### Git шпаргалка краткая

Основное

**git init** - инициализирует пустой репозиторий в папке

**git add \*** - добавит все файлы в репозиторий

**git commit -m "Initial Commit"** - фиксирует все измененния(делает слепок репозитория) или коммитит репозиторий

**git status** — это еще одна важнейшая команда, которая показывает информацию о текущем состоянии репозитория

**git log** – покажет все сделанные коммиты

**git show b10cc123** - покажет коммит с ключом b10cc123

**git diff 09bd8cc..ba25c0ff** – сравнит коммиты с ключами 09bd8cc и ba25c0ff

**git checkout 09bd8cc1 hello.txt** вернет вайл hello.txt к состоянию которое было при метке 09bd8cc1

Удаленные репозитории

**git remote add origin** [**https://github.com/Seriyyy95/testing.git**](https://github.com/Seriyyy95/testing.git) - добавит удаленный репозиторий с <https://github.com/Seriyyy95/testing.git> в и назовет его origin

**git remote** –v покажет список удаленных репозиториев

**git push origin master** – отправит наш репозиторий на ветке master в удаленный репозиторий origin (<https://github.com/Seriyyy95/testing.git>)

**git pull origin master** – скачает изменения с origin (<https://github.com/Seriyyy95/testing.git)и> вольет их в ветку master

**git push** — вливание локальных изменений в удаленный репозиторий

**git pull** — вливание изменений из удаленного репозитория в локальный

**git clone** [**https://github.com/tutorialzine/awesome-project.git**](https://github.com/tutorialzine/awesome-project.git) - клонирует удаленный репозиторий

Ветвления, ветки

**git branch –a** – покажет все ветки которые доступны

**git branch develop** – создаст новую ветку develop

**git checkout -b develop** – создаст ветку develop и передет на нее

**git checkout master** – переключится на ветку master

**git merge develop --no-ff** - сливает ветку develop с той на которой сейчас находится голова (HEAD)

**git branch -d develop** – удалит ветку develop

### Git шпаргалка Полная

**Создание локального репозитория**  
  
Создание репозитория в папке где выполняется команда  
**$ git init**  
  
Создание репозитория в указанном каталоге  
**$ git init <directory>**  
 **Создание репозитория Git для совместной работы**  
 **$ git init --bare --share sharedproject.git**  
  
Данная команда создает каталог с именем sharedproject.git c правами на запись в него. Подробнее [тут](http://pr0git.blogspot.ru/2015/02/git-git-windows-1.html).  
  
**Клонирование удаленного репозитория в локальный**  
  
Клонирование удаленного репозитория в локальный каталог с именем по умолчанию  
**$ git clone https://github.com/n0tb0dy/RemoreBranches.git**  
  
Клонирование удаленного репозитория в локальный каталог с указанным именем  
**$ git clone https://github.com/n0tb0dy/RemoreBranches.git LocalBranches**  
  
**Клонирование локального репозитория на удаленный**  
  
Если у вас уже есть локальный репозиторий Git и вы хотите его выложить в общий доступ, то сперва вам надо создать удаленный репозиторий (например на GitHub), а затем дать команды представленные ниже, изменив соотвественно часть с названием вашего репозитория.  
  
1. Связываем локальный репозиторий с удаленным  
**$ git remote add origin https://github.com/n0tb0dy/UpRemote.git**  
  
2. Верифицируем что удаленный репозиторий связан с нашим  
**$ git remote -v**  
  
3. Публикуем ветку master на удаленном репозитории  
**$ git push -u origin master**  
  
Более подробно можно почитать [тут](http://pr0git.blogspot.ru/2015/02/git-1_20.html).  
  
**Задаем имя пользователя и электронную почту**  
  
Глобально для всех проектов текущего пользователья  
**$ git config --global user.name "John Doe"**  
**$ git config --global user.email johndoe@example.com**  
  
Для конкретного проекта (эти настройки переопределят глобальные)  
**$ git config --local user.name "John Doe"**  
**$ git config --local user.email johndoe@example.com**  
  
**Просмотр настроек Git**  
  
**Всех** (глобальных, системных и локальных). Некоторые параметры могут появится в списке несколько раз, так как читаются из трех файлов настроек. Подробнее [тут](http://pr0git.blogspot.ru/2015/02/git_64.html).  
**$ git config --list**  
  
