Contents

About this course

This document provides complementary exercises that you can solve as you go through the bash guide. For each exercise, the sections of the course that are required to solve it are listed. Some also rely on external tutorials.

You don't need to solve all the exercises. You can pick the ones you find interesting. Some exercises, though, are labelled *recommended*. These either provide you with useful 'building blocks' for future exercises and scripts of your own, or test some basic knowledge that you will need.

We recommend you start with the easy exercises, and once you feel confident, continue with the more challenging ones. If you run into problems, you can ask our staff members for help.

Recommended chapters

Every exercise has a list of "requirements" that designate chapters which you will need to solve that specific exercise. However, there are some chapters that we recommend you read first, before you start any exercise, as they contain important knowledge that will prevent you from making simple "beginner mistakes".

Please read at least

- 2 Commands and Arguments
- 3 Variables and Parameters
- 5.1 Expansion and Quotes
- 5.2 Expansion Order

before commencing with the exercises.

A note about solutions

We have master solutions for every exercise, and the requirements and tips are designed to guide you towards that master solution. However, our master solutions are certainly not the only way of solving an exercise, and might not necessarily be the best way. If you think you're onto something, but the tips don't seem to make sense for your solution, you're encouraged to disregard them and solve the task your own way.

Exercises

Your first script (recommended)

Write a simple script that prints "Hello, World" to the console when called. Most of these steps are required for every script you write and won't be mentioned again in future exercises.

Requirements:

- 2.3 Scripts

Tips:

- Create a new file named `hello`
- The first line in this file should be a shebang that declares it as a bash script.
- Then, your script should invoke `echo` and print `Hello, World`.
- Save your script.
- Make your script file executable by invoking `chmod a+x hello` from your console.
- Now, try your script by invoking `./hello` from your console.

Count to 100 (recommended)

Write a script that prints the numbers from 1 to 100 to your terminal.

Requirements

- 3 Variables and Parameters
- 5.3 Brace Expansion
- 6.4 Conditional Loops

Set up a script directory for your user (recommended)

Create a directory in which you can put your bash scripts. The directory should be set up such that you can invoke all the scripts therein simply by typing their name in a console, rather than the full path.

Requirements:

- 3 Variables and Parameters
- 3.2 Environment Variables

Tips:

- Create a directory for your scripts. For example, you could use `~/scripts`.
- You will modify the environment variable `$PATH`. Append the path of your scripts directory to the end of `$PATH`, separated from the rest by a colon.
- You can append to a variable by simply using expansion. For example `variable="$variable:newtext"` will append `:newtext` to `variable`.
- **Messing with `PATH` can be dangerous!** Please test your modifications in a terminal before making them permanent.
- In order to make your new `PATH` permanent, you can edit your profile file. It should be stored in `~/.profile`. If it is not there, you could use `~/.bash_profile` or `~/.bashrc`.
- The new `PATH` will only become active when you log out and back in, or open a new terminal (depending on which file you used).

Parse some options (recommended)

Write a script that parses some options and prints out which options have been set and which not. While this is not immediately useful, you can reuse it for many other scripts.

Requirements:

- 6.4 Conditional Loops
- 6.5 Choices
- 7.1 Command-line Arguments

Tips:

- You can use `getopts` to simplify option parsing.
- `getopts` takes two arguments: an option string and a name.
  - The option string contains all the characters that are valid options. If a character is followed by a colon (:), that option is expected to have an argument.
  - The name is simply the name of a shell variable. `getopts` will set that variable to the option it found.
- `getopts` needs to be called repeatedly. It will produce a new option with every call, until there are none left, in which case it returns an error. That makes it ideal to be used in a `while` loop.
- In the loop body, you need to find out which option was set and print it. You can use a case block for this.
- Include at least one option that takes an argument. Your script should also print the argument if that option was set.
- An option's argument is automatically stored in the shell variable `$OPTARG`.
- If the user provides an invalid option, the variable specified by `getopts`' name argument is set to a question mark.
- Your script should provide a help option, `-h`, which prints all available options.
- Your script should print an error message if the user provides an invalid option.

Output all your arguments (recommended)

Write a script that outputs all the arguments given to it–similar to `echo`–but makes it clear where one argument ends and the next begins by enclosing them in angle brackets.

