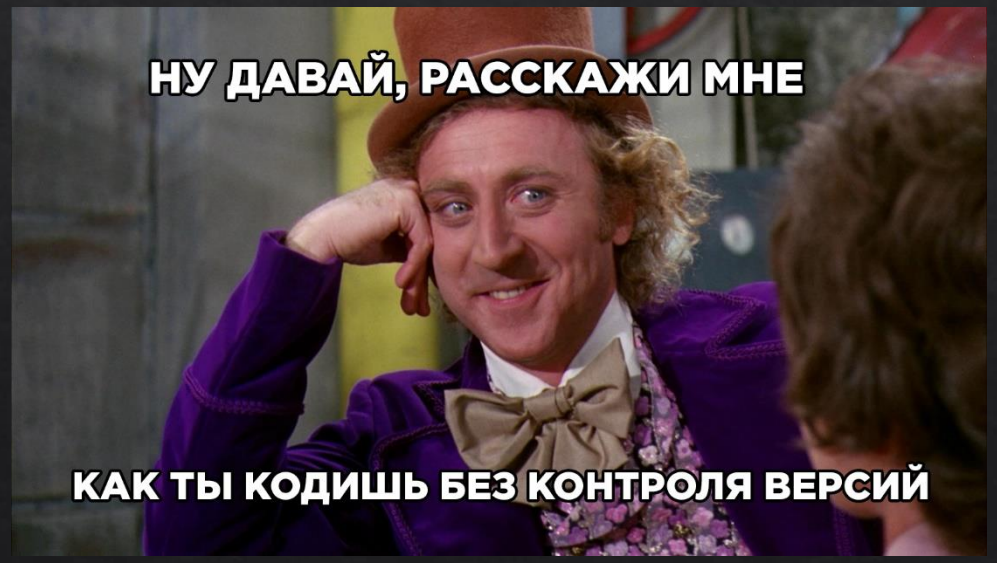
Система контроля версий

Система контроля версий

Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.



Установка Git

```git
# linux
sudo apt install git

# mac
brew install git

# windows
# ручками с оффициального сайта -_-
# https://git-scm.com/download/win
```

```git
git version
# git version 2.34.1
```

Конфигурация

git

```
# get your local email
git config user.email

# set email only for this repo
git config user.email "your.mail@mail.com"

# or set global email
git config --global user.email "your.mail@mail.com"

# some configs
git config user.name "okumuramura"
git config --global color.ui true
```

Создание репозитория

Для создания нового локального репозитория вам нужно:

- Перейти в папку с вашим проектом
- Выполнить команду `git init`

После этого в папке проекта появится папка с именем .git

```git
cd /<your-project-path>
git init
```

Копирование репозитория

- Для клонирования репозиториев применяется команда `git clone`

```git
# clone local repositoty
git clone /<repo-path>

# clone remote repository
git clone <user>@<host>/<path>
```

- В качестве примера клонируем репозиторий с примерами для этого курса:

```git
git clone github.com/okumuramura/python-lessons livecoding
```

