‹ (../04-redirection/index.html) Introduction to the Command Line for Genomics (../) › (../06-organi

Writing Scripts and Working with Data

Overview

Teaching: 20 min
Exercises: 20 min
Questions

- How can we automate a commonly used set of commands?

Objectives

- Use the `nano` text editor to modify text files.
- Write a basic shell script.
- Use the `bash` command to execute a shell script.
- Use `chmod` to make a script an executable program.

Writing files

We've been able to do a lot of work with files that already exist, but what if we want to write our own files? We're not going to type in a FASTA file, but we'll see as we go through other tutorials, there are a lot of reasons we'll want to write a file, or edit an existing file.

To add text to files, we're going to use a text editor called Nano. We're going to create a file to take notes about what we've been doing with the data files in `~/shell_data/untrimmed_fastq`.

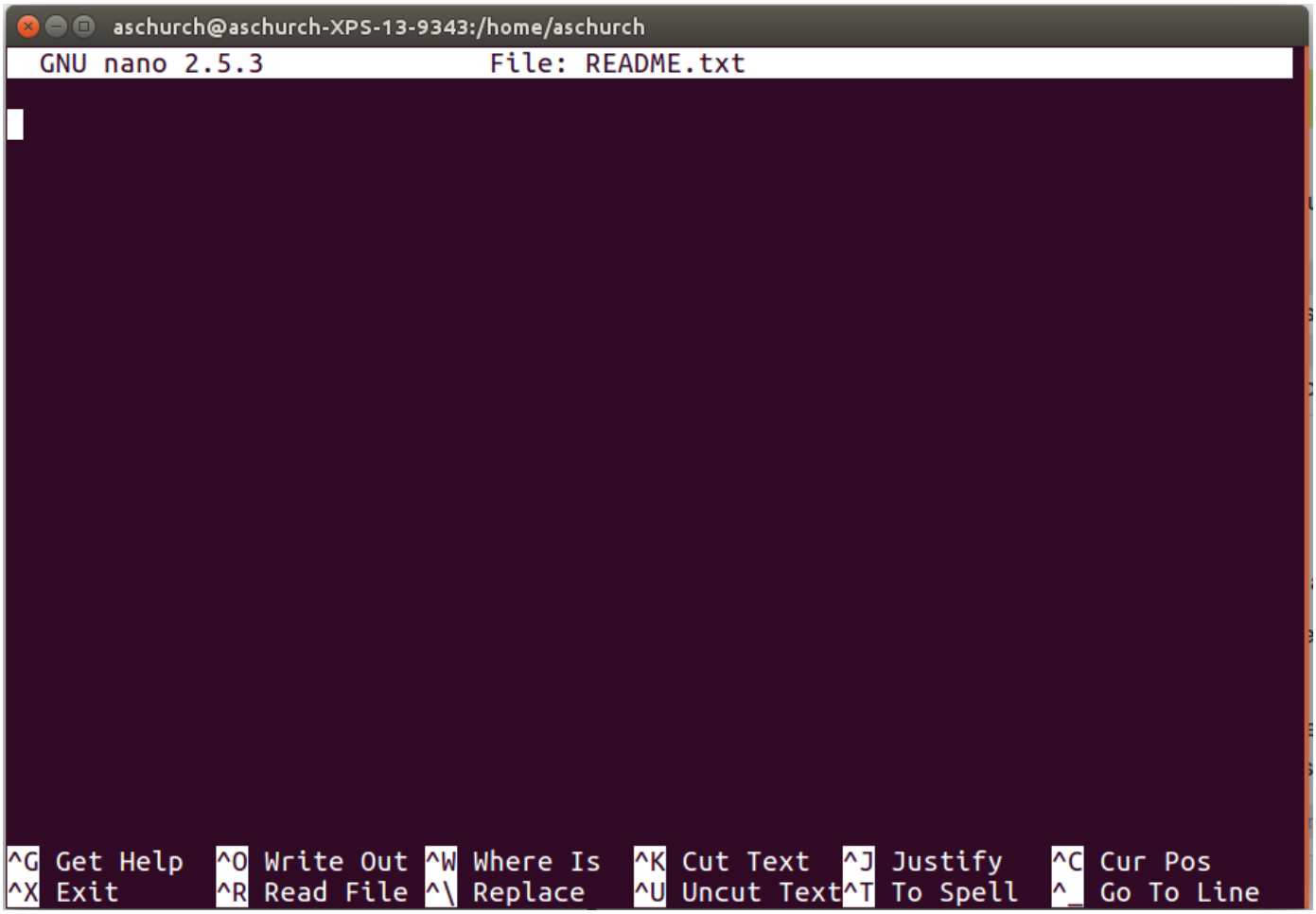
This is good practice when working in bioinformatics. We can create a file called `README.txt` that describes the data files in the directory or documents how the files in that directory were generated. As the name suggests, it's a file that we or others should read to understand the information in that directory.

