Git

- GIT Top Commands
- Merge/Rebase Examples
- GIT HandBook

GIT Top Commands

???????? STATUS

Инициализация каталога как Git-репозитория

```
git init
```

Команда показывающая какие файлы в каком состоянии находятся

```
git status
```

Показать историю коммитов в текущей ветке / всех ветках

```
git log
git log --all
```

???????? BRANCH ?????????

Отобразить активные ветки / все активные и доступные ветки

```
git branch
git branch -a
```

Переключиться на другую ветку

```
git checkout <branch-name>
```

Создать новую ветку и переключиться на неё

```
git checkout -b <branch-name>
```

Проиндексировать все файлы в папке (для commit)

```
git add .
```

Удалить ветку из remote REPO

```
git push origin --delete develop
```

Отправка всех веток локального репозитория на удалённый репозиторий

```
git push <link.repo> --all
git push --force <branch-name>
```

???????? ? COMMITs

Создать коммит с указанием имени

```
git commit -m "new commit"
```

Автоматическая индексация всех уже отслеживаемых файлов

```
git commit -a
git commit -am "new commit" (автоиндекс+имя)
```

Редактор названия коммитов по-умолчанию

```
git config --global core.editor "vim"
```

Изменить сообщение последнего коммита

```
git commit --amend
```

Забыть все последующие коммиты от указанного по <hash>

```
git reset --hard <hash>
```

Изменить дату текущего коммита

```
GIT_COMMITTER_DATE="Thu Jun 08 13:44:15 2023 +0300" git commit --amend
```

???????? ? TAGs

Создать новый Tag

```
git tag -a v1.0 -m "feature01"
git tag --list
```

Отправить Tag на удалённый репозиторий

```
git push origin v1.0
```

???????? ? ALIASes

Алиасы для графического формата команды git log:

Подключить алиас с ключом --all (отображение всех ветвей)

```
git config --global alias.log0 "log \
  --graph \
  --abbrev-commit \
  --decorate \
  --format=format:'\
%C(bold blue)%h%C(reset) - \
%C(bold green)(%ci)%C(reset) \
%C(white)%s%C(reset) \
%C(dim white)- %an%C(reset)\
%C(auto)%d%C(reset)' \
  --all
```

Подключить алиас с ключом --branches (отображение локальных веток)

```
git config --global alias.log1 "log \
  --graph \
  --abbrev-commit \
  --decorate \
  --format=format:'\
%C(bold blue)%h%C(reset) - \
%C(bold green)(%ci)%C(reset) \
%C(white)%s%C(reset) \
%C(dim white)- %an%C(reset)\
%C(auto)%d%C(reset)' \
  --branches"
```

Отключить алиас

```
git config --global --unset alias.log1
```

???????? ? REPOs

Клонировать удалённый репозиторий (папка будет создана)

```
git clone <link.repo>
```

Клонировать удалённый репозиторий по branch+tag

```
git clone <link.repo> --branch '<tag-name>'
```

Скачать изменения без слияния (fetch)
Изменения сохраняются в FETCH_HEAD

```
git fetch origin develop
```

Посмотреть HASH скачанного коммита

```
git rev-parse FETCH_HEAD         # полный hash
git log FETCH_HEAD -1 --oneline  # краткий hash
git diff HEAD FETCH_HEAD         # посмотреть изменения
```

Слить изменения

Способ 1: Через временную ветку

```
git branch temp-branch FETCH_HEAD # создать ветку
git merge temp-branch             # смержить
git branch -d temp-branch         # удалить ветку
```

Способ 2: Через HASH

```
git merge FETCH_HEAD     # смержить напрямую
git merge <hash>         # смержить по hash
git pull origin develop  # fetch + merge одной командой
```

??????? ???????? ? REPOs

Пример с Новым Repo

```
# Инициализация и Push-инг Commit-а в Новый Repo
git init
git config --global user.name "Yuriy"
git config --global user.email "xeonmp22@gmail.com"
git init --initial-branch=main
#git remote set-url origin
git@gitlab.rebrainme.com:devops_users_repos/725/devops/07_cicd.dev.git
```

```
#git remote remove origin
git remote add origin git@gitlab.rebrainme.com:devops_users_repos/725/devops/07_cicd.dev.git
git add .
git commit -am "34: Logging 6/prom.grafana"
# При 1-ом Commit-е делаем --set-upstream
git push -u origin main
# Для последующих Commit-ов
git push --all
```

Пример с Рабочим Repo

```
# Очистка рабочего Repo от файлов Старых Commit-ов
cd $HOME/rebrainme/devops/07_cicd
git rm -rf roles/
cd ../07_cicd.dev
rm -rf .git

# Копирование новых файлов в рабочий Repo
cp -r .gitignore main.tf playbook.yml README.md roles/ ../07_cicd
cd ../07_cicd
git add .
git commit -am "34: Logging 6/prom.grafana"
git push --all
```

Merge/Rebase Examples

???????? ???????

