Version: 1.0

Selection

Welcome!

Summary

Welcome to 42! This activity will introduce you gently to the basic skills and knowledge required to start in good conditions.

#Shell #Unix #Git

Intellectual Property Disclaimer

All content presented in this training module, including but not limited to texts, images, graphics, and other materials, is protected by intellectual property rights held by Association 42.

Terms of Use:

- **Personal use:** You are permitted to use the contents of this module solely for personal purpose. Any commercial use, reproduction, distribution, modification, or public display is strictly prohibited without prior written permission from Association 42.
- **Respect for Integrity:** You must not alter, transform, or adapt the content in any way that could harm its integrity.

Protection of Rights:

Any violation of these terms constitutes an infringement of intellectual property rights and may result in legal action. We reserve the right to take all necessary measures to protect our rights, including but not limited to claims for damages.

For any questions regarding the use of the content or to obtain authorization, please contact: `legal@42.fr`

Contents

Chapter 1

Instructions

- Only this page will serve as a reference; do not trust rumors.
- Watch out! This document could potentially change up to one hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` consider a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- Additionally, your exercises will be checked and graded by a program called Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated, and there is no way to negotiate with it. So, to avoid bad surprises, be as thorough as possible.
- Shell exercises must be executable with /bin/sh.
- You cannot leave any additional files in your directory other than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...

Context

The C Piscine is intense. It's your first big challenge at 42 — a deep dive into problem-solving, autonomy, and community.

During this phase, your main objective is to build your foundation — through struggle, repetition, and especially **peer-learning** exchange.

In the AI era, shortcuts are easy to find. However, it's important to consider whether your AI usage is truly helping you grow — or simply getting in the way of developing real skills.

The Piscine is also a human experience — and for now, nothing can replace that. Not even AI.

For a more complete overview of our stance on AI — as a learning tool, as part of the ICT curriculum, and as a growing expectation in the job market — please refer to the dedicated FAQ available on the intranet.

Main message

- 👉 Build strong foundations without shortcuts.
- 👉 Really develop tech & power skills.
- 👉 Experience real peer-learning, start learning how to learn and solve new problems.
- 👉 The learning journey is more important than the result.
- 👉 Learn about the risks associated with AI, and develop effective control practices and countermeasures to avoid common pitfalls.

Learner rules:

- You should apply reasoning to your assigned tasks, especially before turning to AI.
- You should not ask for direct answers to the AI.
- You should learn about 42 global approach on AI.

Phase outcomes:

Within this foundational phase, you will get the following outcomes:

- Get proper tech and coding foundations.
- Know why and how AI can be dangerous during this phase.

Comments and example:

- Yes, we know AI exists — and yes, it can solve your projects. But you're here to learn, not to prove that AI has learned. Don't waste your time (or ours) just to demonstrate that AI can solve the given problem.
- Learning at 42 isn't about knowing the answer — it's about developing the ability to find one. AI gives you the answer directly, but that prevents you from building your own reasoning. And reasoning takes time, effort, and involves failure. The path to success is not supposed to be easy.
- Keep in mind that during exams, AI is not available — no internet, no smartphones, etc. You'll quickly realise if you've relied too heavily on AI in your learning process.
- Peer learning exposes you to different ideas and approaches, improving your interpersonal skills and your ability to think divergently. That's far more valuable than just chatting with a bot. So don't be shy — talk, ask questions, and learn together!
- Yes, AI will be part of the curriculum — both as a learning tool and as a topic in itself. You'll even have the chance to build your own AI software. In order to learn more about our crescendo approach you'll go through in the documentation available on the intranet.

✓ Good practice:

> I'm stuck on a new concept. I ask someone nearby how they approached it. We talk for 10 minutes — and suddenly it clicks. I get it.

✗ Bad practice:

> I secretly use AI, copy some code that looks right. During peer evaluation, I can't explain anything. I fail. During the exam — no AI — I'm stuck again. I fail.

Chapter 2

Forewords

If you haven't already, now is a great time to introduce yourself to the people seated next to you!

Chapter 3

Tutorial: The tree of Life

Welcome! Before we dive right in, do you know how to navigate the files and folders present on your system? Maybe you already know what is a file and a directory, but wait! There's more! Together with the person next to you, look up what is the **UNIX directory tree**. Identify and understand the concepts of **file**, **directory**, **tree structure**, **root**.

