practical computing for biologists

Steven H. D. Haddock
The Monterey Bay Aquarium Research Institute, and University of California, Santa Cruz

Casey W. Dunn
Department of Ecology and Evolutionary Biology, Brown University

ISBN 978-0-87893-391-4

Sinauer Associates, Inc. • Publishers
Sunderland, Massachusetts U.S.A.

http://practicalcomputing.org/

Available at Sinauer and Amazon.

Appendix 2

REGULAR EXPRESSION SEARCH TERMS

Regular expressions—ways to perform adaptive searches and replacements—are described in Chapters 2 and 3. Here we provide a quick reference to some of the more common regular expression terms. This table and the text of the book itself do not encompass the entire range of regular expressions. There are many other useful constructs, for example, embedding miniature scripts into your replacement terms, and searching for A or B in a string using the syntax `(sword|jelly)fish`. If you would like to delve deeper, there are many online references, and there is even an in-depth reference guide built into the `Help` menu of `TextWrangler`.

There is some variation in the terms supported from program to program and from language to language. The most widespread terms, which can be used almost anywhere that regular expressions are supported, are the POSIX Extended Regular Expressions. These include `.`, `*`, `+`, `{}`, `()`, `[]`, `[^]`, `^`, `$`, `?`, and `|`. While quite a bit can be accomplished with the POSIX terms, in many implementations the language has been supplemented with some nonstandard terms. Most of these nonstandard terms are based on Perl regular expressions. These include many of the character class wildcards listed in the tables below, such as `\d`, `\w`, and `\n`. These extra wildcards make it easier to write clear regular expressions. Lack of support for Perl-like regular expressions is one of the most common causes of confusion when moving to a new programming context.

If you are using regular expressions in a new context but find that they don't behave as expected, or that they generate errors, check to see which regular expressions are supported by the tool you are using. POSIX does define its own set of wildcards, but the syntax is different from the Perl-style `\w` format that we use in this book. These wildcards include `[:digit:]` in place of `\d` and `[:alpha:]` instead of `\w` that we use in this book (though not including the digits). These POSIX character classes can be used in some contexts where Perl classes aren't available, including SQL queries and the command-line tool `grep`. If you don't want to switch between wildcard types, a more universal solution is to replace character class wildcards with an explicit character range, such as `[0-9]` or `[A-Z]`.

Wildcards	
`\w`	Letters, numbers and _
`.`	Any character except `\n` `\r`
`\d`	Numerical digits
`\t`	Tab
`\r`	Return character. Also used as the generic end-of-line character in TextWrangler
`\n`	Line-feed character. Also used as the generic end-of-line character in Notepad++
`\s`	Space, tab, or end of line
`[A-Z]`	A single character of the ranges indicated in square brackets
`[^A-Z]`	A single character including all characters *not* in the brackets. Note that this will include `\n` unless otherwise specified, and may cause you to match across lines
`\`	Used to escape punctuation characters so they are searched for as themselves, not interpreted as wildcards or special symbols
`\\`	The \ symbol itself, escaped
Boundaries	
`^`	Match the start of the line, i.e., the position before the first character
`$`	Match the last position before the end-of-line character

Quantifiers, used in combination with characters and wildcards	
`+`	Look for the longest possible match of one or more occurrences of the character, wildcard, or bracketed character range immediately preceding. The match will extend as far as it can while still allowing the entire expression to match.
`*`	As above, matches as many of the previous character to occur, but allows for the character not to occur at all if the match still succeeds
`?`	Modifies greediness of + or * to match the shortest possible match instead of longest
`{}`	Specify a range of numbers to repeat the match of the previous character. For example: `\d{2,4}` matches between 2 and 4 digits in a row `[AC]{4,}` matches 4 or more of the letter A or C in a row
Capturing and replacing	
`()`	Capture the search results between the parentheses for use in the replacement term
`\1` `$1`	Substitute the contents of the matched into the replacement term, in numerical order. Syntax depends on the text editor or language that you are using.

Appendix 3

SHELL COMMANDS

Terminal operations are described in Chapters 4–6, 16, and 20. Many of the built-in `bash` shell commands are summarized here for quick reference. To get more information about a command and its options, type `man`, followed by the name of the command. If you are not sure which command applies, you can also search the contents of the help files using `man -k` followed by a keyword term.