This script actually has some uses–you can find out how exactly your arguments were affected during word splitting, which is useful for debugging.

Requirements:

- 2.1 Preventing Word Splitting
- 3 Variables and Parameters
- 3.1 Special Parameters
- 6.4 Conditional Loops
- 7.1 Command-line Arguments

Tips:

- Make sure your arguments are always preserved exactly as they were given.
- You can use a loop to process each argument in turn.
- Note that angle brackets (`<` and `>`) have a special meaning in bash (namely redirection). That means you have to properly quote them.

Find big files

Write a script that takes a directory as an argument and prints the five biggest files found in that directory. You may print the 5 biggest files found recursively in that directory.

Requirements:

- 3.1 Special Parameters
- 6.1 Control Operators
- 7.4 Pipes

Tips:

- Your script should take exactly one argument (the directory). If it gets more or fewer arguments, make it print an error.
- Test whether the given argument is a directory. If not, print an error.

- Have a look at `find`. Make it print a file's size.

- Find a way to sort the output of `find`. Your script should output the five largest files found using `find`.

Self-reproducing script

Write a script that replicates itself, i.e. copies itself to a new file called backup.sh.

Requirements:

- 3.1 Special Parameters
- 7.3 Redirection

Tips:

- Do not hardcode the script's file name. Use a special parameter instead, so that your script still works when moved to another directory.
- This exercise can be done in multiple ways. Try the following:
    - Find and use a command that does exactly what you want.
    - Use `cat` and a redirection.

Make your bash prompt fancy

Your bash prompt is what you see in a terminal when you enter commands. It can be modified by setting the variable `PS1`. Try setting `PS1` to something different from your terminal.

In this exercise, you will customize your bash prompt. There are some things you need to know first, though:

- Within your command prompt, you can use several *escape sequences* that will be replaced by some interesting information:

Sequence	Decoding
\d	the date in "Weekday Month Date" format (e.g., "Tue May 26")
\e	an ASCII escape character (033)
\h	the hostname up to the first '.'
\H	the hostname
\j	the number of jobs currently managed by the shell
\n	a newline
\s	the name of the shell
\t	the current time in 24-hour HH:MM:SS format
\T	the current time in 12-hour HH:MM:SS format
\@	the current time in 12-hour am/pm format
\u	the username of the current user
\v	the version of bash (e.g., 2.00)
\w	the current working directory
\W	the basename of the current working directory
\!	the history number of this command
\#	the command number of this command
\$	if you're logged in as root, this prints a #, otherwise a $
\\	a backslash

- You can use the following codes to make your prompt colorful:

Code	Color
`\[\e[0;30m\]`	black
`\[\e[0;34m\]`	blue
`\[\e[0;32m\]`	green
`\[\e[0;36m\]`	cyan
`\[\e[0;31m\]`	red
`\[\e[0;35m\]`	purple
`\[\e[0;33m\]`	brown
`\[\e[0;37m\]`	gray
`\[\e[1;30m\]`	dark gray
`\[\e[1;34m\]`	light blue
`\[\e[1;32m\]`	light green
`\[\e[1;36m\]`	light cyan
`\[\e[1;31m\]`	light red
`\[\e[1;35m\]`	light purple
`\[\e[1;33m\]`	yellow
`\[\e[1;37m\]`	white
`\[\e[0m\]`	back to normal

Now that you know this, write a script that adjusts your bash prompt to your liking.

Requirements:

- 3 Variables and Parameters

Tips:

- This endeavour becomes a lot easier if you define variables for the color codes you need. Be aware that you need to quote these codes properly.
- Set your `PS1` to something you like. Use the variables you defined to change the color.
- At the end of your prompt, you should use the code for *back to normal*. If you don't do that, the output of your commands will be colored as well.
- Your prompt will change when you run the script from a terminal. If you want to make the changes permanent, you can execute your script from the `~/.bashrc` file–this file is executed whenever you open a terminal.

Maze generator

Write a script that randomly outputs / or \ in a loop in order to generate an ASCII maze.

