ФАЙЛ .GITIGNORE

Содержит имена файлов и/или шаблоны имён, игнорируемых при наблюдении и создании коммита.

Создавать в корне папки с репозиторием.

Пример:

```
build/
temp.cpp
*.so
.idea/
```

ЛЕНЬ ДЕЛАТЬ ЭТОТ ФАЙЛ ВРУЧНУЮ?

https://www.gitignore.io/

ФАЙЛ .GITIGNORE

Если файл добавлен в репозиторий и его название есть в .gitignore, то этот файл игнорироваться не будет

`git rm имя_файла` — удаление файла из репозитория и из рабочей директории

`git rm --cached имя_файла` — удаление файла из репозитория, но в рабочей директории он остается

ХРАНЕНИЕ ПАПОК

Системы контроля версий хранят в репозитории только содержание файлов (а именно строки). Пустые папки храниться в репозитории не будут.

Ситуации, когда нужны пустые папки в репозитории:

- на начальном этапе разработки, когда формируется файловая структура проекта (но сами программы еще не написаны)
- когда папка нужна приложению, но её содержание не должно сохраняться в репозитории (папка с логами работы приложения)

ХРАНЕНИЕ ПАПОК

При формировании файловой структура проекта, но сами программы внутри папок еще не написаны:

- В пустую папку помещается пустой файл
- Пустой файл сохраняется в репозитории
- В git-сообществе этот пустой файл часто называют .gitkeep

ХРАНЕНИЕ ПАПОК

В репозитории должна храниться пустая папка, но любое её содержимое должно игнорироваться git-ом

1 способ

- В пустую папку помещается файл .gitignore
- Содержание вложенного .gitignore:

```
*
!.gitignore
```

ХРАНЕНИЕ ПАПОК

В репозитории должна храниться пустая папка, но любое её содержимое должно игнорироваться git-ом

2 способ

- В пустую папку помещается файл .gitkeep
- Содержание корневого .gitignore (порядок строк имеет значение):

```
имя_пустой_папки/*
!.gitkeep
```

GIT ALIASES

Aliases (псевдонимы) — это короткие команды git, которые являются аналогами полных команд

`git st == git status`

Алиасы прописываются в файле конфигурации git (`~/.gitconfig`):

```
[alias]
  st = status
  co = checkout
  ci = commit
  br = branch
  adog = log --all --decorate --oneline --graph
  hist = log --pretty=format:\"%h %ad | %s%d [%an]\"
                                    --graph --date=short
```

Одна строка

GIT ALIASES

Альтернативный способ добавления: выполнить в консоли

```
git config --global alias.st status
git config --global alias.co checkout
git config --global alias.ci commit
git config --global alias.br branch
git config --global alias.adog
                    "log --all --decorate --oneline -graph"
git config --global alias.hist "log --graph --date=short
                  --pretty=format:\'%h %ad | %s%d [%an]\'"
```

ОРГАНИЗАЦИЯ ВЕТОК В РЕПОЗИТОРИИ

- Дан программный продукт
- Идёт активная разработка
- Каждый участник работает в своей ветке
- Как назвать ветки?
- И как не запутаться в названиях?

ВАРИАНТ 1: GIT FLOW

- Основная ветка, из которой берутся готовые версии продукта: master
- Основная ветка, в которую стекается новый функционал: develop
- Разработка нового функционала: feature/taskname
- Исправление ошибок: hotfix/errorname