Let's change our working directory to `~/shell_data/untrimmed_fastq` using `cd`, then run `nano` to create a file called `README.txt`:

Bash

```
$ cd ~/shell_data/untrimmed_fastq
$ nano README.txt
```

You should see something like this:



The text at the bottom of the screen shows the keyboard shortcuts for performing various tasks in `nano`. We will talk more about how to interpret this information soon.

📌 Which Editor?

When we say, "`nano` is a text editor," we really do mean "text": it can only work with plain character data, not tables, images, or any other human-friendly media. We use it in examples because it is one of the least complex text editors. However, because of this trait, it may not be powerful enough or flexible enough for the work you need to do after this workshop. On Unix systems (such as Linux and Mac OS X), many programmers use Emacs (http://www.gnu.org/software/emacs/) or Vim (http://www.vim.org/) (both of which require more time to learn), or a graphical editor such as Gedit (http://projects.gnome.org/gedit/). On Windows, you may wish to use Notepad++ (http://notepad-plus-plus.org/). Windows also has a built-in editor called `notepad` that can be run from the command line in the same way as `nano` for the purposes of this lesson.

No matter what editor you use, you will need to know where it searches for and saves files. If you start it from the shell, it will (probably) use your current working directory as its default location. If you use your computer's start menu, it may want to save files in your desktop or documents directory instead. You can change this by navigating to another directory the first time you "Save As…"

Let's type in a few lines of text. Describe what the files in this directory are or what you've been doing with them. Once we're happy with our text, we can press Ctrl-O (press the Ctrl or Control key and, while holding it down, press the O key) to write our data to disk. You'll be asked what file we want to save this to: press Return to accept the suggested default of `README.txt`.

Once our file is saved, we can use Ctrl-X to quit the editor and return to the shell.

Control, Ctrl, or ^ Key

The Control key is also called the "Ctrl" key. There are various ways in which using the Control key may be described. For example, you may see an instruction to press the Ctrl key and, while holding it down, press the X key, described as any of:

- Control-X
- Control+X
- Ctrl-X
- Ctrl+X
- ^X
- C-x

In nano , along the bottom of the screen you'll see ^G Get Help ^O WriteOut . This means that you can use Ctrl-G to get help and Ctrl-O to save your file.

Now you've written a file. You can take a look at it with less or cat , or open it up again and edit it with nano .

Exercise

Open README.txt and add the date to the top of the file and save the file.

Solution

Use nano README.txt to open the file.
Add today's date and then use Ctrl-X followed by y and Enter to save.

Writing scripts

A really powerful thing about the command line is that you can write scripts. Scripts let you save commands to run them and also lets you put multiple commands together. Though writing scripts may require an additional time investment initially, this can save you time as you run them repeatedly. Scripts can also address the challenge of reproducibility: if you need to repeat an analysis, you retain a record of your command history within the script.