Once you both feel comfortable with these notions, turn to another person next to you and check your understanding with them.

Chapter 4

Tutorial: Shell we dance?

- Find out how to open a terminal. If you find yourself facing a black window that seems to be waiting for you to type something into it, you're on the right track.

- When you opened your **terminal**, it automatically launched a program that will be your new best friend from now on: the **shell**. It's this shell that will read the **commands** you type at the **command prompt** and execute them. If no more programs run in your terminal, the window closes. Find the command to quit the shell and close the terminal.

Clicking on the little red cross in the window is not the solution you're looking for.

- Open a new a terminal, then find the command that displays the name of the **current working directory**. This directory is your user home. It's yours and is the default directory when you open a terminal.

- Commands can also accept **command options** to control and customize their behaviours. In particular, all commands availalbe accept an option to display a help message. Find how to display the help message of the command that displays the current working directory.

Not sure where to start? Ask the person next to you to look this up together.

- Your shell can do much more than just diplaying the current working directory. For example it can also **list** the content of the current working directory! Find the command that lists the content of the current working directory and try it.

- This command is a good instance of a command that can accept many options to control its behaviour. In particular, that command has many options used to control the formating of its output. Find the option that formats the output as a **long listing**.

- Now that you know that `ls` accepts options, it's time to discover an interesting feature of UNIX file systems: hidden files and directories. No worries. there is no hacking involved (yet...), this feature is merely used to keep some less interesting files out of the default listing. However, it is possible to display them anyway. Find a way to display the hidden files and folders within your current working directory.

You should check with the person next to you if they know what makes a hidden file or folder hidden.

- Find the command to **change directory**. Navigate to the `root directory (/)`, then **list** its content. Do you recognize some directories from your home `path`?
- Navigate to a directory called `Users` (or similar), then **list** its content. Do you recognize some names?
- Minimize the terminal and, on your `Documents`, click on the right button and select `"New Folder"`, name it `I_love_42`. Now, go back to your terminal and change the directory to your `Documents`, which is placed in your **home directory**. Once there, check if the folder you have just created is present.
- Create another directory on your `Documents`, but this time you have to **make the directory** using your terminal. Name it: `I_adore_42_even_more_now`.
- Check that the folder appeared on your `Documents`. What happened?
- Create a file named `HelloWorld.txt` inside your directory. Write some text in it using a text editor available in your terminal. After doing so, list in long format what is in your folder. Ask your peer on the right what is the code `"rwrr"` that appears at the beginning of the line.
- If their explanation was correct, you should have learnt about the **UNIX access rights** and that now you are able to edit the file you just created. Write inside of it `'Hello World'`. There are many ways to edit files; take some time to find out which one will be more convenient in the future.

Hint: it is not **echo**. But some **text editors** are available in your terminal.

- Display the content of your file using the appropriate command.
- Find the command that allows you to **remove a directory** and delete the one(s) you have just created.

In a programmer's life, reading the **manual** should become a reflex! Look for what a manual is and then read the manual of the command **man**. Always do `man <command>` before using a command for the first time.

Chapter 5

Tutorial: Let's GIT !

Before starting, make sure you've completed the shell tutorial and are comfortable with basic terminal commands.

- **Git** is a version control system that will store your work for each project. Research what is a **repository**, a **version control system**, and understand the basic concept of **Git**. Discuss with your neighbor: why might programmers need to save different versions of their code?

  

  Think about writing a document - wouldn't you want to keep previous versions in case you make a mistake?

- **Git** is like a time machine for your code. It saves snapshots of your work called **commits**. A **repository** is like a project folder that **Git** watches for changes.
- Each of your project gives you a dedicated repository on a **Github** server. Find your tutorial project repository URL on the intranet and copy it.
- Most of git commands can be called using the following syntax `git <command> <arguments>`. Find out what is the command to **clone** your repository.
- Try the following command :

  Terminal:
  ```
  ?> git clone <your_repository_url>
  ```

  The command should fail. Do you know why ?
- Your repositories are stored on **Github** servers. Check with your peers if you know other versions control platforms for Git.
- Your repositories are stored on an external server. In order to access from your workstation to your repositories, you need to understand first what is a **SSH key**.
- Once you have done some research about what is a SSH key, you will have to find out how to add it to your Github account. Some of your peers should already know how to do it.