**Локальных** для определенного проекта  
**$ git config --local --list**  
  
**Системных**  
**$ git config --system --list**  
  
**Получение справки (помощи) по команде Git**  
  
**$ git help <verb>**  
**$ git <verb> --help**  
  
Например выведем справку по команде config (откроется браузер со справкой)  
**$ git help config**  
  
**Настройка русских шрифтов (cp1251) в Git**  
  
Настраиваем правильное отображение файлов с русскими названиями в командах Git  
**$ git config --local core.quotepath false**  
  
Настраиваем кодировку **Windows cp1251** для коммитов в Git  
**$ git config --local core.pager "iconv.exe -f cp1251 -t utf-8 | less"**  
**$ git config --local i18n.commitEncoding utf8**  
**$ git config --local i18n.logoutputencoding cp1251**  
  
Эти команды замечательно работают в **msysgit 1.9.5**. Как будет в других версия не знаю. Но надеюсь, что в более новых тоже будет работать. Более подробно про настройку русского языка в Git можно почитать [тут](http://pr0git.blogspot.ru/2015/02/git_4.html). Так же они правильно работают  при установке Git из пакетов Cygwin, подробнее можно почитать [тут](http://pr0git.blogspot.ru/2015/03/git-windows-cygwin.html).  
  
Так же можно задать кодовую страницу для файлов проекта командой  
**$ git config --local i18n.filesEncoding windows-1251**  
  
ну или просто строкой в разделе [i18n]  
filesEncoding = windows-1251  
  
А вообще лучше вести проекты в кодировке UTF-8, если это возможно конечно.  
  
**Просмотр информации о состоянии файлов в Git**  
  
Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда:  
**$ git status**  
  
И ее более краткий вывод:  
**$ git status -s**  
  
Просмотр разницы (что конкретно было изменено в файлах) между рабочим каталогом и индексом (staged area)  
**$ git diff**  
  
Просмотр разницы между последним коммитом и индексом  
**$ git diff --staged**  
  
Более подробно смотрим [тут](http://pr0git.blogspot.ru/2015/02/git_13.html).  
  
**Фиксация изменений (коммит)**  
  
Если дать команду **git commit** без дополнительных параметров, то сперва будет вызван редактор для ввода комментария к коммиту и после сохранения комментария будет произведен коммит (фиксация изменений)  
**$ git commit**  
  
Чтобы включить в комментарий к коммиту информацию о том какие именно были сделаны изменения в каких файлах надо дать команду  
**$ git commit -v**  
  
По существу по данной команде в комментарий будет также помещена дельта diff изменений, таким образом вы сможете точно увидеть всё, что сделано.  
  
Чтобы редактор не вызывался, можно написать комментарий прямо в командной строке в ключе **-m**  
**$ git commit -m "Commit Comment"**  
  
Автоматически добавить все измененные файлы в коммит  
**$ git commit -a**  
  
**Удаление файлов из Git**  
По существу это удаление файла из отслеживаемых. Если файл уже был до этого закоммичен в Git, то из старых коммитов его по прежнему можно будет достать.  
  
Удаление файла из отслеживаемых Git, а так же его физическое удаление из рабочего каталога  
**$ git rm <file\_name>**  
  
Удаление проиндексированного измененного файла  
**$ git rm -f <file\_name>**  
  
Удаление файла из индекса, но сохранение его в рабочем каталоге  
**$ git rm --cached <file\_name>**  
  
Более подробно смотрим [тут](http://pr0git.blogspot.ru/2015/02/git_13.html).  
  
**Переименование файла**  
  
**$ git mv <old\_file\_name> <new\_file\_name>**  
  
**Просмотр истории коммитов**  
  
Самый простой вариант это **git log** с разными ключами (смотрим help). Тут приведу просто примеры. А подробнее [тут](http://pr0git.blogspot.ru/2015/02/git_11.html) или в мануале.  
  