One thing we will commonly want to do with sequencing results is pull out bad reads and write them to a file to see if we can figure out what's going on with them. We're going to look for reads with long sequences of N's like we did before, but now we're going to write a script, so we can run it each time we get new sequences, rather than type the code in by hand each time.

We're going to create a new file to put this command in. We'll call it bad-reads-script.sh . The sh isn't required, but using that extension tells us that it's a shell script.

Bash

```
$ nano bad-reads-script.sh
```

Bad reads have a lot of N's, so we're going to look for NNNNNNNNNN with grep . We want the whole FASTQ record, so we're also going to get the one line above the sequence and the two lines below. We also want to look in all the files that end with .fastq , so we're going to use the * wildcard.

Bash

```
grep -B1 -A2 NNNNNNNNNN *.fastq > scripted_bad_reads.txt
```

Type your grep command into the file and save it as before. Be careful that you did not add the $ at the beginning of the line.

Now comes the neat part. We can run this script. Type:

Bash

```
$ bash bad-reads-script.sh
```

It will look like nothing happened, but now if you look at `scripted_bad_reads.txt`, you can see that there are now reads in the file.

Exercise

We want the script to tell us when it's done.

1. Open `bad-reads-script.sh` and add the line `echo "Script finished!"` after the `grep` command and save the file.
2. Run the updated script.

Solution

```
$ bash bad-reads-script.sh
Script finished!
```

Making the script into a program

We had to type `bash` because we needed to tell the computer what program to use to run this script. Instead, we can turn this script into its own program. We need to tell it that it's a program by making it executable. We can do this by changing the file permissions. We talked about permissions in an earlier episode (http://www.datacarpentry.org/shell-genomics/03-working-with-files/).

First, let's look at the current permissions.

Bash

```
$ ls -l bad-reads-script.sh
```

Output

```
-rw-rw-r-- 1 dcuser dcuser 0 Oct 25 21:46 bad-reads-script.sh
```

We see that it says `-rw-r--r--`. This shows that the file can be read by any user and written to by the file owner (you). We want to change these permissions so that the file can be executed as a program. We use the command `chmod` like we did earlier when we removed write permissions. Here we are adding (`+`) executable permissions (`+x`).

Bash

```
$ chmod +x bad-reads-script.sh
```

Now let's look at the permissions again.

Bash

```
$ ls -l bad-reads-script.sh
```

Output

```
-rwxrwxr-x 1 dcuser dcuser 0 Oct 25 21:46 bad-reads-script.sh
```

Now we see that it says `-rwxr-xr-x`. The `x`'s that are there now tell us we can run it as a program. So, let's try it! We'll need to put `./` at the beginning so the computer knows to look here in this directory for the program.

Bash

```
$ ./bad-reads-script.sh
```

The script should run the same way as before, but now we've created our very own computer program!

You will learn more about writing scripts in a later lesson (https://datacarpentry.org/wrangling-genomics/05-automation/index.html).

Moving and Downloading Data

So far, we've worked with data that is pre-loaded on the instance in the cloud. Usually, however, most analyses begin with moving data onto the instance. Below we'll show you some commands to download data onto your instance, or to move data between your computer and the cloud.

Getting data from the cloud

There are two programs that will download data from a remote server to your local (or remote) machine: `wget` and `curl`. They were designed to do slightly different tasks by default, so you'll need to give the programs somewhat different options to get the same behaviour, but they are mostly interchangeable.

- `wget` is short for "world wide web get", and it's basic function is to *download* web pages or data at a web address.
- `cURL` is a pun, it is supposed to be read as "see URL", so its basic function is to *display* webpages or data at a web address.

Which one you need to use mostly depends on your operating system, as most computers will only have one or the other installed by default.

Let's say you want to download some data from Ensembl. We're going to download a very small tab-delimited file that just tells us what data is available on the Ensembl bacteria server. Before we can start our download, we need to know whether we're using `curl` or `wget`.

