $HOME) or using the `printenv` command. They are part of the environment in which a process runs. For example, a running process can query the value of the TEMP environment variable to discover a suitable location to store temporary files, or the HOME or USER variable to find the directory structure owned by the user running the process.

Definition 2.1.2 (Shell Variables) A shell variable is a variable that is only accessible to the shell in which it is created. It is a key-value pair. It is created using the = operator. It can be accessed using the $ followed by the name of the variable (e.g. $var). They are local to the shell in which they are created.

Question 2.1.1 How to create a shell variable? How to create an environment variable?

Answer 2.1.1 `var=value` will create a shell variable.
`export var=value` will create an environment variable.

Question 2.1.2 How to set the prompt string? How to set the secondary prompt string?

Answer 2.1.2 `PS1="newprompt"` will set the prompt string.
`PS2="newprompt"` will set the secondary prompt string. This is useful when writing multi-line commands.

Question 2.1.3 Where are commands searched for? How to add a new path to search for commands? What symbol is used to separate the paths?

Answer 2.1.3 The **PATH** variable stores the paths to search for commands.
`export PATH=$PATH:newpath` will add `newpath` to the **PATH** variable.

The colon `:` is used to separate the paths.

2.2 Printing Variables to the Terminal

Question 2.2.1 List the frequently used shell variables already existing in your shell.

Answer 2.2.1 Frequently used shell variables:

- **USER** stores the currently logged in user.
- **HOME** stores the home directory of the user.
- **PWD** stores the current working directory.
- **SHELL** stores the path of the shell being used.
- **PATH** stores the paths to search for commands.
- **PS1** stores the prompt string for the shell.
- **PS2** stores the secondary prompt string for the shell
- **HOSTNAME** stores the network name of the system

Question 2.2.2 List the special shell variables useful in scripts.

Answer 2.2.2 Special shell variables useful in scripts:

- **$0** stores the name of the script.
- **$1, $2, $3, ...** store the arguments to the script.
- **$#** stores the number of arguments to the script.
- **$*** stores all the arguments to the script as a single string.
- **$@** stores all the arguments to the script as array of strings.
- **$?** stores the exit status of the last command.
- **$$** stores the process id of the current shell.
- **$!** stores the process id of the last background command.
- **$-** stores the current options set for the shell.
- **$IFS** stores the Internal Field Separator.
- **$LINENO** stores the current line number of the script.
- **$RANDOM** stores a random number.
- **$SECONDS** stores the number of seconds the script has been running.

Question 2.2.3 How to print things to the terminal? How to print the value of a variable? How to print escape characters? How to supress the newline?

Answer 2.2.3 `echo "Hello World"` will print `Hello World` to the terminal.
`echo $var` will print the value of `var` to the terminal.
`echo -e "Hello\nWorld"` will print `Hello` and `World` in two lines.
`echo -n "Hello"` will print `Hello` without a newline.

Question 2.2.4 How to print all the environment variables? Only some variables?

Answer 2.2.4 `printenv` will print all the environment variables.
`env` will also print all the environment variables.
`printenv var1 var2 var3` will print only `var1`, `var2`, and `var3`.

Question 2.2.5 How to escape a character in a string? How to escape a variable in a string?

Answer 2.2.5 `echo -e "Hello\nWorld"` will print `Hello` and `World` in two lines.
`echo "Hello $var"` will print `Hello` and the value of `var`.

2.3 Bash Flags

Question 2.3.1 How to list the bash flags set in the shell? How to set or unset them? What does the default flags mean? (himBHs)

Answer 2.3.1 `echo $-` will list the bash flags set in the shell.
`set -o flag` will set the flag.
`set +o flag` will unset the flag.
The default flags are:
h: locate and remember (hash) commands as they are looked up.
i: **interactive** shell
m: monitor the jobs and report changes
B: **braceexpand** - expand the expression in braces
H: **histexpand** - expand the history command
s: Read commands from the standard input.

2.4 Signals

Remark 2.4.1 If you press `Ctrl+S` in the terminal, some terminals might stop responding. You can resume it by pressing `Ctrl+Q`. This is because `ixoff` is set. You can unset it using `stty -ixon`. This is a common problem with some terminals, thus it can placed inside the `/.bashrc` file.