Схема коммитов до объединения веток

```
* efa552b - (2026-06-25 13:32:25 +0300) v6: develop - shion (develop)
* c828a46 - (2026-06-25 13:32:25 +0300) v5: develop - shion
* e17a9c1 - (2026-06-25 13:32:25 +0300) v4: develop - shion
| * 10834a6 - (2026-06-25 13:32:08 +0300) v6: main - shion (HEAD -> main)
| * ecb682a - (2026-06-25 13:32:08 +0300) v5: main - shion
| * 3725452 - (2026-06-25 13:32:08 +0300) v4: main - shion
|/
* 6dca252 - (2026-06-25 13:32:08 +0300) v3: main - shion
* 6630ea4 - (2026-06-25 13:32:08 +0300) v2: main - shion
* 6b1466c - (2026-06-25 13:32:08 +0300) v1: main - shion
```

code.txt [Main ver.]

```
01_main
02_main
03_main
04_main
05_main
06_main
```

code.txt [Develop ver.]

```
01_main
02_main
03_main
04_develop
05_develop
06_develop
```

Git Merge

??????? 1. ?????????????? Main

```
git checkout main
# Включаем режим union-слияния для файла code.txt
echo "code.txt merge=union" > .gitattributes
# Слияние ветки develop с приоритетом изменений main
git merge -Xours develop -m "Merge branch 'develop'"
rm .gitattributes
```

> Ветка develop вливается в ветку main. Файлы объединяются построчно
> В конфликтных местах (строки 04-06) побеждает версия из main
> Не конфликтующие строки (07-09 из develop) добавляются в результирующий файл

```
*   6a3d02b - (2026-06-25 13:34:55 +0300) Merge branch 'develop' - shion (HEAD -> main)
|\
| * efa552b - (2026-06-25 13:32:25 +0300) v6: develop - shion (develop)
| * c828a46 - (2026-06-25 13:32:25 +0300) v5: develop - shion
| * e17a9c1 - (2026-06-25 13:32:25 +0300) v4: develop - shion
* | 10834a6 - (2026-06-25 13:32:08 +0300) v6: main - shion
* | ecb682a - (2026-06-25 13:32:08 +0300) v5: main - shion
* | 3725452 - (2026-06-25 13:32:08 +0300) v4: main - shion
|/
* 6dca252 - (2026-06-25 13:32:08 +0300) v3: main - shion
* 6630ea4 - (2026-06-25 13:32:08 +0300) v2: main - shion
* 6b1466c - (2026-06-25 13:32:08 +0300) v1: main - shion
```

code.txt [Merge]

01_main
02_main
03_main
04_main
05_main
06_main
04_develop
05_develop
06_develop

??????? 2. ????????????? Develop

```
git checkout develop
# Включаем режим union-слияния для объединения строк файла code.txt
echo "code.txt merge=union" > .gitattributes
# Слияние ветки main с приоритетом изменений develop
git merge -Xtheirs main -m "Merge branch 'main' into develop"
rm .gitattributes
```

> Ветка main вливается в ветку develop. Файлы объединяются построчно
> В конфликтных местах (строки 04-06) побеждает версия из develop (-Xours)
> Не конфликтующие строки (04-06 из main) добавляются в результирующий файл

```
*   47739bd - (2026-06-25 13:39:07 +0300) Merge branch 'main' into develop - shion (HEAD ->
develop)
|\
| * 10834a6 - (2026-06-25 13:32:08 +0300) v6: main - shion (main)
| * ecb682a - (2026-06-25 13:32:08 +0300) v5: main - shion
| * 3725452 - (2026-06-25 13:32:08 +0300) v4: main - shion
* | efa552b - (2026-06-25 13:32:25 +0300) v6: develop - shion
* | c828a46 - (2026-06-25 13:32:25 +0300) v5: develop - shion
* | e17a9c1 - (2026-06-25 13:32:25 +0300) v4: develop - shion
|/
* 6dca252 - (2026-06-25 13:32:08 +0300) v3: main - shion
* 6630ea4 - (2026-06-25 13:32:08 +0300) v2: main - shion
* 6b1466c - (2026-06-25 13:32:08 +0300) v1: main - shion
```

code.txt [Dev ver.]

