Начало работы:

1) Создание пользователя и электронной почты:

git config --global user.name "NewNameUser"

git config --global user.email "Newuser@company.com"

2) Клонирование репозитория:

git clone [gitosis@git.yourserver.com:yourproject.git](mailto:gitosis@git.yourserver.com:yourproject.git)

3) Просмотр веток:

git branch -a

4) Создание ветки:

git branch BranchName

5) Переключение на ветку:

git checkout BranchName

4a) Создание и переключение на ветку:

git checkout -b BranchName

6) Создание игнорируемого файла:

touch .gitignore

6а) Внесение имен в файл, которые не будут добавляться в репозиторий

notepad .gitignore

7) Добавление файлов в StageArea:

git add .

8) Добавление файлов в локальный репозиторий из StageArea:

git commit -m ‘change note’

9) Добавление файлов в онлайн репозиторий

git push remote origin (первый раз)

или

git push -u RemoteBranch LocalBranch

последующие push:

git push

10) Забрать изменения с сервера:

git pull**Основные команды git bash**

Создание репозитория:

git init

Создание файла:

touch FileName

nano FileName

Информация о репозитории:

git status

Информация о коммитах и ветках:

git log

git log --oneline

Добавить файлы в Stage:

git add FileName.FileType

git add \*.FileType

git add .

Добавление файлов в репозиторий из Stage:

git commit -m ‘about commit’

Посмотреть n последних коммитов. Выведем три последних коммита:

git log -n 3 --oneline

Для вывода списка коммитов, начиная с какой-то временной метки, ключ --since=”<date> <time>”:

git log --since="2018-03-05 23:21:00" --oneline

Для вывода списка коммитов до какой-то даты, ключ --until:

git log --until="2018-03-05 23:21:00" --oneline

Список коммитов, сделанных конкретным автором:

git log --author="Writer" --oneline

Все коммиты, в которых встречается слово «create»:

git log --grep="create" --oneline

Передвинуть указатель HEAD на коммит с id=2b826bb:

git checkout 2b826bb

Скопировать снимок репозитория относительно последнего коммита ветки master, т.е. того на который указывает HEAD:

git checkout master

Посмотреть содержимое репозитория:

ls -la

Посмотреть содержимое файла HEAD:

cat HEAD

Просмотр файлов в репозитории:

git ls-tree master

Удаление директории из репозитория:

git rm -r DirName

Удаление файла из репозитория:

rm FileName

git rm FileName

git commit -m ‘remove FileName’

или

git rm FileName

git commit -m ‘remove FileName’

Скопировать файл из репозитория в рабочую папку:

git checkout FileName

Переименование файла:

переименовывается файл в ОС, затем

git add .

git commit -m ‘rename FileName’

или в git

git mv OldFilename NewFileName

git commit -m ‘rename FileName’

Отмена изменений в файлах в рабочей директории, возврат файлов из репозитория:

git checkout -- FileName

-- означает, что файл будет взять из текущей ветки

Замена последнего коммита:

git commit --amend -m ‘new commit’

Отмена изменений в файле в выбранном коммите:

git checkout CommitNumber BranchName FileName

git add FileName

git commit -m ‘return Filename from CommitNumber’

Использование git revert для быстрой отмены изменений -- отмена коммита:

git revert CommitNumber

Удаление коммитов из репозитория (без изменения рабочей директории) (ключ --soft):

git reset --soft ToCommitNumber

Удаление коммитов из репозитория и очистка stage (без изменения рабочей директории) (ключ --mixed):

git reset --mixed ToCommitNumber

Удаление коммитов из репозитория, очистка stage и внесение изменений в  рабочую директорию (ключ --hard). *Все компоненты git (репозиторий, stage и рабочая директория) будут приведены к одному виду в соответствии с коммитом, на который будет перенесен указатель HEAD:*

git reset --hard ToCommitNumber

Отмена изменений work directory (возврат файлов из commit):

git checkout FileName

или

git checkout .