Other Ctrl+Key combinations:

- **Ctrl+C**: Interrupt the current process, this sends the SIGINT signal.
- **Ctrl+D**: End of input, this sends the EOF signal.
- **Ctrl+L**: Clear the terminal screen.
- **Ctrl+Z**: Suspend the current process, this sends the SIGTSTP signal.

- **Ctrl+R**: Search the history of commands using reverse-i-search.
- **Ctrl+T**: Swap the last two characters
- **Ctrl+U**: Cut the line before the cursor
- **Ctrl+V**: Insert the next character literally
- **Ctrl+W**: Cut the word before the cursor
- **Ctrl+Y**: Paste the last cut text

2.5 Variable Manipulation

Question 2.5.1 How to remove a variable that is set? How to check if a variable is present?

Answer 2.5.1 `unset var` will remove the variable `var`.
`echo $var` will print if the variable `var` is present.
`test -v var` will check if the variable `var` is present.
`test -z $var+x` will check if the variable `var` is present.

Question 2.5.2 How to print a default value if variable is absent? if present? how to set a default value if variable is absent?

Answer 2.5.2 `echo $var:-default` will print `default` if `var` is absent.
`echo $var:+default` will print `default` if `var` is present.
`echo $var:=default` will set `var` to `default` if `var` is absent.

Remark 2.5.1 `echo $var:-default` will print `default` if `var` is absent **or** empty.
`echo $var-default` will print `default` if `var` is absent, but not if `var` is empty.
`echo $var:+default` will print `default` if `var` is present **and** not empty.
`echo $var+default` will print `default` if `var` is present, regardless of whether `var` is empty or not.

Question 2.5.3 How to print the number of characters in a variable? How about a substring of the variable?

Answer 2.5.3 `echo $#var` will print the number of characters in `var`.
`echo $var:0:5` will print the first 5 characters of `var`.

Remark 2.5.2 The second number in the substring is the length of the substring. Thus, `echo $var:5:3` will print the 3 characters starting

from the 5th character. If the length of substring is more than the length of the variable, it will print till the end. Thus, `echo $var:5:100` will print the characters from the 5th character till the end.

Definition 2.5.1 (Variable Match and Delete) We can match some part of the variable and delete it from left or right:
`${var%pattern}` will delete the shortest match of `pattern` from the end of `var`.
`${var%%pattern}` will delete the longest match of `pattern` from the end of `var`.
`${var#pattern}` will delete the shortest match of `pattern` from the start of `var`.
`${var##pattern}` will delete the longest match of `pattern` from the start of `var`.

Question 2.5.4 If we have a variable with value "abc.def.ghi.xyz". How can we get the following:

- "abc.def.ghi"
- "abc"
- "def.ghi.xyz"
- "xyz"

Answer 2.5.4 The variable is `var="abc.def.ghi.xyz"`.

- `echo ${var%.*}` will print "abc.def.ghi".
- `echo ${var%%.*}` will print "abc".
- `echo ${var#*.}` will print "def.ghi.xyz".
- `echo ${var##*.}` will print "xy

Question 2.5.5 How to replace some part of a variable by something else? How to do for all occurences?

Answer 2.5.5 `echo ${var/pattern/string}` will replace the first occurence of `pattern` with `string`.
`echo ${var//pattern/string}` will replace all occurences of `pattern` with `string`.

Question 2.5.6 How to make a variable lowercase? How to make it uppercase? How to do for entire string? How to do for first character? How to make a variable Sentence case? (First letter is capital, rest lower)

Answer 2.5.6 The following commands can be used to manipulate the case of a variable:

```
1 echo ${var,,} # will make the variable lowercase.
2 echo ${var^^} # will make the variable uppercase.
3 echo ${var,} # will make the first character lowercase.
```

```
4 echo ${var^} # will make the first character uppercase.
5 lower=${var,,} # will store the lowercase variable in lower.
6 echo ${lower^} # will make the first character uppercase. (
    Sentence case)
```

Question 2.5.7 How to make a variable integer only? How to make it lowercase only? Uppercase only? Read only?

