Version control. Getting started with Git.

PRAGMATIC IT Learning & Outsourcing Center

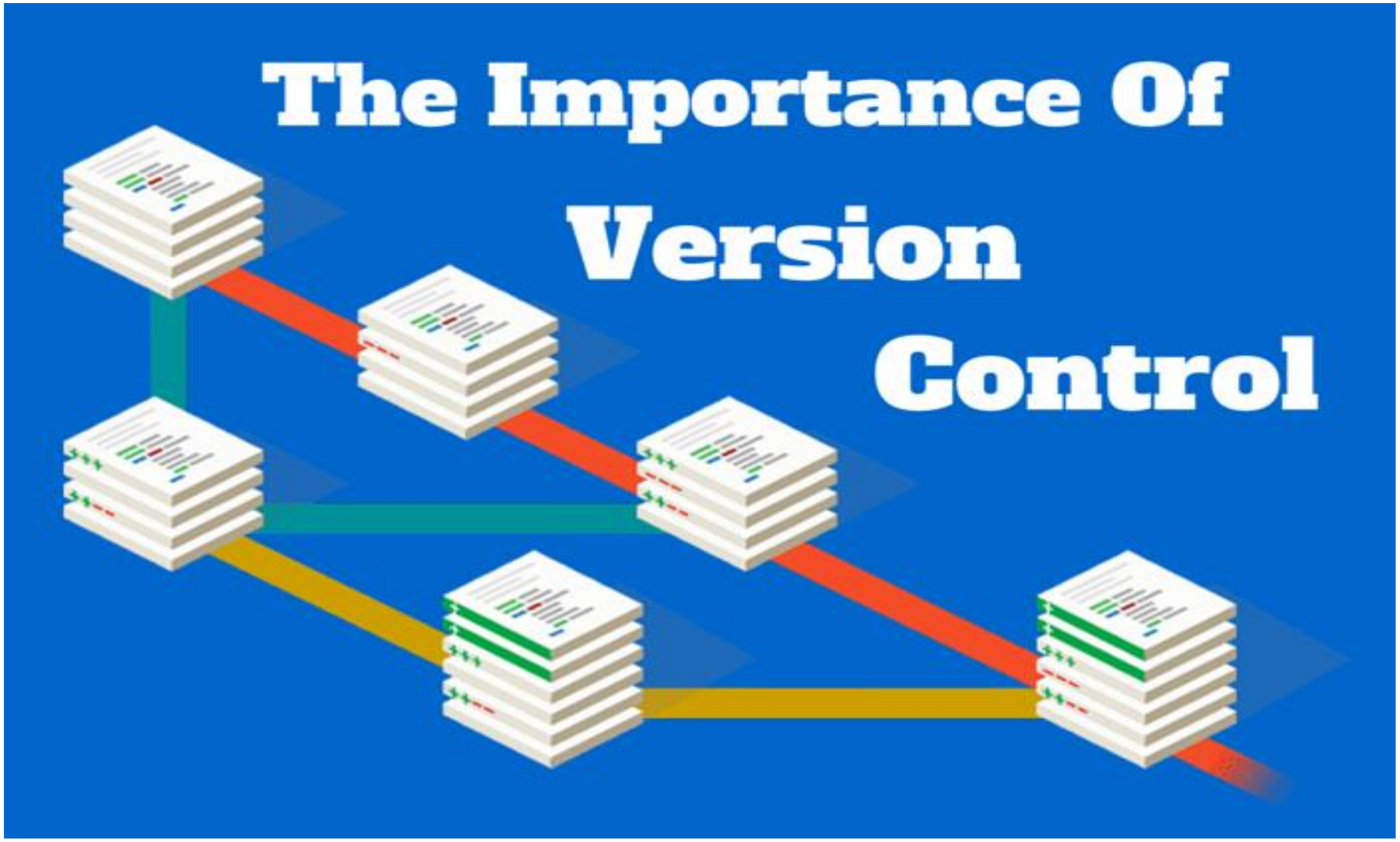
www.pragmatic.bg

Lector: Stefan Vadev
Skype: stefanvadev77
E-mail: stefan.vadev@gmail.com
Facebook:
https://www.facebook.com/stefan.vadev.7

Summary

- What is version control
- How Git works
- Git commands
- Workshop

Version control



What is version control

Version control is a system that records changes to a file or set of files over time so that you can recall specific **versions** later. For the examples in this book, you will use software **source** code as the files being **version controlled**, though in reality you can do this with nearly any type of file on a computer.

Why is version control needed

At the simplest level, developers could simply retain multiple copies of the different versions of the program, and label them appropriately. This simple approach has been used in many large software projects. While this method can work, it is inefficient as many near-identical copies of the program have to be maintained. This requires a lot of self-discipline on the part of developers and often leads to mistakes.

Why is version control needed

Moreover, in software development, legal and business practice and other environments, it has become increasingly common for a single document or snippet of code to be edited by a team, the members of which may be geographically dispersed and may pursue different and even contrary interests. Sophisticated revision control that tracks and accounts for ownership of changes to documents and code may be extremely helpful or even indispensable in such situations.

Intro to Git

What is Git

First of all, GitHub is not git. Many people understandably confuse the two. GitHub is a website for hosting projects that use git.

