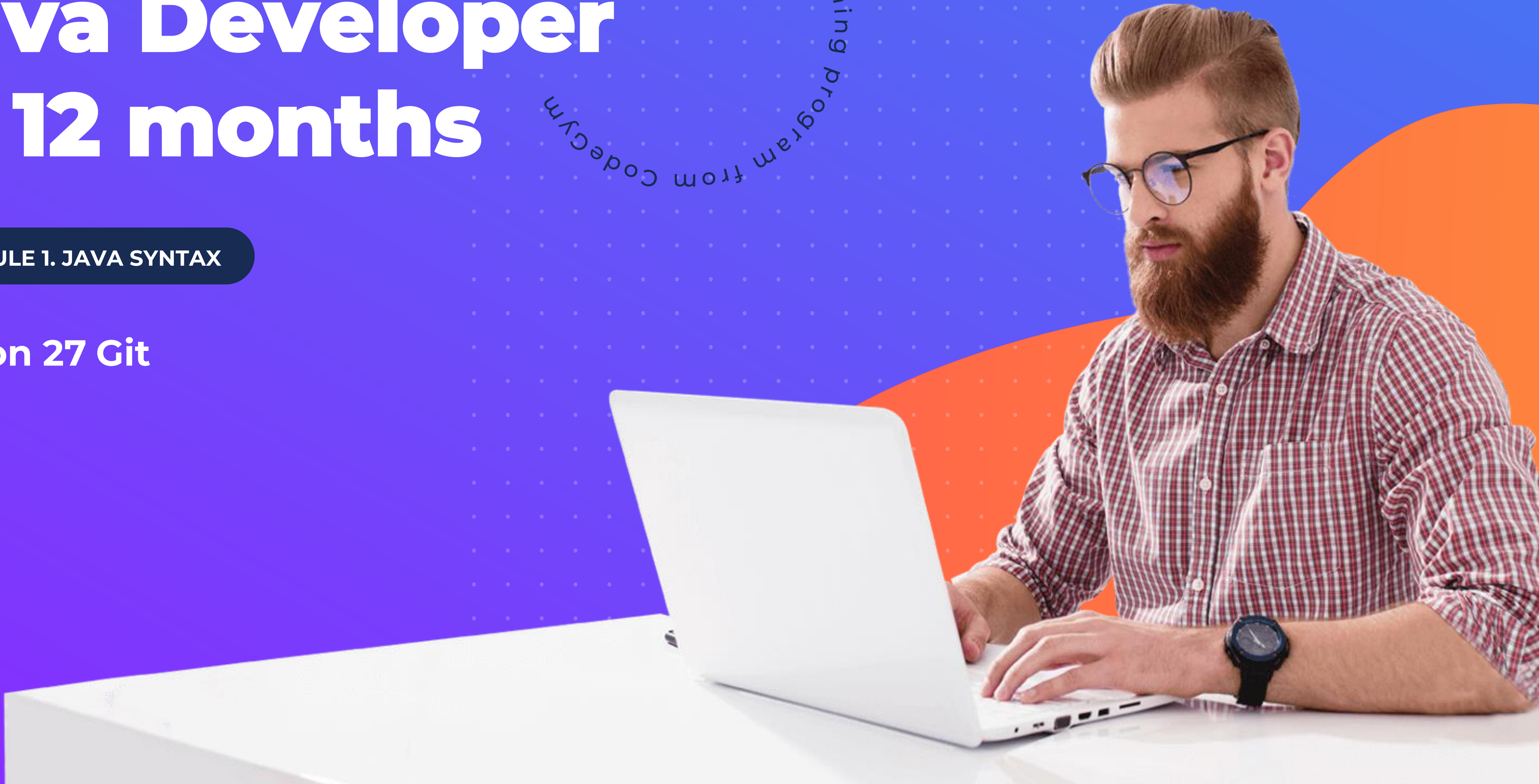
CODEGYM

Java Developer in 12 months

Mentor-supported training program from CodeGym

MODULE 1. JAVA SYNTAX

Lesson 27 Git



Lesson plan

- Installing Git
- Getting started with Git
- How to work with Git in IntelliJ IDEA

Git

Git is a distributed version control system for our code.

