Git: installation and exercises

Learning goals

Before the next day, you should have achieved the following learning goals:

- Understand the concept of source code version control.
- Use the basic git commands: commit, push, pull, diff, status, log.
- Use `github.com` to create an account, create a repo, clone it, and push changes onto it.
- Understand what an enviroment variable is.
- Be able to set and modify an environment variable.
- Understand what `PATH` is for and change or update it.

1 Install Git in your account

If you want to use Git (and you do!) you need to install it in your computer. Git is not installed in the labs by default, so you will need to install it in your personal account. You have more than enough space. (Hint: your drive is probably H: or I:).

Remember that Git is free open-source software so you can get it easily online. (Hint: searching "git windows download" will probably get good results)

Once you have installed Git, check that everything works by going to the relevant folder and executing `git`. You should see a listing of all the possible commands. (Hint: the folder you are looking for is called `bin`, for "binary file"; this is a typical name for the folder that stores the executable files of a project).

2 Add git.exe to your path

If you have successfully completed the previous exercise, you can execute git from its own folder. In a real environment, however, you want to execute Git from your project folder, wherever this is. In order to execute a program from a different folder, you need to add it (e.g. `git.exe`) to your PATH.

2.1 What is the PATH?

Every program that you execute in Windows must be either (a) called absolutely, with its fully qualified name (e.g. `c:\windows\notepad.exe`), (b) be in the current folder, or (c) be included in the PATH.

When you type a command that is not a fully qualified name, Windows will look for it in the current folder. If it is not there, it will look for it in all the folders in the PATH. If it cannot find it after all these steps, it will complain:

```
'.....' is not recognized as an internal or external command,
operable program or batch file.
```

	Windows	Unix
Read value	%VARIABLE%	$VARIABLE
Print value	echo %VARIABLE%	echo $VARIABLE
Write value	set VARIABLE=...	set VARIABLE=...
PATH separator	semicolon (;)	colon (:)

Table 1: Environment variables in Windows and Unix

2.2 How can I check what my PATH is?

Very easy. Just type 'echo %PATH%' on the command line (on Windows systems) or 'echo $PATH' (on Unix systems: Linux, Max, etc).

You can follow the same procedure to see the value of all other environment variables, e.g. echo %SOME_VARIABLE%, echo $SOME_VARIABLE.

2.3 How can I edit the PATH?

The right way of doing it requires to have Administrator rights, so you probably cannot do it in the lab[1]. You can check online how to do it, it is explained in thousands of places (googling "change path windows 7/10/Vista/XP/whatever" will probably get you good results).

In the lab, you have to do it manualy by using the command 'set' (this is the same in Windows and in most Unix systems):

```
> set PATH=newPath
```

where 'newPath' is the new value that you want for this environment variable. This will *overwrite* your PATH in the same way that writing `i=1` in Java overwrites the value of the variable `i`.

Sometimes this is not what you want: you want to add things to your PATH, keeping the old list of folders that you already had. This is easy too; the only thing you have to know is that the old value of PATH can be accessed with %PATH% (Windows) or $PATH (Unix). In other words, if you type...

```
> set PATH=%PATH%
```

...you will have the same path (i.e. you have overwritten it with the same value). Now it is easy to add things. You have probably noticed that folders are separated by a semicolon (;) if you are on a Windows machine and with a colon (:) if you are on a Unix machine, so if you want to add the folder `h:\git\bin` to your PATH, you only need to type:

```
> set PATH=%PATH%;h:\git\bin
```

or something like:

```
> set PATH=$PATH;/usr/bin
```

on Unix systems.

Notes for Unix users

- In Unix (Linux, Mac OS X, etc), entries in the path are separated with a colon (:), not a semicolon.
- In Unix (Linux, Mac OS X, etc), environment variables (like PATH) are accessed using an initial dollar symbol instead of using enclosing percents, i.e. to see your path, type: `echo $PATH`

3 Basic actions on GitHub

Open an account on GitHub. Ask the faculty for the right account name to use for "Programming in Java". Once your account is open, perform the following two tasks:

- Create a new repository names `My first repository`.
- Fork the `currency-exchanger` repository from account `bbk-msccs` so that you have your own copy.

[1] But you can try just in case...

4 Basic actions with git

4.1 Working on your own

Clone your own copy of the `currency-exchanger` repository. This will create a local copy of it on your computer.

Add a few lines of code to the program, or modify some of them, to produce a new version. Commit it and write a short message explaining your changes.

Make additional changes and commit them. Write a descriptive message.

Change to a different folder and clone your repository again. Verify that the clone is equal to the original `currency-exchanger`.

Change to the most up-to-date copy and use `git log` to see your latest changes. Use `git diff` to see the differences between the most recent version you cloned and `HEAD`.

Push your changes to your copy of `currency-exhanger`.

Change to a different folder once again and clone your repository again. Verify that this clone contains all your recent changes to `currency-exchanger`.

Make sure you have completed all the former steps before moving on.

4.2 Working with others

Talk to one of your colleagues in the lab and get their account name and the name they have assigned to their fork of `currency-exchanger`. Look at their repository online on GitHub to see their changes.

Pull their repository into your local copy. If the likely case that this results in a conflict, resolve the conflict before committing again; you can solve the conflict in any way you choose: removing their changes, removing your changes, or doing a manual merge of both. Once the conflict is solved, commit.

(If pulling the code did not result in a conflict, pull again from other colleagues until you have a conflict. Solve the conflict and commit.)

Push your latest changes to your copy of the repository. Tell your colleague to pull from it so that you have the same version. Use your web browser to see the changes on GitHub, both yours and your colleague's.