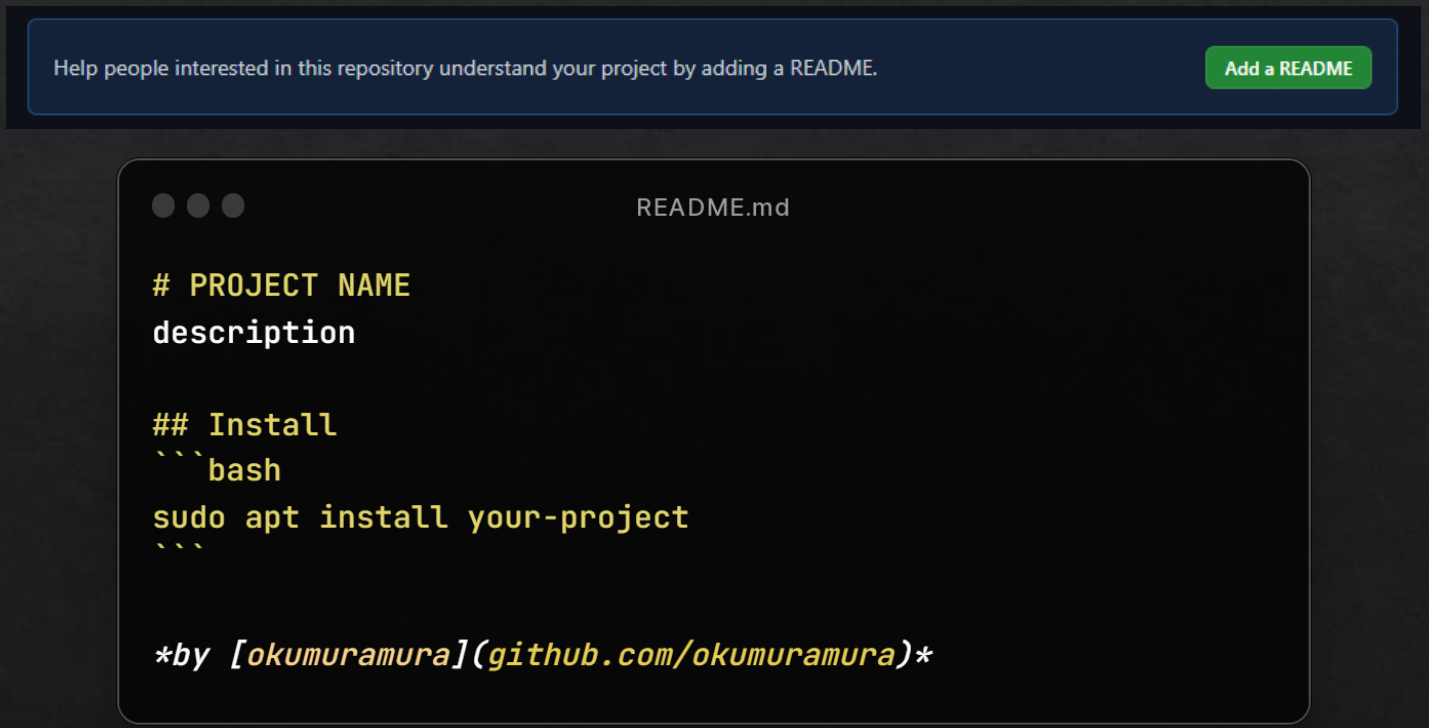
Состояние репозитория

git

```
# check changes
git status

# show commits history
git log

# show difference between current state and HEAD
git diff

# show current breanch
git branch
```

README

- Файл README.md не является обязательной частью ваших проектов или репозитория git, но настоятельно рекомендуется его использовать для наиболее полного представления проекта перед пользователями: его назначения, функционала, примеров использования.



.gitignore

- При разработке в директории проекта могут появляться файлы, которые не нужно помещать в репозиторий (файлы базы данных, настройки редактора, сгенерированные файлы). Шаблоны для игнорирования таких файлов задаются в файле .gitignore. Обычно файл .gitignore размещается в корневой директории проекта, но возможно описание и нескольких подобных файлов для разных директорий.

.gitignore

```
# Local data
info.py
*.db
config.toml

# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# ide
.vscode
.idea
```

Шаблоны игнорирования

Используя шаблоны игнорирования мы можем указать как конкретные файлы, так и более обобщённые правила для игнорирования файлов.

Помимо игнорируемых файлов вы можете указать и исключения к игнорированию. При этом исключение будет применено ко всем правилам, описанным выше него, однако может быть перекрыто следующими.

.gitignore

```
# ignore specific file
/cache/temp.log

# ignore all *.log files
*.log
# will ginore: debug.log .log /logs/gui.log etc.

# do not ignore dev.log
!dev.log

# with template
logs/*_03_22/*
# will ignore all files in:
#   logs/01_03_22/
#   logs/12_03_22/
#   logs/_03_22/
```

Обновление репозитория

Для добавления нового «коммита» в ваш репозиторий необходимо выполнить ряд действий:

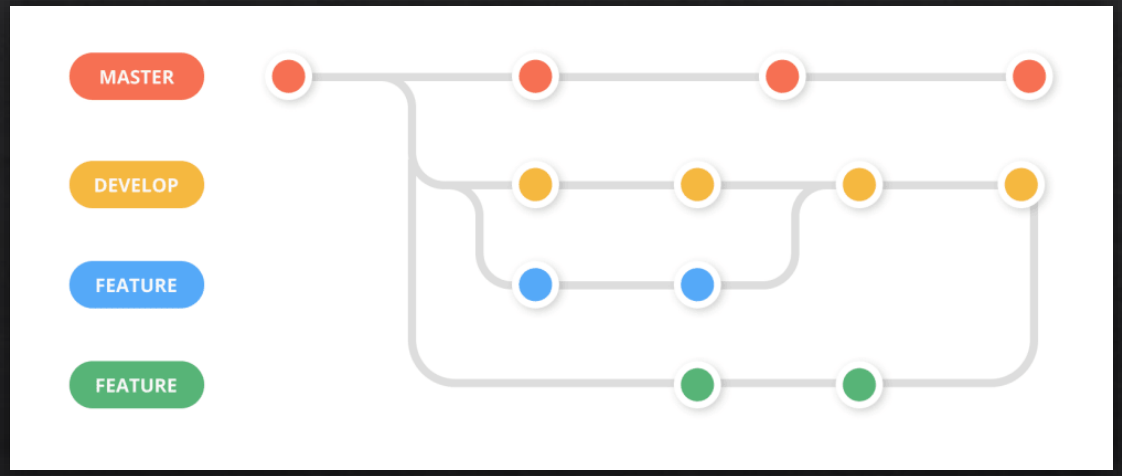
- При помощи команды `add` добавим файлы, изменения в которых должны быть зафиксированы в следующем «коммите»
- Создать «коммит» используя команду `commit` и указав сопровождающее сообщение

```git
# lock changes in README.md
git add README.md
# lock changes in all files
git add .

# check commit status
git status

# make commit
git commit -m "added README file"
```

Ветки

При разработке часто возникает необходимость разработать новый функционал или опробовать «сырую» версию не испортив «основную».

В Git для этого используются ветки. Ветки позволяют вести разработку сразу нескольких вариантов, отменять и объединять некоторые решения, поддерживать ваш проект не меняя стабильную версию.



Базовая работа с ветками

git

```
# check current branch
git branch
# * master
#   dev-branch

# create and switch to new branch
git checkout -b new-branch

# just switch to branch
git checkout master

# show difference between master and new-branch
git diff master new-branch

# try merge new-branch to master
git merge new-branch

# delete new-branch
git branch -d new-branch
```

Удалённые репозитории

Git поддерживает работу с сервисами управления репозиториями и хранения кода. Наиболее распространёнными решениями являются сервисы GitHub и GitLab.

git

```
# show current remote urls
git remote -v

# add new remote repo
git remote add <url> new

# update url for remote repo "new"
git remote set-url new <new-url>

# get updates from remote repo
git fetch

# or get updates and try merge
git pull

# and filnaly puhs local commits to remote repo "new"
git push -u new master
```