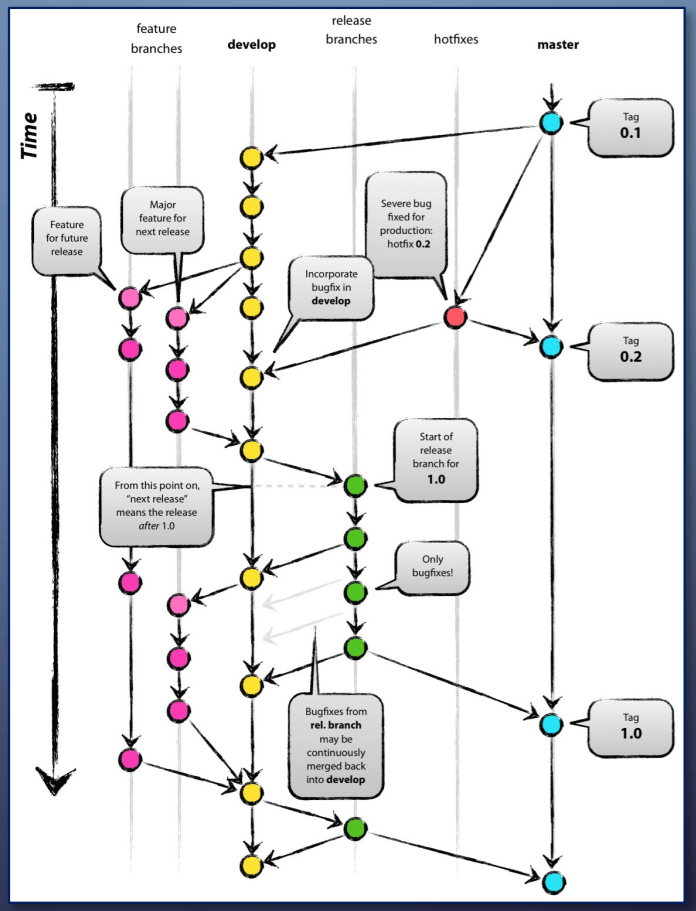
ВАРИАНТ 1: GIT FLOW

- Разработчик добавляет новый функционал в ветку feature/taskname
- Когда функционал готов, ветка объединяется с develop
- Когда в develop накапливается много функционала, готового к работе, ветка объединяется с master
- Если в проекте обнаруживается ошибка, то
  - от ветки master создаётся новая ветка hotfix/errorname
  - В неё вносятся исправления
  - Ветка вливается в master и в develop

ВАРИАНТ 1: GIT FLOW

Подробнее (теги, release-ветки и т.д.):

https://danielkummer.github.io/git-flow-cheatsheet/index.ru_RU.html



ВАРИАНТ 2: GITHUB FLOW

- Основная ветка: main
- Разработка нового функционала: feature/taskname
- Исправление ошибок: hotfix/errorname

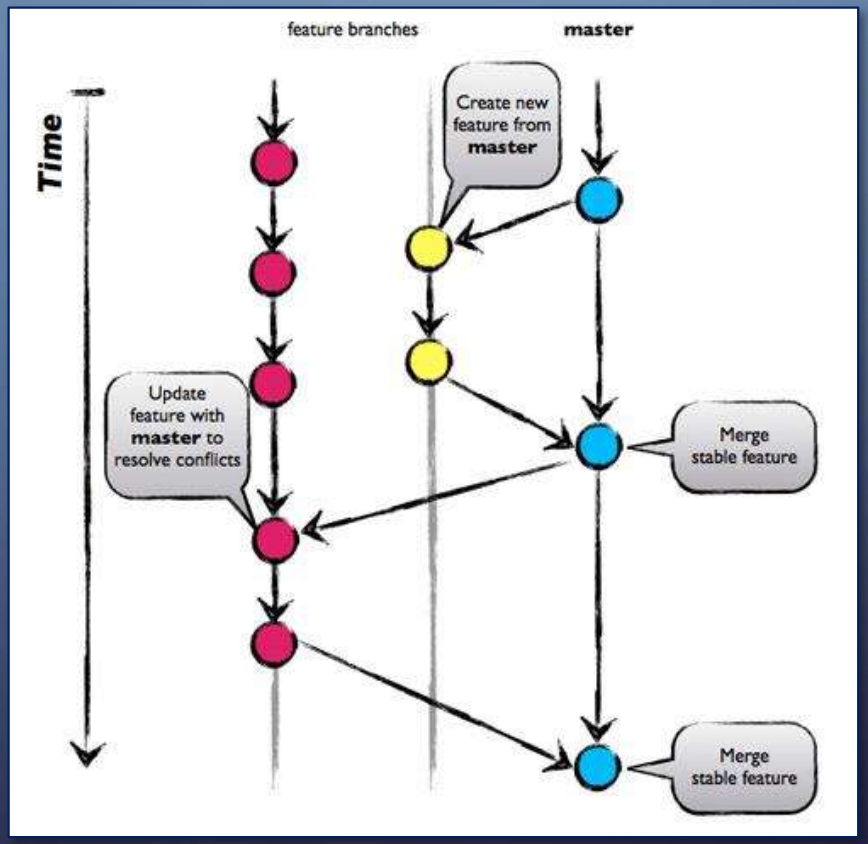
ВАРИАНТ 2: GITHUB FLOW

- Разработчик добавляет новый функционал в ветку feature/taskname
- Когда функционал готов, ветка объединяется с main
- Если в проекте обнаруживается ошибка, то
  - от ветки master создаётся новая ветка hotfix/errorname.
  - В неё вносятся исправления
  - Ветка вливается в main

ВАРИАНТ 2: GITHUB FLOW

Подробности:

- https://guides.github.com/introduction/flow/
- https://habr.com/post/346066/



ВАРИАНТ 3: GITLAB FLOW

- Основная ветка, из которой берутся готовые версии продукта: production
- Основная ветка, в которую стекается новый функционал: main
- Разработка нового функционала: feature/taskname
- Исправление ошибок: hotfix/errorname

ВАРИАНТ 3: GITLAB FLOW

- Разработчик добавляет новый функционал в ветку feature/taskname
- Когда функционал готов, ветка объединяется с main
- Если в проекте обнаруживается ошибка, то
  - от ветки master создаётся новая ветка hotfix/errorname.
  - В неё вносятся исправления
  - Ветка вливается в main и в production

ВАРИАНТ 3: GITLAB FLOW

Подробности:

- https://about.gitlab.com/topics/version-control/what-is-gitlab-flow/
- https://habr.com/company/softmart/blog/316686/