Why we need it?

The team needs some kind of work management system. It is needed to track changes that occur over time.

That is, step by step, we see which files have changed and how. This is especially important when you analyze what was done within the framework of one task: this makes it possible to go back.

Installing Git

Installation for Windows
As usual, you need to download the exe file and run it. Everything is simple here: click on the first Google link (https://git-scm.com/downloads), install and that's it.

Installation for Linux
Usually git is already installed and included in Linux distributions, as it is a tool originally written for developing the Linux. But there are situations when it is not. To check this, you need to open a terminal and type: git --version.

For Ubuntu and derivative distributions, you need to write: sudo apt-get install git.

Installation on macOS
Here, too, first you need to check if there is already a git (see above, how on Linux).

If not, the easiest way is to download the latest version: https://sourceforge.net/projects/git-osx-installer/files/.
If XCode is installed, then git will already be automatically installed.

Setting up Git

The git has a user setting from which the work will go. This is a reasonable and necessary thing, because when a commit is created, git takes exactly this information for the Author field.

To set up a username and password for all projects, you need to write the following commands:

```
git config --global user.name ”Ivan Ivanov” git config
--global user.email ivan.ivanov@gmail.com
```

Some terms

To be on topic, it is advisable to add a few new words and actions to your appeal:

- git repository;
- commit;
- branch;
- merge;
- conflicts;
- pull;
- push;
- how to ignore some files (.gitignore).

Git states

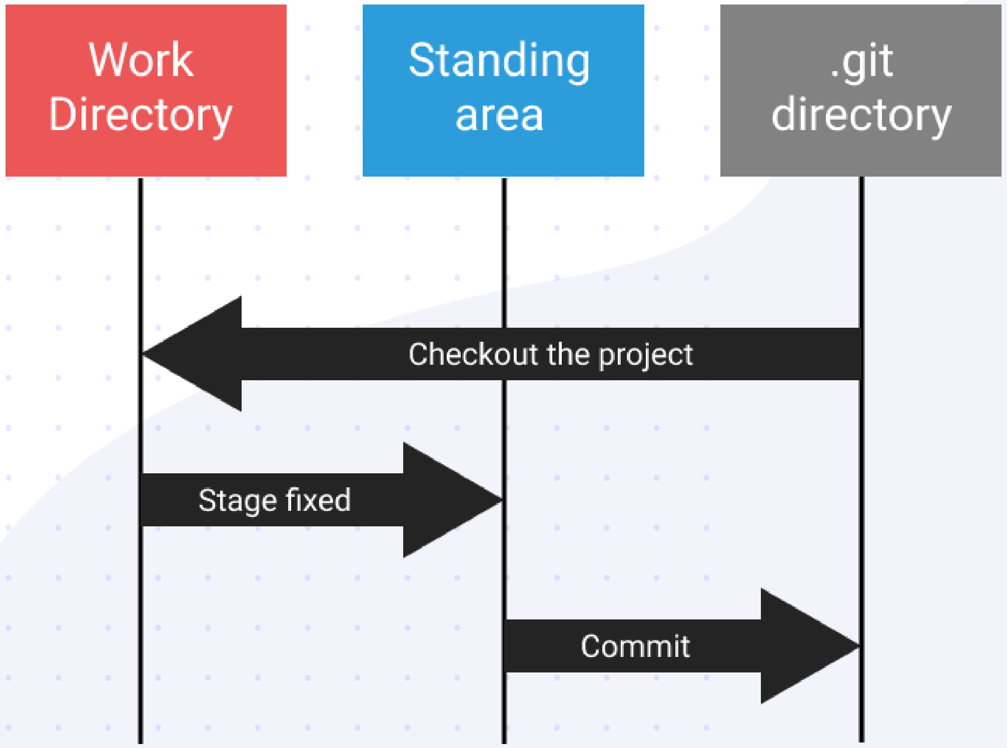
The Git has several states you should understand and remember:

- untracked;
- modified;
- staged;
- committed.

