HuGen2080 book

Daniel E. Weeks

2024-01-04

Table of contents

Preface

This is a Quarto book created from markdown and executable code using Quarto within RStudio.

Book web site: https://danieleweeks.github.io/HuGen2080/

Book source code: https://github.com/DanielEWeeks/HuGen2080

Created by Daniel E. Weeks

Websites:

https://www.sph.pitt.edu/directory/daniel-weeks

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

This is a Quarto book created from markdown and executable code using Quarto within RStudio.

Book web site: https://danieleweeks.github.io/HuGen2080/

Book source code: https://github.com/DanielEWeeks/HuGen2080

Created by Daniel E. Weeks

Websites:

https://www.sph.pitt.edu/directory/daniel-weeks

2 Logistics

2.1 GitHub: Set up an account

Please go to https://github.com and set up a GitHub account.

Choose your GitHub user name carefully, as you may end up using it later in a professional context.

2.2 GitHub Classroom

As GitHub Classroom will be used to distribute course materials and to submit assignments, it would be best if you get git working on your own computer. The easiest way to do this is to install RStudio, R, and git on your computer.

Please follow the detailed instructions in https://github.com/jfiksel/github-classroom-for-students

In particular, see Step 5 re generating an ssh key so you don't need to login every time.

3 GitHub

3.1 GitHub Introduction lecture

Here's a recording of this lecture (32 minutes 8 seconds):

Recording

3.2 GitHub Introduction slides

PDF slide set

4 Git Commands

4.1 git - best practices

pull - work - commit - pull - push

- `git pull`
- Make changes
- `git commit` your changes to your local repository
- `git pull` the latest remote changes to your local repository
- `git push` your changes.

Pay attention to any error messages.

4.2 Outline of essential Git commands

Here's an outline of essential Git commands, initially created by ChatGPT:

4.2.1 Initialization and Configuration

- `git init`: Initializes a new Git repository in the current directory.
- `git config`: Configure Git settings.

4.2.2 Basic Workflow

- `git add`: Stage changes.
- `git commit -m "message"`: Commits staged changes with a descriptive message.

4.2.3 Remote Repositories

- `git clone`: Clones a remote repository to your local machine.
- `git push`: Send local changes to remote repository.
- `git pull`: Retrieve changes from remote.
- `git remote`: Manage remote repositories.

4.2.4 Status and Changes

- `git status`: Shows the current state of your working directory.
- `git diff`: Displays changes between working directory and the last commit.

4.2.5 History and Logs

- `git log`: View commit history.
- `git log --oneline`: Compact commit history.

4.2.6 Ignoring Files

- Create `.gitignore` file.

4.2.7 Branching

- `git branch`: List/create branches.
- `git checkout`: Switch branches.
- `git merge`: Merge branches.

4.2.8 Undoing Changes

- `git reset`: Unstage or reset changes.
- `git revert`: Create undoing commits.

4.2.9 Tagging

- `git tag`: Create and manage tags.

4.2.10 Stashing

- `git stash`: Temporarily store changes.

5 Basic Shell Commands

5.1 Acknowledgment and License

This chapter is a derivative of the Basic Shell Commands cheat sheet from the DEPRECATED-boot-camps/shell/shell_cheatsheet.md file created by Software Carpentry and is used under the Creative Commons - Attribution license CC BY 3.0

Minor section numbering and formatting changes were made here.

This chapter is licensed under the CC BY 3.0 license by Daniel E. Weeks.

5.2 Shell Basics:

Command	Definition
`.`	a single period refers to the current directory
`..`	a double period refers to the directory immediately above the current directory
`~`	refers to your home directory. *Note:* this command does NOT work on Windows machines (Mac and Linux are okay)
`cd`	changes the current directory to the directory `dirname`
`./dirname`	
`ls -F`	tells you what files and directories are in the current directory
`pwd`	tells you what directory you are in (`pwd` stands for *p*rint *w*orking *d*irectory)
`history`	lists previous commands you have entered. `history | less` lets you page through the list.
`man` *cmd*	displays the *man*ual page for a command.

5.3 Creating Things:

5.3.1 How to create new files and directories..

Command	Definition
`mkdir ./dirname`	makes a new directory called dirname below the current directory. *Note:* Windows users will need to use `\` instead of `/` for the path separator
`nano filename`	if `filename` does not exist, `nano` creates it and opens the `nano` text editor. If the file exists, `nano` opens it. *Note: (i)* You can use a different text editor if you like. In gnome Linux, `gedit` works really well too. *(ii)* `nano` (or `gedit`) create text files. It doesn't matter what the file extension is (or if there is one)

5.3.2 How to delete files and directories...

5.3.2.1 *Remember that deleting is forever. There is NO going back*

Command	Definition
`rm ./filename`	deletes a file called `filename` from the current directory
`rmdir ./dirname`	deletes the directory `dirname` from the current directory. *Note:* `dirname` must be empty for `rmdir` to run.

