Laboratory 3: BASH scripts, parallelism, and job control

Any combination of commands that can be entered in terminal's command–line can be automated with shell scripts. The best shell scripts automate repetitive tasks—this makes life easier for you and limits the number of mistakes that you can make. Some shell scripts are very simple—just one or two commands—others are highly complex—with hundreds of lines of code. By convention shell scripts use the extension '.sh' (even if they are bash scripts). The extension is not necessary for function, but it is a simple way of keeping organized.

All scripts are text files with two important attributes: (1) The first (shebang) line of the file should be `#!/bin/bash` in order to tell the computer to use 'bash' to interpret the commands that follow—other interpreters can be used in place of bash (e.g. `/usr/bin/perl`), but we will stick to bash for this lab. (2) The file must be executable. To make a file executable type `chmod +x file.sh` in the terminal.

Once automated, highly repetitive tasks can also often be made to run in parallel (i.e. several instances can be run at once). Parallelism can be accomplished on a single computer or multiple computers either manually or using a facilitating utility (e.g. 'xargs').

Tasks

(1) First install OpenSSL, if it is not installed already, by typing `sudo apt install openssl` in the terminal.

(2) Make a simple, but computationally intense, bash script.

- (a) Type `echo '#!/bin/bash' > hash.sh` in the terminal to create a functionless script file called 'hash.sh'.
- (b) Type `echo 'openssl speed 2>&1 | sha256sum' >> hash.sh` in the terminal.
- (c) Type `echo 'exit' >> hash.sh` in the terminal.
- (d) Type `chmod +x hash.sh` in the terminal to make the file executable.
- (e) Type `ls -l -h` in the terminal to check that the file is executable.
- (f) Type `./hash.sh` in the terminal to execute the script.
- (g) Answer question (1).

(3) Take control of the processes running on your computer—the brutal approach.

- (a) Execute 'hash.sh' in the terminal. Open a second terminal window and type `ps -a` in the new terminal.
- (b) In the fourth column ('CMD') you should see 'hash.sh', 'openssl', and 'sha256sum' among other things. This means that script is running.
- (c) Type `ps -a | grep openssl | awk '{print $1}' | xargs kill` in the terminal.
- (d) Go back to the first terminal and answer question (2).
- (e) Execute 'hash.sh' in the terminal. Open a second terminal window and type `killall openssl` in the new terminal.
- (f) Go back to the first terminal and answer question (3).

(4) Take control of the processes running on your computer—the nice approach.

(a) Type `./hash.sh &` in the terminal. Answer question (4).

(b) Type `top` in the terminal. Determine how much processing power 'openssl' is using.

(c) Type `q` in the terminal to exit top.

(d) Does your computer seem a little sluggish now? If you have multiple processing cores, the computer will not be slow. To make your computer (or virtual machine) grind to a halt, start more process like the one in (a) by typing `seq 1 $(( $(nproc) * 4 )) | xargs -I {} -P $(nproc) bash -c ' echo thread {} started; ./hash.sh'` in the terminal. Answer question (5).

(e) Type `top` in the terminal and determine how much processing power 'openssl' is using as well as the priority for the processes.

(f) Exit top and type `ps -a` in the terminal and find the PIDs for the 'openssl' processes.

(g) Type `renice -n +10 x` in the terminal where 'x' is a space separated list of process IDs ('PID' column) from the ps output.

(h) Type `top` in the terminal. Determine how much processing power 'openssl' is using as well as the priority for the processes now. Answer question (6).

(5) Retaining control of the processes running on your computer.

(a) Execute 'hash.sh' in the terminal and redirect the output to a file called 'hash.out' by typing `./hash.sh > hash.out` in the terminal.

(b) Close the terminal window. Click on 'Close Terminal' if/when prompted.

(c) Open a new terminal and examine 'hash.out'. Answer question (7).

(d) Execute 'hash.sh' as before, but use the nohup utility by typing `nohup ./hash.sh > hash.out` in the terminal.

(e) Close the terminal window. Click on 'Close Terminal' if/when prompted.

(f) Open a new terminal and examine 'hash.out'. Answer question (8).

(g) Execute 'hash.sh' using the nohup utility as a background task by typing `nohup ./hash.sh > hash.out &` in the terminal.

(h) Close the terminal window. Click on 'Close Terminal' if/when prompted.

(i) Open a new terminal and examine 'hash.out'. Answer question (9).

(6) A simple example of automating repetitive tasks: the partial recreation of a published analysis of 14 complete Lycopodiopsida plastid sequences following Mower et al. (2019).

(a) Install the NCBI GenBank Entrez toolkit by typing `sudo apt install ncbi-entrez-direct` in the terminal.