Git states

The states in which the files from our code are:

- A file that is created and not added to the repository will be in the untracked state.
- We make changes to files that have already been added to the git repository - they are in the modified state.
- Of those files that we have changed, we select only those (or all) that we need (for example, we do not need compiled classes), and these classes with changes fall into the staged state.
- A commit is created from the prepared files from the staged state and goes into the git repository. After that, the staged state is empty. But modified can still contain something.



CODEGYM

Commit

A commit is the main object in source control.

It contains all the changes. The commits are linked together as a singly linked list.

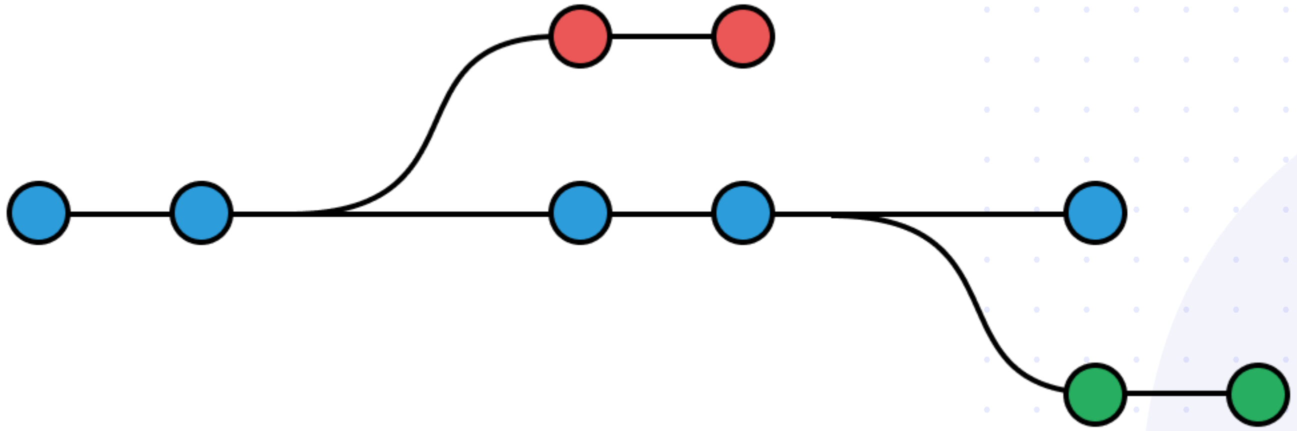
A commit also has its own information (metadata):

- a unique commit ID that can be used to find it;
- the name of the author who created this commit;
- the date when the commit was created;
- a comment that describes what was done during this commit.

Branch

A branch is a pointer to some commit.

The commit knows which commit was before it, when a branch points to a commit, all previous commits are included in that commit.



CODEGYM

Getting started with Git

You can work only with the local repository, and with remote.

To work out the necessary commands, you can use only the local repository. It stores all information only locally in the project in the .git folder.

If we talk about the remote, then all the information is stored somewhere on the remote server: only a copy of the project is stored locally, the changes of which can be pushed (git push) to the remote repository.

Working with git in local repository

To create a local repository, you need to write:

```
git init
```

To check the current status of your work, write this:

```
git status
```

To add files to the "staged" status, you need to write "git add". We have a few options here, for example:

- git add -A — add all files to the "staged" status
- git add . — add all files from this folder and all subfolders. Essentially the same as the previous one;
- git add <filename> — adds a specific file. Here you can use regular expressions to add files according to some pattern. For example, git add *.java: This means that you only want to add files with the java extension.