  
Вывод простой истории коммитов  
**$ git log**  
  
Вывод последних n записей, в примере вывод двух последних записей  
**$ git log -2**  
  
Вывод дельты (diff) разницы между последними двумя изменениями (на уровне строк)  
**$ git log -p -2**  
  
Вывод изменений между двумя последними коммитами на уровне слов  
**$ git log -p -2 --word-diff**  
  
Вывод краткой статистики по 2 последним коммитам  
**$ git log -2 --stat**  
  
И очень полезный ключ --pretty (позволяет изменить формат вывода лога)  
**$ git log --pretty=oneline**  
  
**$ git log --pretty=format:"%h - %an, %ar : %s"**  
  
Параметры ключа **format**

| **Параметр** | **Описание выводимых данных** |
| --- | --- |
| %H | Хеш коммита |
| %h | Сокращённый хеш коммита |
| %T | Хеш дерева |
| %t | Сокращённый хеш дерева |
| %P | Хеши родительских коммитов |
| %p | Сокращённые хеши родительских коммитов |
| %an | Имя автора |
| %ae | Электронная почта автора |
| %ad | Дата автора (формат соответствует параметру --date=) |
| %ar | Дата автора, относительная (пр. "2 мес. назад") |
| %cn | Имя коммитера |
| %ce | Электронная почта коммитера |
| %cd | Дата коммитера |
| %cr | Дата коммитера, относительная |
| %s | Комментарий |

Можно так же посмотреть ASCII граф веток коммитов по ключу --graph  
**$ git log --pretty=format:"%h  %s" --graph**  
  
Есть параметры, ограничивающие по времени, такие как **--since** и **--until**, весьма полезны. Например, следующая команда выдаёт список коммитов, сделанных за последние две недели:  
**$ git log --since=2.weeks**  
  
Другой полезный фильтр это опция –S, которая как параметр принимает строку и показывает только те коммиты где эта строка была изменена, добавлена или удалена.  
**$ git log -S<stirng>**  
  
Пример будет искать строку MyStringForSearch  
**$ git log -SMyStringForSearch**  
  
Список коммитов с хэшем (короткое число)  
**$ git log --oneline**  
  
**Отмена изменений**  
  
Изменение комментария к последнему комииту, но только в том случае, если после последнего коммита не было ни каких изменений в рабочем каталоге  
**$ git commit --amend**  
  
Отмена индексации файла (исключение из индекса)  
**$ git reset HEAD <file>**  
  
Отмена изменений файла (до внесения файла в коммит)  
**$ git checkout -- <file>**  
  
С этой командой надо быть особо осторожным, подробнее [тут](http://pr0git.blogspot.ru/2015/02/git_72.html).  
  
Удаление раз и навсегда последнего коммита. Его больше ни кто ни когда не увидит. И вы в том числе :). Произойдет откат на предыдущий коммит. Все изменения которые были в последнем коммите будут утеряны. Хорошо подумайте прежде чем это делать.  
**$ git reset --hard HEAD~1**  
  
**Работа с удаленными репозиториями**  
  
Просмотр удаленных репозиториев  
**$ git remote**  
  
Более подробный вывод о них  
**$ git remote -v**  
  
Добавление удаленного репозитория (вместо origin можно задать любое слово)  
**$ git remote add origin https://github.com/n0tb0dy/UpRemote.git**  
**$ git remote add tr https://github.com/n0tb0dy/UpRemote.git**  
  
Получение изменений с удаленного репозитория под именем tr в локальную ветку tr  
**$ git fetch tr**  
  
Отправка данных на удаленный репозиторий. Формат **git push [удал. сервер] [локальная ветка]**  
**$ git push origin master**  
  
Инспекция удаленного репозитория **git remote show [удал. сервер]**  
**$ git remote show origin**  
  
Переименование удаленных репозиториев (по существу переименование локальной ссылки на удаленный репозиторий)  
**$ git remote rename <old\_name> <new\_name>**  
**$ git remote rename tr newtr**  
  
Удаление удаленного репозитория :) (попросту отключение от него - в примере от origin)  
**$ git remote rm origin**  
  
Подробней о работе с удаленными репозиториями [тут](http://pr0git.blogspot.ru/2015/02/git_16.html).  
  
Если у вас свой собственный репозиторий Git на сервере с само подписанным сертификатом, то перед любыми командами работы у удаленным репозиторием (clone, fetch, push, pull и т.п.), Git будет ругаться на само подписанный сертификат. Решить проблему можно изменив чуток конфиг  
**$ git config --local http.sslVerify false**  
  
Или же перед каждой операцией работы с удаленным репозиторием вставлять доп команду  
**$ git -c http.sslVerify=false push origin newbranch**  
  
А вообще настройка своего сервера Git это отдельная тема. Частично рассмотрена мной [тут](http://pr0git.blogspot.ru/2015/03/git-windows-server-20xx.html).  
  