Requirements

- 3.2 Environment Variables
- 6.1 Control Operators
- 6.4 Conditional Loops
- 8.3 Arithmetic Evaluation

Tips:

- First, find a way to generate random numbers.
- Based on that random number, print a \ or /.
- You want all your \s and /s to be on the same line. Look at `echo`'s `-n` option to achieve that.
- It's a good idea to slow down your script a bit. Check out the `sleep` command.

Simple backup script

Write a script that backs up all files in your home directory that have been modified in the last 24 hours and packs them in a tarball (which is a compressed `.tar.gz` file).

Requirements:

- 7.4 Pipes

Tips:

- A great utility for finding files with certain properties is the `find` command.
- By default, `find` also lists directories. You want to make sure it only lists files.
- Pipes can be used to redirect a command's stdout to another command's stdin. However, sometimes, you may need to use one command's stdout as *arguments* to another command. To achieve that, you can use `xargs`.

Screen brightness control

Write a script that takes a numeric argument and adjusts your screen brightness accordingly.

Requirements:

- 3 Variables and Parameters
- 7.1 Commands and Arguments

Tips:

- Have a look at `/sys/class/backlight`. Find a file inside that directory or one of its subdirectories that lets you control the brightness.
- Be sure not to set your brightness to 0 while experimenting, as that usually makes your screen turn pitch black.
- You need `sudo` to write to that file. Because of that, you can't use redirection. Have a look at `tee` to circumvent this problem.

Extend your script to allow absolute and relative changes (for example 'brightness 40' or 'brightness +10') and to prevent turning off your screen entirely by setting the brightness too low.

Automatic update script

Write a script that installs all available updates, then shuts down. If the update process failed, the script should still shut down your laptop, but write the error to a log file.

Requirements:

- 6.1 Control Operators
- 6.3 Conditional Blocks
- 7.3 Redirection

Tips:

- The easiest way is to create the log file either way, but remove it when the updates were successful.
- If there already exists a logfile, you should move it first so it doesn't get overwritten. Have a look at `mv`'s `-b` option for safe moving.
- It is possible use a control operator after a redirection.

Disable/Enable external monitor output

Write a script that changes your monitor settings. Depending on what you need, there are many variations for this: you might want a script to simply turn your external monitor off (even when it's still plugged in), or you can go for more advanced actions like turning off your internal monitor and only displaying on your external monitor.

Requirements:

- 6.3 Conditional Blocks

Tips:

- Have a look at `xrandr`, a command to manage your monitors. Find out how you can use it to turn outputs on or off.
- `xrandr` can also modify your monitor's orientation, resolution, or position.
- Create a script which uses xrandr to set up your monitors the way you like it.
- If you have several monitor settings that you use often, create a script for each of them.

- If you only have two settings and want your script to *toggle* between them, you can use `xrandr` in combination with `grep` to detect whether a specific monitor is on or not, and then act accordingly.
- Another possibility is to use xrandr to find out whether a specific monitor is connected or not, and then configure your setup according to that.

Display a random headline from Reddit

Write a script that pulls a random post title from a specific subreddit's front page and displays it. For example, you could use https://www.reddit.com/r/Showerthoughts.

Requirements:

- 3 Variables and Parameters
- 5.6 Command Substitution
- 7.4 Pipes

Tips:

- Reddit has a json API. Just add '.json' to the subreddit's URL. You might want to use `curl` to look at the json page.
- You can use `wget` to get a local copy of the json page and test it out locally on the file as input.
- You might need to change the user agent for `curl`. Have a look at the man page, and google for suitable user agents.
- Find a way to parse the json output in order to get a list of post titles.
- Pick a random post title by generating a random number and picking the corresponding title.
- You can use `sed` to format the output.

Wallpaper change every 10 minutes

Write a script that changes your wallpaper every 10 minutes.

Requirements:

- 4 Globbing
- 6.4 Conditional Loops

Tips:

- Find a way to set your wallpaper from command line.
- You could either use a list of files, or pick a random file from a specific directory.
- The `sleep` executable could come in handy.
- Make your script wait, then set the next wallpaper, then repeat.

Your own TODO program

Write a script that maintains a todo list for you. You should be able to display, remove and add todo items.