Command	Description	Usage
`ls`	List the files in a directory	`ls -la`
	Parameters that follow can be folder names (use `*` as a wildcard)	`ls -1 *.txt`
		`ls -FG scripts`
		`ls ~/Documents`
	`-a` Show hidden files	`ls /etc`
	`-l` Show dates and permissions	
	`-1` List the file names on separate lines. Useful as a starting point for regexp into a list of commands	
	`-G` Enable color-coding of file types	
	`-F` Show a slash after directory names	
`cd`	Change directory	
	Without a slash, names are relative to the current directory	`cd scripts`
	With a preceding slash (`/`) names start at the root level	`cd /User`
	Tilde (`~/`) starts at the user's home directory	`cd ~/scripts`
	Two dots (`..`) goes "up" to the enclosing directory	`cd My\ Documents`
	One dot refers to the current directory	`cd 'My Documents'`
	Minus sign goes to the previously occupied directory	`cd ../..`
	Use tab key (see below) to auto-complete partially typed paths	`cd ..`
	Use backslash before spaces or strange characters in the directory name, or put the whole name in quotes	`cd -`

Command	Description	Usage
pwd	Print the working directory (the path to the folder you are in)	
↑	↑ key to step back through previously typed commands The cursor can be repositioned with the ← and → keys, and commands can then be edited Press return from anywhere in the line to re-execute. On OS X you can also reposition by option-clicking at a cursor location	
tab	Auto-complete file, folder, or script names at the command line	cd ~/Doc tab
less	Show contents of a file, page by page These commands also apply to viewing the results of man While less is running: q Quit viewing space Next page b Back a page 15 g Go to line 15 G Go to the end ↑ or ↓ Move up or down a line /abc Search file for text abc n After an initial search, find next occurrence of the search item ? Find previous occurrence of the search item h Show help for less	less data.txt
mkdir	Make a new directory (a new folder)	mkdir scripts
rmdir	Remove a directory (folder must be empty)	rmdir ~/scripts
rm	Remove file or files Use the -f flag to delete without confirmation (careful!) Use the -r flag to recursively delete the files in a directory and then the directory itself	rm test.txt rm -f *_temp.dat
man	Show the manual pages for a Unix command Use -k to search for a term within all the manuals The result is displayed using the less command above, so the same shortcuts allow you to navigate through	man mkdir man -k date man chmod

Command	Description	Usage
cp	Copy file, leaving original intact Does not work on folders themselves Single period as destination copies file to current directory, using same name	cp test1.txt test1.dat cp temp ../temp cp ../test.py .
mv	Move file or folder, renaming or relocating it Unlike cp, this does work on directories	mv test1.txt test1.dat mv temp ../temp2
|	Pipe output of one command to the input of another command	history | grep lucy
>	Send output of a command to a file, overwriting existing files Do not use a destination file that matches a wildcard on the left side	ls -1 *.py > files.txt
>>	Send output of a command to a file, appending to existing files	echo "#Last line" >> data.txt
<	Send contents of a file into command that supports its contents as input	mysql -u root midwater < data.sql
./	Represents the current directory in a path—the same location as pwd Trailing slash is optional Can execute a file in the current directory even when the file directory is not included in the PATH	cp ../*.txt ./ ./myscript.py
cat	Concatenate (join together) files without any breaks. Streams the contents of the file list across the screen	cat README cat *.fta > fasta.txt
head	Show the first lines of a file or command Use the -n flag to specify the number of lines	head -n 3 *.fasta ls *.txt | head
tail	Show the last lines of a file or output stream Use the -n flag to specify the number of lines to show With a plus sign, skip that number of lines and show to the end. Use -n +2 to show from the second line of the file to the end, skipping one header line	tail -n 20 *.fta tail -n +3 data.txt
wc	Count lines, words, and characters in an output stream or file	wc data.txt ls *.txt | wc
which	Show the location of executable files in the system path	which man