5.3.3 How to copy and rename files and directories...

Command	Definition
`mv tmp/filename .`	moves the file `filename` from the directory `tmp` to the current directory. *Note: (i)* the original `filename` in `tmp` is deleted. *(ii)* `mv` can also be used to rename files (e.g., `mv filename newname`
`cp tmp/filename .`	copies the file `filename` from the directory `tmp` to the current directory. *Note: (i)* the original file is still there

5.4 Pipes and Filters

5.4.1 How to use wildcards to match filenames...

Wildcards are a shell feature that makes the command line much more powerful than any GUI file managers. Wildcards are particularly useful when you are looking for directories, files, or file content that can vary along a given dimension. These wildcards can be used with any command that accepts file names or text strings as arguments.

5.4.1.1 Table of commonly used wildcards

Wildcard	Matches
`*`	zero or more characters
`?`	exactly one character
`[abcde]`	exactly one of the characters listed
`[a-e]`	exactly one character in the given range
`[!abcde]`	any character not listed
`[!a-e]`	any character that is not in the given range
`{software,carpentry}`	exactly one entire word from the options given

See the cheatsheet on regular expressions on the second page of this PDF cheatsheet for more "wildcard" shortcuts.

5.4.2 How to redirect to a file and get input from a file ...

Redirection operators can be used to redirect the output from a program from the display screen to a file where it is saved (or many other places too, like your printer or to another program where it can be used as input).

Command	Description
`>`	write `stdout` to a new file; overwrites any file with that name (e.g., `ls *.md > mardkownfiles.txt`)
`>>`	append `stdout` to a previously existing file; if the file does not exist, it is created (e.g., `ls *.md >> markdownfiles.txt`)
`<`	assigns the information in a file to a variable, loop, etc (e.g., `n < markdownfiles.md`)

5.4.2.1 How to use the output of one command as the input to another with a pipe...

A special kind of redirection is called a pipe and is denoted by `|`.

Command	Description
`|`	Output from one command line program can be used as input to another one (e.g. ls *.md | head gives you the first 5 `*.md` files in your directory)

5.4.2.1.1 Example:

```
ls *.md | head | sed -i `s/markdown/software/g`
```

changes all the instances of the word `markdown` to `software` in the first 5 `*.md` files in your current directory.

5.5 How to repeat operations using a loop...

Loops assign a value in a list or counter to a variable that takes on a different value each time through the loop. There are 2 primary kinds of loops: `for` loops and `while` loops.

5.5.1 For loop

For loops loop through variables in a list

```
for varname in list
do
    command1 $varname
    command2 $varname
done
```

where,

- `for`, `in`, `do`, and `done` are keywords
- `list` contains a list of values separated by spaces. e.g. `list` can be replaced by `1 2 3 4 5 6` or by `Bob Mary Sue Greg`. `list` can also be a variable:
- `varname` is assigned a value without using a `$` and the value is retrieved using `$varname`

–

```
list[0]=Sam
list[1]=Lynne
list[2]=Dhavide
list[3]=Trevor
.
.
.
list[n]=Mark
```

which is referenced in the loop by:

```
for varname in ${list[@]}
do
    command1 $varname
    command2 $varname
done
```

Note: Bash is zero indexed, so counting always starts at `0`, not `1`.

5.5.2 While Loop

While loops loop through the commands until a condition is met. For example

```
COUNTER=0
while [ ${COUNTER} -lt 10 ]; do
    command 1
    command 2
    COUNTER=`expr ${COUNTER} + 1`
done
```

continues the loop as long as the value in the variable COUNTER is less than 10 (incremented by 1 on each iteration of the loop).

- `while`, `do`, and `done` are keywords

5.5.2.1 Commonly used conditional operators

Operator	Definition
`-eq`	is equal to
`-ne`	is not equal to
`-gt`	greater than
`-ge`	greater than or equal to
`-lt`	less than
`-le`	less than or equal to

Use `man bash` or `man test` to learn about other operators you can use.

5.6 Finding Things

5.6.1 How to select lines matching patterns in text files...

To find information within files, you use a command called `grep`.

Example command	Description
`grep [options] day haiku.txt`	finds every instance of the string `day` in the file haiku.txt and pipes it to standard output

5.6.1.1 Commonly used `grep` options

	`grep` options
`-E`	tells grep you will be using a regular expression. Enclose the regular expression in quotes. *Note:* the power of `grep` comes from using regular expressions. Please see the regular expressions sheet for examples
`-i`	makes matching case-insensitive
`-n`	limits the number of lines that match to the first n matches
`-v`	shows lines that do not match the pattern (inverts the match)
`-w`	outputs instances where the pattern is a whole word

5.6.2 How to find files with certain properties...

To find file and directory names, you use a command called `find`

Example command	Description
`find . -type d`	`find` recursively descends the directory tree for each path listed to match the expression given in the command line with file or directory names in the search path

5.6.2.1 Commonly used `find` options

	`find` options
`-type [df]`	`d` lists directories; `f` lists files
`-maxdepth n`	`find` automatically searches subdirectories. If you don't want that, specify the number of levels below the working directory you would like to search
`-mindepth n`	starts `find`'s search `n` levels below the working directory

6 Summary

In summary, this book is a work in progress.

References