git clone

— копирует репозиторий и устанавливает настройки для наблюдения за оригиналом (применяются в командах fetch, push, pull).

$ git clone repo

— копирует репозиторий сохранив название корневой папки.

```
$ git clone repo newRepo
```

— копирует репозиторий переименовав корневую папку репозитория.

```
$ git clone git@github.com:/
zlatov/simpleparser.git
```

— скопирует репозиторий в папку simpleparser.

```
$ git clone git@github.com:/
zlatov/simpleparser.git temp
```

— скопирует репозиторий в папку temp.

git status

— показывает текущее состояние репозитория.

git init

— создаёт новый репозиторий.

```
$ git init --bare
```

— создать „голый“ репозиторий в текущей папке (используется как централизованное хранилище).

```
$ git init repo
```

— создать папку „repo“ в текущей папке и создать в ней репозиторий.

```
$ git init --bare repo
```

— аналогично предыдущему примеру, но создаётся „голый“ репозиторий.

git add

— добавляет файл или группу файлов в index (staging area).

```
$ git add .
```

— добавить все файлы и папки текущей директории в index.

```
$ git add path/to/file1
```

— добавить только один файл (или папку) в index.

```
$ git add .
```

```
$ git add -u
```

— добавляем в индекс измененные и новые и удаляем из индекса удаленные файлы

git commit

— переносит изменения из index (staging area) в local repo.

```
$ git commit -m «comment»
```

— фиксирует изменения, добавленные в staging area.

```
$ git commit -a -m «comment»
```

— фиксирует все изменения, сделанные в рабочей директории. Особенность -a в том, что перед этой командой ней не нужно выполнять команду git add . (это не распространяется на новые файлы, которые ещё никогда ранее не добавлялись).

git checkout

— загрузить любое состояние репозитория (checkout можно выполнить, только тогда, когда у нет никаких изменений в текущем состоянии). При каждой операции checkout ссылка HEAD указывает на текущее положение (читать как: метка HEAD — это текущее состояние репозитория, т.е. переносится чекаутом).

```
$ git checkout
branchNameOrCommitHash
```

Например:

```
$ git checkout a71b72
```

— при таком переключении меняется текущая ветка на ветку «(no branch)».

```
$ git checkout master
```

— переключиться на ветвь master (переключение происходит на верхушку ветви).

```
$ git checkout master~1
```

— переключиться на одну фиксацию назад относительно верхушки ветви master.

```
$ git checkout head~1
```

— переключиться на одну фиксацию назад относительно текущего вашего положения.

```
$ git checkout a71b72~3
```

— переключиться на три фиксации назад относительно фиксации с именем, начинающимся на «a71b72».

```
$ git checkout -b branchName
branchNameOrCommitHash
```

— переключается на фиксацию branchNameOrCommitHash и тут же создаёт от неё новую ветвь branchName и переключается на эту ветвь. Примечание: если переключиться назад, то команда git log не покажет историю «будущего», но история не потерена, выполните команду git log –all, чтобы увидеть и узнать hash последнего коммита и перейти к нему, если неоходимо; или выполните команду git checkout master, чтобы сразу перейти на верхушку ветки.

git branch

— просмотр, создание и удаление ветвей разработки.

```
$ git branch
```

— просмотреть список существующих ветвей и узнать какая ветвь выбрана текущей.

```
$ git branch -r
```

— просмотреть список удалённых ветвей.

```
$ git branch branchName
```

— создать ветвь с именем name. Создание ветви не приводит к переключению в эту ветвь.

```
$ git checkout -b branchName
```

— создать ветвь и переключиться на нее.

```
$ git branch -d branchName
```

— удаляет ветвь branchName. Если ветвь ещё не была с лита с основной ветвью, то git предупредит об этом и удаления не будет.

```
$ git branch -D branchName
```

— удаляет ветвь branchName, даже если вы ещё не сливали её изменения с основной ветвью.

```
$ git checkout master
```

```
$ git merge testing
```

```
$ git branch -d testing
```

— после выполненных работы в ветке testing сольём с рабочей веткой master, после чего не забываем удалить ненужную ветвь.

git diff

— увидеть изменения между work tree и index или между index и local repo.

```
$ git diff
```

— изменения между work tree и index (внесенны изменения, но не проиндексированы).

```
$ git diff --cached
```

— изменения между index и local repo (проиндексированные изменения, но не закомичены).

```
$ git diff master~2..master
```