Requirements:

- 7.1 Command-line Arguments
- 7.2 File Descriptors
- 7.3 Redirection
- 9 Functions

Tips:

- Think about how you want to store your todo items. You could use plain text, or something fancy like json.
- Add one functionality at a time. First make a script that adds todo items, then extend it to display and finally delete them.
- Find a tool that is suitable for your storage format. (For example, if you use plain text, text parsers such as `sed` or `grep` are suitable.)
- Optionally, extend your script so you can add priorities or deadlines to your items.

Reminder script

Write a script that takes a number of minutes and a string as parameter. After that many minutes have passed, the script should display the string on your screen to remind you.

Requirements:

- 3.1 Special Parameters
- 7.1 Command-line Arguments
- 7.4 Pipes

Tips:

- Have a look at `notify-send`, a command that can display pop-ups on your screen.
- Have a look at `at`, a command that can be used to postpone execution of a script to a given time.
- Find out how to combine these two in order to create a reminder script.

Youtube downloader

Write a script that takes a string as input, searchs for that on youtube, and then downloads the audio of the first match.

Requirements:

- 3 Variables and Parameters
- 7.4 Pipes

Tips:

- You can use `curl` to find the search results. Have a look at `curl`'s `-data-urlencode` option.
- Browse to youtube and search for something. Look at the url of the results page and find out how to get that page using `curl` and its `-data-urlencode` option.
- Filter out all youtube video links from the results page. These links start with *'watch?v='* followed by an 11-character video ID.
- You can use `grep` for filtering. Have a look at it's `-o` and `-E` options.
- Have a look at `youtube-dl` and find out how to use it.
- Make your script download the first video you found using `youtube-dl`.

Web radio

Write a script that, when executed, starts playing your favorite webradio station.

Requirements:

- 6.3 Conditional Blocks

Tips:

- You will need a webradio station whose stream is accessible via an IP address. You can use the Xatworld radio search[1] to find such a station.
- Once you have the IP address, find out how to play it from the console.
- Extend your script to take an argument which is the name of a station, and make it play the station that you provided as argument. The script would have a predefined list of stations it can play.

Image resizer

Write a script that creates resized copies of all the pictures in the current folder.

Requirements:

- 4 Globbing
- 6.4 Conditional Loops

---

[1] https://www.xatworld.com/radio-search/

Tips:

- We recommend downloading some pictures from the web and putting them in a folder to experiment with (to prevent you from deleting your own pictures).
- Have a look at ImageMagick's `convert` command. Find out how to use it and resize a single image.
- Make sure your `convert` command only resizes images that are larger than the target size.
- `convert` can only operate on one image at a time. Your script will have to call it repeatedly.
- The resized copies should be stored in a separate, newly created subfolder.
- Find a way to iterate over all the files in a folder and call `convert` for each.
- Don't worry about files that are not images—`convert` will ignore them anyway.

Assignment-fetcher

Write a script that downloads all pdf files from a given website (for example, to get all your course assignments at once).

This exercise is a bit more involved, and is best solved using regular expressions (regex). Regex are not covered in this course, but you can find great online resources, like this Quick Start Tutorial[2]. Regex can be used in combination with `grep`, for example.

Requirements:

- 3 Variables and Parameters
- 5.6 Command Substitution
- 6.5 Choices
- 7.1 Command-line Arguments
- 7.4 Pipes

Tips:

- Find a way to isolate all URLs that end in '.pdf' from a website. You could use `grep` or `sed`.
- If you want to use `grep`, have a look at its `-o` and `-e` options.
- Depending on your course website, the URLs might be absolute (start with http) or relative (start with a / or a directory). Your script should be able to handle both.
- When you get relative URLs, you will have to prepend the base URL. Google for HTML relative URLs for more information.
- You can get the base url from the full url using `grep` with regular expressions.
- Process all the URLs such that you end up with absolute URLs only.
- Optionally, filter the URLs for a certain keyword that you can give your script as an argument. That way, you can prevent your script from downloading unrelated PDFs.
- Download all the files from the URL list to the current directory.
- Course slides at ETH are sometimes password protected. Have a look at wget's `-user` and `-password` options. Unfortunately, these don't work with the official nethz login, so if your course website uses that, you can't use a bash script.

---

[2]http://www.regular-expressions.info/quickstart.html