Command	Description	Usage

`grep`	Search for phrase in a list of files or pipe and show matching lines:
	`grep -E "`*`searchterm`*`"` *`filelist`*
	Often used in conjunction with piped output: `command | grep searchterm`
	Use quotes around search terms, especially spaces or punctuation like >, `&`, `#`, and others
	To search for tab characters, type ctrl V followed by tab inside the quotes
	Optional flags:
	`-c` Show only a count of the results in the file
	`-v` Invert the search and show only lines that do not match
	`-i` Match without regard to case
	`-E` Use full regular expressions
	Terms should be enclosed in quotes. Use `[ ]` to indicate a character range rather than the wildcards of Chapters 2 and 3
	General wildcard equivalents:
	`\s  [[:space:]]`
	`\w  [[:alpha:]]`
	`\d  [[:digit:]]`
	`-l` List only the filenames containing matches
	`-n` Show the line numbers of the match
	`-h` Hide the filenames in the output	

`agrep`	Search for approximate matches, allowing insertions, deletions, or mismatched characters. (Must be installed separately.) See Chapter 21
	Optional flags include:
	`-d ","` Use comma as delimiter between records
	`-2` Return results with up to 2 mismatches. Maximum is 8 mismatches
	`-B -y` Return the best match without specifying a number of mismatches
	`-l` Only list file names containing matches
	`-i` Match without regard to case
		`agrep -d "\>" -B -y ATG seqs.fta`
		`agrep -3 siphonafore taxa.txt`

`chmod`	Change access permissions on a file (usually to make a script executable or Web accessible)
	First option is one of `u`, `g`, `o` for user, group, other
	Second option after the plus or minus is `r`, `w`, or `x`, for read, write, or execute. Can also use binary encoding as explained in Appendix 6
		`chmod u+x file.pl`
		`chmod 644 myfile.txt`
		`chmod 755 myscript.py`

Command	Description	Usage
`set`	Show environmental variables, including functions that have been defined	

`$HOME`	The environmental variable containing the path user's home directory	`echo $HOME`
		`cd $HOME`

`$PATH`	The user's `PATH` variable, where the directories to search for commands are stored	`export PATH=$PATH:/usr/local/bin`
`nano`	Invoke the text editor. Control key sequences include: ctrl X Exit `nano` (will be prompted to save) ctrl O Save file without exiting ctrl Y Scroll up a page ctrl V Scroll down a page ctrl C Cancel operation ctrl G Show help and list of commands	`nano filename.txt`
ctrl C	Interrupt the current process	

`sort`	Sort lines of a file
	`-k` *`N`* Sort using column number *N* instead of starting at the first character. Columns are delimited by a series of white space characters
	`-t ","` In conjunction with `-k`, use commas as the delimiter to define columns
	`-n` Sort by numerical value instead of alphabetical
	`-r` Sort in reverse order
	`-u` Return only one unique representative from a series of identical sorted lines
		`sort -k 3 data.txt`
		`sort -k 2 -t "," F1.csv`
		`sort -nr numbers.txt`
		`sort A.txt > A_sort.txt`

`uniq`	Return a single line for each consecutive instance of that line in a file or output stream. To remove all duplicates from anywhere in the file, it must be sorted before being piped to the `uniq` command
	Use `-c` flag to return a count along with the repeated element
		`uniq -c records.txt`
		`sort names | uniq -c`

Command	Description	Usage
cut	Extract one or more columns of data from a file -f 1,3 Return columns 1 and 3, delimited by tabs -d "," Use commas as the field delimiter instead of tabs. Used in combination with -f -c 3-8 Return characters 3 through 8 from the file or stream of data	cut -c 5-15 data.txt cut -f 1,6 data.csv cut -f2 -d ":" > Hr.txt
curl	Retrieve the contents of a URL from over the network. URL should be placed in quotes. Without additional parameters, will stream contents to the screen For some Linux versions, wget offers similar functionality See man curl for ways to send user login information at the same time -o Set the name of the output file to save individual files for the data. See #1 below -m 30 Set a time out of 30 seconds [01-25] In the URL, substitute two digit numbers from 01 to 25 into the address in succession {22,33} {A,C,E} Substitute items in brackets into URL #1 The substituted value, for use in generating the filename	curl "www.myloc.edu" > myloc.html curl "http://www.nasa.gov/weather[01-12]{1999,2000}" -m 30 -o weather#1_#2.dat
sudo	Run the command that follows as a superuser with privileges to write to system files	sudo python setup.py install sudo nano /etc/hosts
alias	Define a shortcut for use at the command line. To make persistent, add to startup settings file .bash_profile or equivalent	alias cx='chmod u+x'

function	Create a shell function—like a small script
	$1 is the first user argument supplied after the command is typed
	$@ is all the parameters—useful for loops as below
	Variable names are defined with the format NAME= with no spaces. They are retrieved with $NAME
	Save it in .bash_profile to make it permanent
		myfunction() {
		  # insert commands here
		  echo $1
		}