Прекращение отслеживания файла:

git rm --cashed FileName

Удаление файла из Stage:

git restore --staged FileName

Сделать ответвление от ветки:

git branch ChildBranch ParrentBranch

Слияние ветки MergeBranch в текущую активную ветку. *Флаг --no-ff вынуждает Git всегда создавать новый объект коммита при слиянии, даже если слияние может быть осуществлено алгоритмом fast-forward. Это позволяет не терять информацию о том, что ветка существовала, и группирует вместе все внесённые изменения*:

git merge --no-ff MergeBranch

Отправки локальной ветки в удаленный репозиторий, создание удаленной ветки и связывание ее с локальной:

git push -u origin LocalBranch

Связать локальную ветку с удаленной:

при push: git push -u origin RemoteBranch

или

git branch -u RemoteBranch/LocalBranch

или

git branch --set-upstream-to RemoteBranch/LocalBranch

Отправить все локальные ветки в удаленный репозиторий:

git push origin --all

Отправить все локальные метки в удаленный репозиторий:

git push origin --tags

Отправка ветки CurrentBranch в уже созданную удаленную ветку CurrentBranch:

git push origin CurrentBranch

Удаление локальной ветки:

git branch -d BranchName

Удаления remote tracking ветки feature:

git push origin -d RemoteBranch

или

git push origin :feature

Пометить коммит тэгом:

git tag -a TagName

Показать изменения между файлами рабочей директории и файлами в Stage:

git diff

Показать изменения между файлами Stage и файлами в репозитории:

git diff --staged

Показать список удаленных репозиториев:

git remote

Показать локальные и отслеживаемые удалённые ветки:

git branch -vv

Отправка изменений из локальной ветки LocalBranch в удаленный репозиторий в ветку RemoteBranch, предварительно создав ее там:

git push origin LocalBranch:RemoteBranch

*Но при таком способе публикации, не устанавливается связь между локальной версией ветки и опубликованной. Т.е. если кто-то закоммитит изменения в эту удаленную ветку и вы сделаете git pull, то будет ошибка*

*Т.е. гит не знает с какой ветки ему мерджить. Поэтому можно либо каждый раз указывать это руками:*

*git pull origin feature*

*Либо прописать в конфиге:*

*git config branch.feature.remote origin*

*git config branch.feature.merge refs/heads/feature*

**Восстановление одного файла из репозитория:**

Из локального репозитория:

git checkout path/to/file

или

git checkout BranchName path/to/file

Из удаленного репозитория:

git fetch

git checkout origin/master -- Foldername/Filename

\_\_\_

git pull = git fetch + git merge

git fetch не влияет на локальные файлы и изменения

git merge вливает эти изменения в локальную копию

Создать локальную ветку feature и подключить её к удалённой ветке origin/feature, после чего переключиться в эту ветку:

git checkout --track -b feature origin/feature

Это важно на данном этапе, поскольку просто команда pull смержит удалённую ветку к нам в master, а это не то, что нам нужно.

Иногда возникает необходимость срочно переключиться в другую ветку, например для багфикса. Но на полноценный коммит в этой ветке еще не хватает. Для этого существует команда git stash

Теперь можно смело переключаться в другую ветку и работать там.

По возвращению в эту ветку, необходимо сделать git stash apply:

Общее правило такое: если мы работаем с веткой самостоятельно и не планируем публиковать её на сервере — то выгоднее использовать rebase. Если же мы публикуем ветку командой push, то использовать rebase НЕЛЬЗЯ, иначе мы автоматически инвалидируем работу коллег

Вкратце: rebase запоминает коммиты из ветки в виде патчей, «перематывает» текущую ветку (как будто и не было никакого branch) и применяет патчи, оформляя их в виде коммитов. В отличие от rebase, merge делает слияние двух веток в одну.

Сливаем изменения из рабочей ветки feature в master, ветка feature нигде не публиковалась, никто из коллег с ней не работал:

git checkout master

git rebase feature

Подливаем изменения из master в рабочую ветку feature, ветка feature опубликована на удалённом репозитарии, с ней также работают коллеги:

git checkout feature

git merge master

Сливаем изменения из рабочей ветки feature в master, ветка feature публиковалась на удалённом репозитарии для совместной работы:

git checkout master

git merge feature