01_main
02_main
03_main
04_develop
05_develop
06_develop

Git Rebase

??????? 1. ?????????? Main

```
git checkout develop
git rebase main
```

> **Суть:** Коммиты из ветки develop переносятся в конец ветки main и пересоздаются с новыми hash
> **Файлы:** При перебазировании каждый коммит из develop по очереди применяется поверх текущего состояния main
> **Хронология:** Ветка develop перестраивается поверх main, её коммиты получают новые hash
> Конфликты разрешаются в пользу ветки develop

> Rebase main (+develop) — подобен процессу, когда вы синхронизируете свою инфраструктуру (main)
> с последними изменениями с тестового стенда (develop), перестраивая историю develop поверх main,
> но сохраняя все свои экспериментальные наработки (develop) в конфликтных местах
> Вы поднимаете main на новый уровень, содержащий все изменения develop
> и история main становится линейным продолжением истории develop

```
* f7f88b7 - (2026-06-25 15:53:56 +0300) v6: develop - shion (HEAD -> develop)
* d5f3c38 - (2026-06-25 15:53:56 +0300) v5: develop - shion
* ce362ac - (2026-06-25 15:53:56 +0300) v4: develop - shion
* 10834a6 - (2026-06-25 13:32:08 +0300) v6: main - shion (main)
* ecb682a - (2026-06-25 13:32:08 +0300) v5: main - shion
* 3725452 - (2026-06-25 13:32:08 +0300) v4: main - shion
* 6dca252 - (2026-06-25 13:32:08 +0300) v3: main - shion
* 6630ea4 - (2026-06-25 13:32:08 +0300) v2: main - shion
* 6b1466c - (2026-06-25 13:32:08 +0300) v1: main - shion
```

code.txt [Rebase]

01_main
02_main
03_main
04_develop
05_develop
06_develop

??????? 2. ?????????? Develop

```
git checkout main
git rebase develop
```

> **Суть:** Commits из ветки main переносятся в конец ветки develop и пересоздаются с новыми hash
> **Файлы:** При перебазировании каждый commit из main по очереди применяется поверх текущего состояния develop
> **Хронология:** Ветка main перестраивается поверх develop, её коммиты получают новые hash
> Конфликты разрешаются в пользу ветки main

> Rebase develop (+main) - подобен процессу, когда вы обновляете свой тестовый стенд(develop),
> пересобирая его на основе последней версии инфраструктуры (main), но с сохранением всех кастомных доработок.
> Теперь develop базируется на свежей стабильной версии, а его история является логичным продолжением истории main.

```
* 94f8376 - (2026-06-25 16:12:33 +0300) v6: main - shion (HEAD -> main)
* daea4c3 - (2026-06-25 16:12:33 +0300) v5: main - shion
* 98ceae4 - (2026-06-25 16:12:33 +0300) v4: main - shion
* efa552b - (2026-06-25 13:32:25 +0300) v6: develop - shion (develop)
* c828a46 - (2026-06-25 13:32:25 +0300) v5: develop - shion
* e17a9c1 - (2026-06-25 13:32:25 +0300) v4: develop - shion
* 6dca252 - (2026-06-25 13:32:08 +0300) v3: main - shion
* 6630ea4 - (2026-06-25 13:32:08 +0300) v2: main - shion
* 6b1466c - (2026-06-25 13:32:08 +0300) v1: main - shion
```

code.txt [Rebase]

01_main
02_main
03_main
04_main
05_main
06_main

??????? REBASE

>

Если в develop больше строк, (например 07-09) и они не конфликтуют с main, то они сохраняются и после rebase
Конфликт возникает только там, где строки пересекаются (04-06)

ЗАПРЕЩЕНО производить git rebase с ветками которые вы уже отправили в общий репозиторий !

GIT HandBook

> Источник **git-scm.com**

- **Git Cheat Sheet**
- Подключение к аккаунту Github используя SSH-ключ
- Первоначальная настройка Git
- **Команды:**
- git **add**
- git **commit**
- git **ignore**
- git **rm**
- git **log**
- git **tag**
- git **merge/rebase**
- git **cherry-pick**
- Основы ветвления и слияния в Git
- Изменение истории: **rebase -i** (squash)
- Создание pull requests в сторонний репозиторий

> **Другие Источники**

- Интерактивное демо по ветвлению [+vpn]
- Ручное разрешение конфликтов слияния
- Методология ветвления git-flow

git-branch1.jpg