Git is a type of version control system (VCS) that makes it easier to track changes to files. For example, when you edit a file, git can help you determine exactly what changed, who changed it, and why.

What is Git

It’s useful for coordinating work among multiple people on a project, and for tracking progress over time by saving “checkpoints”. You could use it while writing an essay, or to track changes to artwork and design files.

Git isn’t the only version control system out there, but it’s by far the most popular. Many software developers use git daily, and understanding how to use it can give a major boost to your resume.

What is Git

In complex projects, where multiple people might be making changes to the same files simultaneously, it’s easy to get into a weird state. Anyone who’s dealt with “merge conflicts” and those baffling >>>>>>> ======= <<<<<<< symbols can attest to this.
If you start to understand how git works, you’ll see why conflicts occur and how to recover from these situations easily.

Let’s install Git

https://git-scm.com/downloads

Git commands

Start your own repository from scratch (in any existing folder on your computer):

git init

This will create a hidden .git folder inside your current folder — this is the "repository" (or repo) where git stores all of its internal tracking data. Any changes you make to any files within the original folder will now be possible to track.

The original folder is now referred to as your working directory, as opposed to the repository (.git folder) that tracks your changes. You work in the working directory. Simple!

Git clone

Clone an existing repo:

git clone https://github.com/cooperka/emoji-commit-messages.git

This will download the .git repository to your computer (that folder mentioned above) and extract the current snapshot of the repo (all the files) to your working directory. By default it will all be saved in a folder with the same name as the repo (in this case emoji-commit-messages).

Git status

The URL you specify here is called the remote origin (it’s where the files are originally downloaded from). View the current status of your project:

git status

This will print some basic information such as what branch you’re on and what files have been changed in your working directory since the last commit.

Git branch

Create a new branch name:

git branch <new-branch-name>

You can think of this like creating a local “checkpoint” (technically called a reference) and giving it a name. The new branch that gets created is simply a reference point to the current state of your repo. It’s similar to doing File > Save as… in a text editor. The branch name can be used in various other commands as you’ll soon see.

Git commit

As you make changes to your files, you can save each checkpoint as you go along in the form of commits (see git commit below). Technically these particular checkpoints are called revisions, and they can be used in various other commands. The name will be a random-looking hash of numbers and letters such as e093542.

✨ **That’s really the core function of git: To save checkpoints (revisions) and share them with other people. Everything revolves around this concept.**

Git checkout

Check out a particular branch:
git checkout <existing-branch-name>
You can think of this like “resuming” from an existing checkpoint. All your files will be reset to whatever state they were in on that particular branch.

⚠️ **Keep in mind that any changes in your working directory will be kept around. See git stash for a simple way to avoid headaches.**

😎 You can use the -b flag as a shortcut to create a new branch and then check it out all in one step. This is quite common:
git checkout -b <new-branch-name>

Git diff

View the differences between checkpoints:

git diff <branch-name> <other-branch-name>

After editing some files, you can simply type git diff to view a list of the changes you've made. This is a good way to double-check your work before committing it.

For each group of changes, you'll see what the file used to look like (prefixed with - and colored red), followed by what it looks like now (prefixed with + and colored green).

Git add

Stage your changes to prepare for committing them:

git add <files>

After editing some files, this command will mark any changes you’ve made as “staged” (or “ready to be committed”).

⚠️ If you then go and make more changes, those new changes will not automatically be staged, even if you’ve changed the same files as before. This is useful for controlling exactly what you commit, but also a major source of confusion for newcomers.

Git commit

Commit your staged changes:

git commit

This will open your default command-line text editor and ask you to type in a commit message. As soon as you save and quit, your commit will be saved locally.

The commit message is important to help other people understand what was changed and why you changed it. There's a brief guide here explaining how to write useful commit messages.

😎 You can use the -m flag as a shortcut to write a message. For example:

git commit -m "Add a new feature"

Git push

Push your branch to upload it somewhere else:
git push origin <branch-name>
This will upload your branch to the remote named origin (remember, that's the URL defined initially during clone).

After a successful push, your teammates will then be able to pull your branch to view your commits (see git pull below).

😎 As a shortcut, you can type the word HEAD instead of branch-name to automatically use the branch you're currently on. HEAD always refers to your latest checkpoint, that is, the latest commit on your current branch.

Git fetch

Fetch the latest info about a repo:

git fetch

This will download the latest info about the repo from origin (such as all the different branches stored on GitHub).

It doesn’t change any of your local files — just updates the tracking data stored in the .git folder.

Git merge

Merge in changes from somebody else:

git merge <other-branch-name>

This will take all commits that exist on the other-branch-name branch and integrate them into your own current branch.

⚠️ This uses whatever branch data is stored locally, so make sure you've run git fetch first to download the latest info.

Git pull

😎 As a shortcut, you can use the pull command to both fetch and merge all in one step. This is more common than merging manually like above:

```
git pull origin master
```

Git Workshop

Now let’s continue here:

https://try.github.io/levels/1/challenges/1

https://www.atlassian.com/git/tutorials/what-is-git

https://www.tutorialspoint.com/git/index.htm

Summary

- What is version control
- How Git works
- Git commands
- Workshop