To see which program you have, type:

Bash

```
$ which curl
$ which wget
```

`which` is a BASH program that looks through everything you have installed, and tells you what folder it is installed to. If it can't find the program you asked for, it returns nothing, i.e. gives you no results.

On Mac OSX, you'll likely get the following output:

Bash

```
$ which curl
```

Output

```
/usr/bin/curl
```

Bash

```
$ which wget
```

Output

```
$
```

This output means that you have `curl` installed, but not `wget`.

Once you know whether you have `curl` or `wget`, use one of the following commands to download the file:

Bash

```
$ cd
$ wget ftp://ftp.ensemblgenomes.org/pub/release-37/bacteria/species_EnsemblBacteria.txt
```

or

Bash

```
$ cd
$ curl -O ftp://ftp.ensemblgenomes.org/pub/release-37/bacteria/species_EnsemblBacteria.txt
```

Since we wanted to *download* the file rather than just view it, we used `wget` without any modifiers. With `curl` however, we had to use the -O flag, which simultaneously tells `curl` to download the page instead of showing it to us **and** specifies that it should save the file using the same name it had on the server: species_EnsemblBacteria.txt

It's important to note that both `curl` and `wget` download to the computer that the command line belongs to. So, if you are logged into AWS on the command line and execute the `curl` command above in the AWS terminal, the file will be downloaded to your AWS machine, not your local one.

Moving files between your laptop and your instance

What if the data you need is on your local computer, but you need to get it *into* the cloud? There are also several ways to do this, but it's *always* easier to start the transfer locally. **This means if you're typing into a terminal, the terminal should not be logged into your instance, it should be showing your local computer. If you're using a transfer program, it needs to be installed on your local machine, not your instance.**

Transferring Data Between your Local Machine and the Cloud

These directions are platform specific, so please follow the instructions for your system:

Please select the platform you wish to use for the exercises: Windows ▾

Uploading Data to your Virtual Machine with PSCP

If you're using a PC, we recommend you use the *PSCP* program. This program is from the same suite of tools as the PuTTY program we have been using to connect.

1. If you haven't done so, download pscp from http://the.earth.li/~sgtatham/putty/latest/x86/pscp.exe (http://the.earth.li/~sgtatham/putty/latest/x86/pscp.exe)
2. Make sure the *PSCP* program is somewhere you know on your computer. In this case, your Downloads folder is appropriate.
3. Open the windows PowerShell (https://en.wikipedia.org/wiki/Windows_PowerShell); go to your start menu/search enter the term **'cmd'**; you will be able to start the shell (the shell should start from C:\Users\your-pc-username>).
4. Change to the Downloads directory:

Bash

```
> cd Downloads
```

1. Locate a file on your computer that you wish to upload (be sure you know the path). Then upload it to your remote machine **(you will need to know your AMI instance address (which starts with ec2), and login credentials)**. You will be prompted to enter a password, and then your upload will begin. **(make sure you substitute 'your-pc-username' for your actual pc username and 'ec2-54-88-126-85.compute-1.amazonaws.com' with your AMI instance address)**

Bash

```
C:\User\your-pc-username\Downloads> pscp.exe local_file.txt dcuser@ec2-54-88-126-85.compute-1.amazonaws.com:/home/dcuser/
```

Downloading Data from your Virtual Machine with PSCP

1. Follow the instructions in the Upload section to download (if needed) and access the *PSCP* program (steps 1-3)
2. Download the text file using the following command **(make sure you substitute 'your-pc-username' for your actual pc username and 'ec2-54-88-126-85.compute-1.amazonaws.com' with your AMI instance address)**

Bash

```
C:\User\your-pc-username\Downloads> pscp.exe dcuser@ec2-54-88-126-85.compute-1.amazonaws.com:/home/dcuser/shell_data/untrimmed_fastq/scripted_bad_reads.txt.

C:\User\your-pc-username\Downloads
```

Key Points

- Scripts are a collection of commands executed together.
- Transferring information to and from virtual and local computers.

‹ (../04-redirection/index.html)

› (../06-organi