And finally, the last stage for working with a local repository (there is one more when working with the remote repository ;)) — creating a new commit:

```
git commit -m “all txt files were added to the project”
```

Next up is a great command for looking at the commit history on a branch

```
git log
```

Now that we have files in the modified state, we can examine the changes made to them. This can be done using the following command:

```
git diff
```

.gitignore

If we do not want to store any files in the repository (for example, compiled classes and / or files that development environments was create), then we create a file in the root of the project called .gitignore, and in this file each line will be a template to ignore.

Example for git ignore file:

*.class is ignoring all files with .class extension;

target/ is ignoring the target folder and everything it contains;

*.iml is to ignore all files with .iml extension;

.idea/ is ignoring the .idea folder.

Working with branches

Of course, it is inconvenient to work in one branch alone and impossible when there is more than one person in the team. This is what branching is for.

A branch is just a movable pointer to commits.

To see a list of all branches in the repository and understand which one you are on, you need to write:

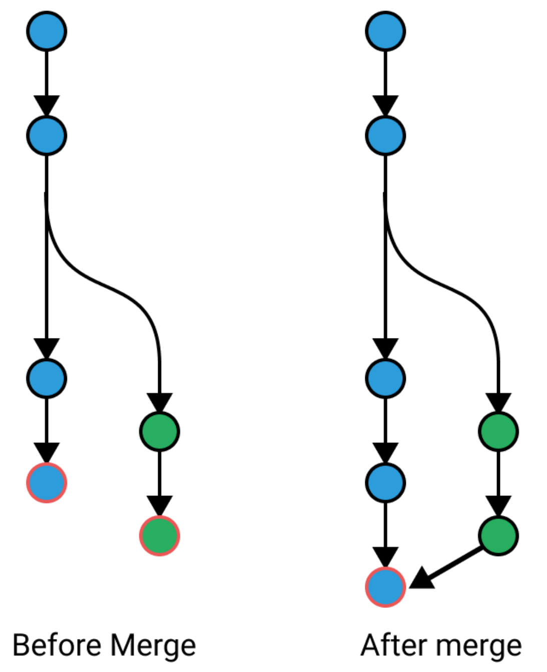
```
git branch –a
```

There are several options for creating branches (the most usable):

- create a new branch based on the one you're on (99% of the time);
- create a branch based on a specific commit (1%).

Resolving conflicts

Here is a picture that shows the process when one branch is merged into another:



So there is master branch. At some point, a secondary one is created from it, in which changes occur. Once the work is done, you need to merge one branch into another.

If the same file is changed in both branches, then when they are merged, a conflict arises that must be resolved before merging.

Working with remote repositories

What are remote repositories?

Examples:

- GitHub is the largest storage for repositories and collaborative development.
- GitLab is an open source web-based DevOps lifecycle tool that provides a code repository management system for Git with its own wiki, bug tracker, CI/CD pipeline, and more.
- BitBucket is a project hosting and collaborative web service based on Mercurial and Git version control. At the same time has a big advantage over GitHub - it has free private repositories. Last year, GitHub also made this feature available to everyone for free.
- And so on...

Working with remote repositories

The first thing to do when working with a remote repository is to clone the project into your local one:

```
git clone URL
```

Now you have a copy of the project locally. To be sure that the latest copy of the project is pulled locally:

```
git pull
```

When we have done something locally and want to commit it to a remote repository, we need to create a new commit locally and then push it to the remote repository:

```
git push
```

CODEGYM

Git in IntelliJ IDEA

Useful hotkeys

ctrl + t - get the latest changes from the remote repository (git pull).

ctrl + k - commit / see all the changes that are currently. This includes both untracked and modified files (git commit).

ctrl + shift + k is the command to push changes to a remote repository. All commits that were created locally and are not yet on the remote will be proposed for push (git push).

alt + ctrl + z - roll back changes in a specific file to the state of the last created commit in the local repository. If you select the entire project in the upper left corner, you can roll back changes to all files.

CODEGYM

Answers to questions

MODULE 1. JAVA SYNTAX