Before adding a key, you will also need to find a way to **generate** a new **SSH key**.

- What is the name of the command to generate a new SSH key pair ?
- Once you have generated a new SSH key pair, display your public key (the part you can share) with:

Terminal:
```
?> cat ~/.ssh/id_rsa.pub
```

This output is the part you need to store in other services (like version control servers). It is usually done in the `Settings` section. This tells the service "trust this computer".

Remember : SSH keys work in pairs: keep your private key secret, share your public key to grant access. You must be sure to have fulfill this step before moving to the folowwing parts. You won't be ablte to validate any exercise without this.

- Once you are sure you have added your `SSH key` to your `Github` account, try again to **clone** your repository for this project.

Terminal:
```
?> git clone <your_repository_url>
```

- Navigate into the created folder. What's the folder name? List its contents including hidden files.
- You should see a hidden **.git** folder - this is where `Git` stores all its magic! Check what Git thinks about your repository:

Terminal:
```
?> git status
```

This command is used to see the difference between your local changes and the repository on `Github`.

- Create a simple test file. For example :

Terminal:

```
?> echo ‘‘Hello Git!’’  > test.txt
```

- Run the command that allows you to check the **status** of your repository. What have changed ?
- Read the message carefully. Explain to one of your mate what **untracked file** means.
- Run the command to **track** your file and check the **status** again. What has changed?

Tracking a file is simply the action to tell `Git` to **add** and its actual local changes they are ready to be saved.

- What you have just done is to change the **status** of your file from **untracked** to **staged**: `Git` saved a local copy of the file at its current state and is now ready to keep track of the modifications. Let’s check it! Modify your file and check again the **status**.
- You can see how git tells you that you modified your file and gives you two options: you can either save the modifications by running the command you used before or discard your local changes. Let’s do the second, check what happened with your changes and check the status again.
- When adding a file git saves its state locally, but the server does not know for the moment that you created a file. You need to tag your changes, then send them to the server. Look for a way to **commit** (save) your changes with a descriptive message.

We recommend you to be explicit with the message, describing what is the content you are committing.

- Once you have used the command to **commit**, check again the status of your files and verify how it has changed. Now, find the command that allows you to check the list of your **commits**.

`Git` **logs** everything you **commit** to the server.

Do you know what the `(HEAD -> main)` part means ?

- If you find in the history the presence of your commit, it means your changes are saved locally. But the server doesn’t know it yet. Look for the command to **push** your commit to the server.
- By default, when cloning a repository you will be on the `main` branch, and since we didn’t change it, you are still there. Run the command to push your commits to the server. (FYI: you are on **remote** ‘‘`origin`’’).

- Check the commit list again, what has changed? Notice the date, the author, and your comment.
- To verify it worked, go to your home directory and **clone** your repository again in a different folder. Your file should be there! You can also check the commits, any surprise?
- Last but not least, remember that your repository should not contain unnecessary files or folders. So remove your file from your local directory, and proceed to save your changes :

Terminal:

```
?> rm test.text
?> git add test.text
?> git commit m “Remove test file”
?> git push origin main
```

- Once you have practiced this basic Git workflow a few times, you should be ready to start working on your exercises ! Just remember : your repository should only contain files needed for your project. And do not hesitate to validate your understanding with your peers. Have fun !

Chapter 6

Exercise 0: Z

	Exercise0	
Z		
Directory: *ex0/*		
Files to Submit: `z`		
Authorized: `None`		

- Create a file called `z` that returns "Z", followed by a new line, whenever the command `cat` is used on it.

Terminal Output

```
?>cat z
Z
?>
```

Don't you know where to start and what a terminal is? Ask the people around you and figure it out together!

Chapter 7

Exercise 1: SSH me!

	Exercise1	
SSH Key		
Directory: *ex1/*		
Files to Submit: `id_rsa_pub`		
Authorized: `None`		

Create your own SSH key. Once it is done:

- Add your public key to your repository, in a file name id_rsa_pub

- The file's name was not chosen randomly.
- Make sure you understand the difference between the public key, and the private key.

Did you check with your neighbors ?

Chapter 8

Submission and Peer Evaluation

Submit your assignment in your `Git` repository as usual. Only the work in your repository will be evaluated during the defense. Make sure to double-check the names of your files to ensure they are correct.

You need to submit only the files requested by this project's subject.