**Работа с ветками**  
  
Посмотреть локальные ветки  
**$ git branch**  
  
Посмотреть последний коммит на каждой из локальных веток  
**$ git branch –v**  
  
Чтобы посмотреть все существующие локальные и удаленные ветки можно дать команду  
**$ git branch –a**  
  
Посмотреть последние коммиты на всех ветках (локальных и удаленных)  
**$ git branch –a -v**  
  
Посмотреть отслеживаемые ветки  
**$ git branch –vv**  
  
Сделать ветку локальную ветку serverfix отслеживаемой  
**$ git branch -u origin/serverfix**  
  
Создать ветку  
**$ git branch <имя\_ветки>**  
  
Создать ветку на определенном коммите  
**$git branch new\_branch 5a0eb04**  
  
Переименовать ветку  
**git branch -m <oldname> <newname>**  
  
Переименовать текущую ветку  
**git branch -m <newname>**  
  
Переключится на ветку  
**$ git checkout <имя\_ветки>**  
Создать ветку и сразу же переключится на нее  
**$ git checkout -b <имя\_ветки>**  
Слияние веток (в примере находимся на ветке master и сливаем с ней ветку hotfix)  
**$ git checkout master**  
**$ git merge hotfix**  
  
Удалить ветку  
**$ git branch -d <имя\_ветки>**  
Удалить ветку serverfix на удаленном сервере  
**$ git push origin --delete serverfix**  
  
**Работа с метками**  
  
Посмотреть все (перечисляет в алфавитном порядке, а не по времени их создания)  
**$ git tag**  
  
Посмотреть попадающие под маску  
**$ git tag -l 'v1.4.2.\*'**  
  
Создать метку на текущем коммите (ключ **-а**) с меточным сообщением (ключ **-m**)  
**$ git tag -a v1.4 -m 'my version 1.4'**  
  
Если ключ **-m** не указывать то откроется окно редактора чтобы ввести сообщение  
  
Создание легковесной метки на текущем коммите  
**$ git tag <имя\_метки>**  
**$ git tag MyTAG**  
  
Посмотреть метки вместе с комментариями к коммитам, а так же с именами поставивших метки  
**$ git show <tag>**  
**$ git show MyTAG**  
  
Так же можно выставлять метки и на уже пройденные коммиты. Подробнее о метках [тут](http://pr0git.blogspot.ru/2015/02/git_17.html).  
  
**Задание псевдонимов для команд Git**  
  
Псевдонимы можно создать как в конфигурационных файлах Git, так и в конфиге Bash, но важно понимать в чем разница.  
  
**Задание псевдонимов в конфигах Git**  
  
**$ git config --global alias.co checkout**  
**$ git config --global alias.br branch**  
**$ git config --global alias.ci commit**  
**$ git config --global alias.st status**  
  
Теперь достаточно давать команды  
**$ git co**  
**$ git br**  
**$ git ci**  
**$ git st**  
  
То есть через задание алиасов в конфиге Git мы не избавляемся от необходимости писать команду **git**, но все же это короче.  
  
Кроме того в эти команды так же можно подставлять параметры  
**$ git config --global alias.unstage 'reset HEAD --'**  
  
Это делает эквивалентными следующие две команды:  
**$ git unstage fileA**  
**$ git reset HEAD fileA**  
  
Более подробно по алисы в конфигах Git читаем [тут](http://pr0git.blogspot.ru/2015/02/git-git_17.html).  
  
Об алиасах заданных через Bash читаем [тут](http://pr0git.blogspot.ru/2015/02/git-bash.html).  
  
**Сравнение файла в разных коммитах**  
  
**$ git diff ffd6b37 c258082 --cc test.txt**  
  
**С помощью внешних утилит ExamDiffPro и P4Merge**  
  
Смотрим изменения файла test.txt между двумя коммитами  
**$ git difftool 9491cc8 02c1df6 --tool=edp --cc test.txt**  
**$ git difftool 9491cc8 02c1df6 --tool=p4m --cc test.txt**  
  
**Слияние (merge)**  
  
Самая главная и нужная команда слияния, это отмена слияния :)  
**$ git merge --abort**  
  
**Всяко разно**  
  
Просмотр истории перемещения указателя HEAD  
**$ git reflog**