Шпаргалка по GIT

Создание

* Клонировать существующий репозиторий

$ git clone ssh://user@domain.com/repo.git

* Создать локально новый репозиторий

$ git init

**Локальные изменения**

* Файлы в рабочей директории были изменены

$ git status

* Изменения в отслеживаемых файлах

$ git diff

* Добавить все текущие изменения в следующий коммит

*Устаревающий способ*

$ git add .

*Советую использовать эту команду*

$ git add -A

* Добавить некоторые изменения в <файле> в следующий коммит

$ git add -p <файл>

* Зафиксировать все локальные изменения в отслеживаемых файлах

$ git commit -a

* Закоммитить ранее добавленные изменения

$ git commit

* Изменить последний коммит  
  *Не изменяйте опубликованные коммиты!*

$ git commit --amend

**История коммитов**

* Показать всю историю коммитов начиная с последних

$ git log

* Показать историю изменений определенного файла

$ git log -p <файл>

* Кто, какие и когда изменения вносил в <файл>

$ git blame <файл>

**Ветки и теги**

* Список всех существующих веток

$ git branch -av

* Переключение в HEAD ветки

$ git checkout <ветка>

* Создать новую ветку на основе текущей HEAD

$ git branch <новая-ветка>

* Создать новую ветку и сразу переключится в нее

$ git checkout -b <новая-ветка>

* Создать новую отслеживаемую ветку на основе удаленной ветки

$ git checkout --track <репо/ветка>

* Удалить локальную ветку

$ git branch -d <ветка>

* Присвоить текущему коммиту тег

$ git tag <имя-тега>

**Обновление и публикация**

* Список всех настроенных удаленных репозиториев

$ git remote -v

* Показать информацию об удаленном репозитории

$ git remote show <репо>

* Добавить новый удаленный репозиторий с именем <remote>

$ git remote add <короткое-имя> <url>

* Скачать все изменения из <remote>, но не интегрировать их в HEAD

$ git fetch <репо>

* Скачать изменения и сразу слить/интегрировать их в HEAD

$ git pull <репо> <ветка>

* Опубликовать локальные изменения в удаленном репозитории

$ git push <репо> <ветка>

* Удалить ветку в удаленном репозитории

$ git branch -dr <репо/ветка>

* Опубликовать теги

$ git push --tags

**Слияние и перемещение**

* Слить <ветку> с текущей HEAD

$ git merge <ветка>

* Переместить вашу текущую HEAD на <ветку>  
  *Не перемещайте опубликованные коммиты!*

$ git rebase <ветка>

* Отменить перемещение

$ git rebase --abort

* Продолжить перемещение после разрешения конфликтов

$ git rebase --continue

* Используйте свой настроенный инструмент слияния для разрешения конфликтов

$ git mergetool

* Используйте свой редактор для решения конфликтов и (после разрешения) отметить файлы как разрешенные

$ git add <разрешенный-файл>

$ git rm <разрешенный-файл>

**Отмена**

* Удалить все локальные изменения в рабочем каталоге

$ git reset --hard HEAD

* Удалить локальные изменения в отдельном <файле>

$ git checkout HEAD <файл>

* Откатить коммит (но проще создать новый коммит с противоположными изменениями)

$ git revert <коммит>

* Сбросить головной указатель HEAD на предыдущий коммит

…и отменить все изменения после него

$ git reset --hard <коммит>

…и сохранить все изменения как незакоммиченые

$ git reset <коммит>

…и сохранить неотслеживаемые локальные изменения

$ git reset --keep <коммит>

**Лучшие практики**

**Добавляйте в коммит связанные изменения**

Коммит должен содержать связанные изменения. Например, правки двух разных багов должны быть в двух разных коммитах. Маленькие коммиты помогают другим разработчикам проще понимать изменения и откатить их, если что-то пойдет не так.   
При помощи таких инструментов, как область подготовленных файлов и возможности добавлять только часть изменений из файла, Git упрощает создание очень мелких коммитов.

**Делайте коммиты часто**

Частая публикация коммитов позволяет вам делать их маленькими и, опять же, помогает добавлять в коммит только связанные изменения. Более того, почаще делитесь своим кодом с остальными. Таким образом, всем будет легче, если вы чаще будете интегрировать свои изменения и это поможет избежать конфликтов при слиянии. Большие и редкие коммиты усложняют жизнь и затрудняет решение конфликтов.

**Не публикуйте коммиты с незаконченной работой**

Вы должны коммитить код, только если он завершен. Это не значит, что вы должны полностью закончить работу над большой задачей перед коммитом. Наоборот: разделите работу над задачей на логические части и помните о своевременной и частой публикации коммитов. Но не публикуйте изменения только чтобы в репозитории что-то появилось перед вашим уходом из офиса в конце рабочего дня. Если вы собираетесь опубликовать коммит только потому, что вам нужна чистая рабочая область (при переключении веток, перед pull и т.д.) присмотритесь к функци “stash”.

**Тестируйте код перед коммитом**

Не поддавайтесь искушению закоммитить что-то, что на ваш взгляд уже готово. Протестируйте внимательно и убедитесь в том, что работа закончена и в коде нет ошибок (на сколько это может проверить один человек). В то время, как полусырые изменения в вашем локальном репозитории вы можете простить себе самому, вы должны тщательно протестировать код перед тем как публиковать/делится своим кодом с другими.

**Пишите хорошие комментарии**

Начните свое сообщение с краткого резюме своих изменений (ориентируйтесь на 50 символов). Отделите это сообщение от основного текста пустой строкой. В теле сообщения следует дать подробные ответы на следующие вопросы:

* Какова была причина изменений?
* В чем отличие от предыдущего коммита?

Используйте слова в настоящем времени («меняется», не «был изменен» или «изменения») чтобы при **git merge** сгенерированное сообщение выглядело хорошо.

**Система контроля версий не хранилище бекапов**

Хранениерезервных копий файлов на удаленном сервере это только приятный побочный эффект использования системы контроля версий. Но вы не должны использовать Git только для этого. При работе с системой обратите особое внимание на семантику коммитов (смотри пункт 1) — вы не должны просто запихивать файлы в репозиторий.

**Используйте ветки**

Ветвление — это одна из самых мощных возможностей Git. И это не случайно: быстрое и простое ветвление было требованием номер один с самого начала. Ветки являются самым простым инструментом чтобы избежать смешивания разных линий разработки. Вы должны повсеместно использовать ветки в своей разработке: для работы над новыми задачами, для исправления багов, для идей…

**Согласуйте рабочий процесс**

Git позволяет выбрать из большого количества рабочих процессов: длинные ветки, тематические ветки, слияние или перемещение, gitflow… Что из этого выберите вы зависит от нескольких факторов: ваш проект, ваше общее направление развития, среда разработки и (возможно самое важное) ваши личные предпочтения и предпочтения вашей команды. Что бы вы не выбрали убедитесь, что все следуют процессу, который был согласован.

**Помощь и документация**

Вызов помощи git в командной строке

$ git help <command>