;	In a command or script, equivalent to pressing return and starting a new line	date; ls

Command	Description	Usage

for	Perform a for loop in the shell. Can be useful in the context of a function	for ITEM in *.txt; do
		  echo $ITEM
		done

if	An if statement in a shell function:
	if [test condition]
	then
	  # insert commands
	else
	  # alternate command
	fi
	Comparison operators are eq for equals, lt for less than and gt for greater than
		if [$# -lt 1]
		then
		  echo "Less than"
		else
		  echo "greater than 1"
		fi

` `	Backtick symbols surrounding a command cause the command to be executed and then substitute the output into that place in the shell command or script	cd `which python`/.. nano `which script.py`
host	Return IP number associated with a hostname, or the hostname associated with an IP address, if available	host www.sinauer.com host 127.0.0.1
ssh	Start a secure remote shell connection	ssh lucy@pcfb.org
scp	Securely copy files to or from a remote location	scp localfile user@host/path/remotefile scp user@host/home/file.txt localfile.txt
sftp	Start a file transfer connection to a remote site. The prompt changes to an ftp prompt, at which the following commands can be used: open From the prompt, open a new sftp connection get Bring a remote file to the local server put Place a local file on the remote system cd Change directory on the remote server lcd Change directory on the local machine quit Exit the sftp connection	sftp user@remotemachine
gzip gunzip zip unzip	Compress and uncompress files	gzip files.tar gunzip files.tar.gz unzip archive.zip
tar	Create or expand an archive containing files or folders -cf Create -xvf Expand -xvfz Expand and uncompress gzip	tar -cf archive.tar ~/scripts tar -xvfz arch.tar.gz

Command	Description	Usage
`&`	When placed at the end of a command, runs it in the background	
`ps`	Show currently running processes. Flags controlling the output vary greatly by system. Usually a good starting point is `-ax`. See `man ps` for more	`ps -ax | grep lucy`
`top`	Show current processes sorted by various parameters, most useful of which is processor usage `-u`	`top -u`
`kill -9`	Terminate a process emphatically, using its process ID. Retrieve PID from the `ps` or `top` command	`kill -9 5567`
`killall`	Terminate processes by name	`killall Firefox`
`nohup`	Run command in background and don't terminate it when logging out or closing the shell window Use in this odd format shown, to prevent program output to cause the command to quit	`nohup` *`command`* `2> /dev/null < /dev/null &`
ctrl Z	Suspend the operation to move it into the background or perform other operations	
`jobs`	Show backgrounded or suspended jobs, won't show normal active processes	
`bg`	Move a suspended process into the background. Optional number after it in the format `%1` will specify the job number	

`apt-get`
`yum`
`rpm`
`port`
	Package installers for various Unix distributions. Search for and install remote software packages. Typically used with `sudo`	`sudo apt-get install agrep`
		`yum search imagemagick`

Appendix 4

PYTHON QUICK REFERENCE

Conventions for this appendix

In the examples below, italicized terms are not real variable or function names, but are stand-ins for an actual name. If a function name is shown as `.function()` then the dot means it is used as a method, coming after the variable name, as in `MyString.upper()`.

Format, syntax, and punctuation in Python

- Indented lines define blocks of statements that are executed in loops, decisions, and functions.
- Comments are marked by # and extend from that symbol to the end of the line. Multi-line comments can be bracketed on both sides by three quote marks.
- To continue a statement on the next line, use the \ character at the end of a line.
- Parentheses `()` pass parameters to functions.[1]
- Square brackets [ ] define lists and retrieve subsets of values from strings, lists, dictionaries, and other types.
- Curly brackets `{}` define dictionary entries.

Python scripts begin with the shebang line, and can include an optional line to enable support of Unicode characters:

```
#! /usr/bin/env python
# coding: utf-8
```

[1] They also are used to define tuples, non-changeable list-like variables that we don't address in this book.