Answer 2.5.7 `declare -i var` will make the variable integer only.
`declare -l var` will make the variable lowercase only.
`declare -u var` will make the variable uppercase only.
`declare -r var` will make the variable read only.

Question 2.5.8 How to remove those restrictions from a variable?

Answer 2.5.8 `declare +i var` will remove the integer only restriction.
`declare +l var` will remove the lowercase only restriction.
`declare +u var` will remove the uppercase only restriction.
`declare +r var` will not work, as read only cannot be removed.

Question 2.5.9 How to store output of a command in a variable?

Answer 2.5.9 `var=$(command)` will store the output of `command` in `var`.

2.6 Arrays

Question 2.6.1 How to create an array? How to access an element of an array? How to print all elements of an array?

Answer 2.6.1 `arr=(1 2 3 4 5)` will create an array.
`echo ${arr[2]}` will access the 3rd element of the array.
`echo ${arr[@]}` will print all elements of the array.

Question 2.6.2 How to get the number of elements in an array? How to get the indices of the array? How to add an element to the array?

Answer 2.6.2 `echo ${#arr[@]}` will print the number of elements in the array.
`echo ${!arr[@]}` will print the indices of the array.
`arr+=(6)` will add the element 6 to the array.

Question 2.6.3 What are associative arrays? How to declare and use them?

Answer 2.6.3 Associative arrays are arrays with key-value pairs.
`declare -A arr` will declare an associative array.
`arr[key]=value` will set the value of `key` to `value`.
`echo ${arr[key]}` will print the value of `key`.

Question 2.6.4 How to store output of a command in an array?

Answer 2.6.4 `arr=($(command))` will store the output of `command` in `arr`.

Process Management 3

3.1 What are processes?

3.1 What are processes? 17

Definition 3.1.1 (Process) A process is an instance of a program that is being executed. It contains the program code and its current activity. Depending on the operating system (OS), a process may be made up of multiple threads of execution that execute instructions concurrently. Several processes may be associated with the same program; for example, opening up several instances of the same program often means more than one process is being executed. Each process has its own 'process id' or **PID** to uniquely identify it.

Question 3.1.1 How to get a snapshot of all the processes running? What are the commonly used flags used with it?

Answer 3.1.1 The **ps** command is used to get a snapshot of all the processes running.

- **ps** will get a snapshot of all the processes running.
- **ps -e** will show all the processes.
- **ps -f** will show full format listing.
- **ps -l** will show long format listing.
- **ps -u** will show user-oriented format listing.
- **ps -x** will show processes without controlling terminals.
- **ps -a** will show all processes with a terminal.
- **ps -A** will show all processes
- **ps aux** is a common command to see all processes
- **ps –forest** will show the processes in a tree form

Appendix

Notation

The next list describes several symbols that will be later used within the body of the document.

c Speed of light in a vacuum inertial frame

h Planck constant

Greek Letters with Pronunciations

Character	Name	Character	Name
α	alpha *AL-fuh*	ν	nu *NEW*
β	beta *BAY-tuh*	ξ, Ξ	xi *KSIGH*
γ, Γ	gamma *GAM-muh*	o	omicron *OM-uh-CRON*
δ, Δ	delta *DEL-tuh*	π, Π	pi *PIE*
ϵ	epsilon *EP-suh-lon*	ρ	rho *ROW*
ζ	zeta *ZAY-tuh*	σ, Σ	sigma *SIG-muh*
η	eta *AY-tuh*	τ	tau *TOW (as in cow)*
θ, Θ	theta *THAY-tuh*	υ, Υ	upsilon *OOP-suh-LON*
ι	iota *eye-OH-tuh*	ϕ, Φ	phi *FEE, or FI (as in hi)*
κ	kappa *KAP-uh*	χ	chi *KI (as in hi)*
λ, Λ	lambda *LAM-duh*	ψ, Ψ	psi *SIGH, or PSIGH*
μ	mu *MEW*	ω, Ω	omega *oh-MAY-guh*

Capitals shown are the ones that differ from Roman capitals.

Alphabetical Index

preface, v