— изменения при перемещении от фиксации master~2 к фиксации master. Возможно указание фиксаций в обратном порядке при этом логика изменений поменяется. Возможно указание разных веток.

git log

— история фиксаций.

```
$ git log
```

— от начала до текущего состояния HEAD.

```
$ git log
branchNameOrCommitHash
```

— от начала до указанной ветви или фиксации.

```
$ git log
branchNameOrCommitHash..
branchNameOrCommitHash
```

— история указанного диапазона.

```
$ git log branchName...
branchName
```

— общая история указанного диапазона (по всем веткам).

```
$ git log FETCH_HEAD
```

— от начала до состояний FETCH_HEAD (состояние FETCH_HEAD доступно после операции fetch или pull).

```
$ git log HEAD..FETCH_HEAD
```

— от места совпадения HEAD и FETCH_HEAD, и до конца истории.

```
$ git log HEAD...FETCH_HEAD
```

— общая историю от места совпадения HEAD и FETCH_HEAD, и до конца истории.

git remote

— наблюдения за удаленными репозиториями.

```
$ git remote
```

— список существующих.

```
$ git remote add name path
```

— добавляет с именем name расположенного по пути path.

```
$ git remote rm name
```

— удаляет с именем name.

```
$ git remote rename
remoteNameOld remoteNameNew
```

— переименовывает из remoteNameOld в remoteNameNew.

git fetch

— загрузить удалённый репозиторий в раздел «наблюдения» локального репозитория. Загрузить репозиторий - ещё не значит «слиться» с ним. Для слияния используется команда git merge.

```
$ git fetch
```

— подгружает данные по текущей ветке из соответствующей ветки удалённого репозитория, или подгружает данные из соседней ветки локального репозитория (подтянутся изменения, но не закоммитятся).

```
$ git fetch repo branch
```

— подгружает содержимое ветки branch репозитория repo в объект состояния FETCH_HEAD.

git merge

— слияние двух состояний HEAD и FETCH_HEAD в новое HEAD состояние. Если происходит конфликт изменений, то эти файлы выходят из index (staging area), до тех пор, пока вы не исправите конфликт и не поместите их обратно в index командой git add .. Команда слияния всегда занимает отдельную фиксацию, при слиянии не допускается изменение каких-либо файлов.

```
$ git merge branch
```

— слить ветку branch с текущей веткой.

git pull

— подгрузка и слияние. Данная операция аналогична последовательному выполнению двух операций: git fetch и git merge.

```
$ git pull
```

— обновить текущую ветку репозитория до состояния другой локальной или удалённой текущей ветки репозитория.

```
$ git pull repoName branchName
```

— обновить ветку branchName локального репозитория до состояния удалённой ветки branchName репозитория repoName.

git push

— втолкнуть изменения текущего репозитория в удалённый. По умолчанию, вталкивать данные можно только в «голые» репозитории.

```
$ git push
```

— вталкивает данные всего локального репозитория в соответствующие ветки удалённого репозитория.

```
$ git push origin master
```

— вталкивает данные текущей ветви в ветвь master удалённого репозитория origin. Если ветвь master не существует в удалённом репозитории, она будет создана. Если для удалённой ветви невозможно выполнить FAST-FORWARD слияние, то данные не будут «втолкнуты» (об этом будет сообщено). Данный способ вызова команды является первым при первой фиксации в пустой удалённый репозиторий.

git reset

— откатить изменения или неудачное слияние до последнего стабильного состояния (до последней фиксации).

```
$ git reset
```

— отменит операцию слияния, но оставит изменения в конфликтующих файлах и/или в index области. При этом состоянии Git будет ожидать от вас фиксации или полной отмены изменений.

```
$ git reset --hard
```

— отменит операцию слияния, очистит index область и вернёт все файлы work tree до состояния последней фиксации по текущей ветви.

git tag

— отметить текущее состояние, как некоторое конечное состояние для новой версии вашего проекта. Используется для создания списка стабильных версий проекта. Имя метки tag может использоваться в командах checkout и других командах, на ровне с именами фиксаций, именами ссылок (HEAD, FETCH_HEAD) и именами веток (master и т.п.).

```
$ git tag
```

— список существующих tag-ов

```
$ git tag -m «описание» tagname
```

— создать метку tag для текущего состояния (на текущей ветке) с именем tagname.

```
$ git tag -d tagname
```

— удалить метку tag с именем tagname