(b) To use the tools, you need an API key. Follow the instructions at https://support.nlm.nih.gov/knowledgebase/article/KA-05317/en-us to obtain a key (create an account if needed).

(c) Install the key by typing `echo 'export NCBI_API_KEY=your-api-key' >> .bashrc` in the terminal. Be sure to replace 'your-api-key' with your actual key (36 hex characters).

(d) Start a new terminal by closing the GUI window (click on 'Close Terminal' if/when prompted) and then opening a new one.

(e) To download the sequences, first create a polite bash download function by typing `downloadGenBank() { MINWAIT=3; MAXWAIT=7; sleep $((MINWAIT+RANDOM % (MAXWAIT-MINWAIT))); X=$(esearch -db nuccore -query "\"$1\"[Accession]" | efetch -format fasta); printf "${X}\n"; }` in the terminal. Answer question (10).

(f) Export your download function by typing `export -f downloadGenBank` in the terminal.

(g) Create a directory to store the sequences in by typing `mkdir MowerEtAl2019` in the terminal.

(h) Download the 14 sequences, two at a time, using your download function with xargs and save them in the MowerEtAl2019 directory by typing `echo -n 'AB197035 AY660566 GU191333 HM173080 KX426071 MH549637 MH549638 MH549639 MH549640 MH549641 MH549642 MH549643 MH598537 MK089531' | tr ' ' '\0' | xargs -0 -I {} -P 2 bash -c 'downloadGenBank "{}" > MowerEtAl2019/{}.fasta'` in the terminal. Answer question (11).

(i) Check that there are 14 fasta files in the MowerEtAl2019 directory and that each contains DNA sequence.

(j) To calculate the length and percent GC for each sequence type `echo 'code sequence length %GC'; find MowerEtAl2019 -type f -name '*.fasta' | xargs -I {} -P 1 bash -c 'awk -v bases=$(grep -v "^>" {} | tr -c -d ABCDGHKMNRSTVWY | wc -m) -v code=$(grep "^>" {} | awk "{print substr(\$2,1,2)substr(\$3,1,2)}") -v gc=$(grep -v "^>" {} | tr -c -d CG | wc -m) -v sequence=$(echo {} | perl -pe "s*MowerEtAl2019/**; s*.fasta**") "BEGIN{print code,sequence,bases,(int(1000*(gc/bases))/10)}"'` in the terminal. Answer question (12).

Questions (https://forms.gle/G5hpFzMMNmrV72Vr6)

(1) For task (2), explain exactly what 'hash.sh' does.

(2) For task (3)(c):

(a) Explain what the 'kill' utility does.

(b) Did your script terminate normally or did you kill it?

(c) Explain what each part in task (3)(c) does.

(3) For task (3)(e):

(a) Explain what the 'killall' utility does.

(b) Did your script terminate normally or did you kill it?

(c) Do you prefer the kill or killall approach?

(d) Why?

(4) For task (4)(a), what does each part of the command do?

(5) For task (4)(d), what does each part of the command do?

(6) For task (4)(h):

(a) What do the utilities nice and renice do?

(b) Has the amount of processor power used by 'openssl' substantially changed after setting nice to +10?

(c) What would happen if you set it to -10?

(d) Suggest a command string, using 'xargs', that you could use to avoid retyping all of the PIDs.

(7) For task (5)(b):

(a) Did hash.sh finish?

(b) Is hash.sh still running?

(c) How can you tell?

(8) For task (5)(e):

(a) Did hash.sh finish?

(b) Is hash.sh still running?

(c) How can you tell?

(d) What does the nohup utility do?

(9) For task (5)(h):

(a) Did hash.sh finish?

(b) Is hash.sh still running?

(c) How can you tell?

(d) What does the ampersand do?

(10) For task (6)(e):

(a) How long with each iteration of the downloadGenBank function wait before beginning a download?

(b) Why is this important?

(c) What will the downloadGenBank function likely return if 'phytoinformtics' is given as the argument?

(11) For task (6)(h):

(a) Explain what each step of the command does.

(b) What would you change if you wanted to download three sequences at a time in parallel?

(12) For task (6)(j):

(a) Explain what each step of the command does.

(b) Do the lengths calculated exactly match Table 1 of Mower et al. (2019)? Explain.

(c) Do the GC content calculated exactly match Table 1 of Mower et al. (2019)? Explain.

Literature cited

Mower, J. P., P.-F. Ma, F. Grewe, A. Taylor, T. P. Michael, R. VanBuren & Y.-L. Qiu. 2019. Lycophyte plastid genomics: extreme variation in GC, gene and intron content and multiple inversions between a direct and inverted orientation of the rRNA repeat. New Phytologist 222: 1061–1075.

Due at the start of class February 14.