Встановлення

<https://git-scm.com/>

Реєстрація на github

[github.com/signup/free](https://github.com/signup/free)

Графічні системи

<https://desktop.github.com/>

<https://www.git-tower.com/windows>

<https://tortoisegit.org/download/>

Документація

<https://git-scm.com/book/ru/v2>

<https://marklodato.github.io/visual-git-guide/index-ru.html?no-svg>

<http://pr0git.blogspot.com/2015/02/git_17.html>

<https://learn.javascript.ru/screencast/git>

<https://www.youtube.com/watch?v=TES0ENoIDbE>

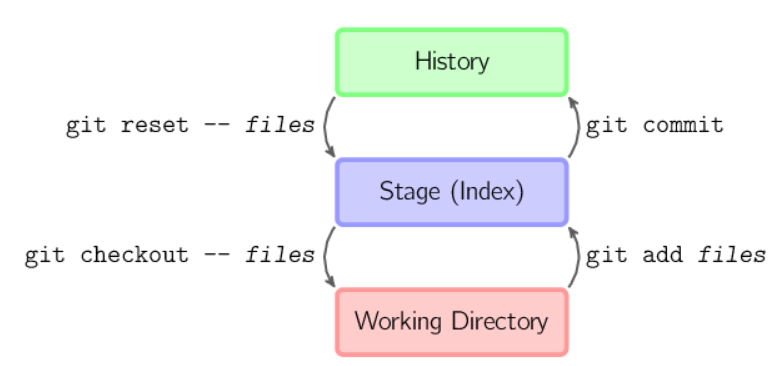
<https://www.youtube.com/watch?v=iQqDce_9y3k>

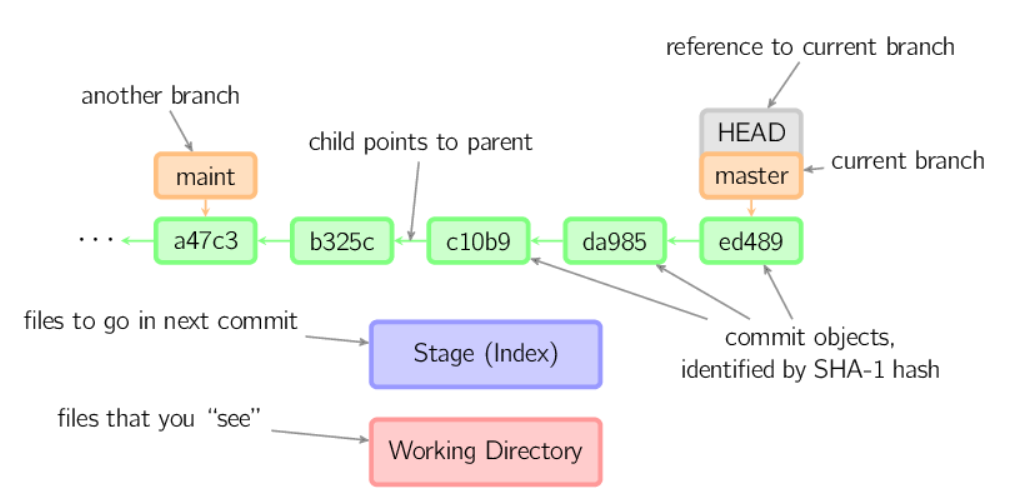
<https://www.youtube.com/watch?v=JdUzxh8miQw&list=PLuY6eeDuleIOMB2R_Kky05ZfiAx2_pbAH>

Додаткові модулі

<https://github.com/dahlbyk/posh-git>

Команди





|  |  |  |
| --- | --- | --- |
|  |  |  |
| git init | Створення |  |
| git config | Налаштування користувача  user.name  user.email | **git config --global code.editor sublime\_text** |
| РІВНІ  Системний  Глобальний  --global  Локальний  --local |
| git help | Отримання довідки |  |
| git status | Відслідковування стану (локальної копії і Stage\_area) |  |
| **Додавання змін** | | |
|  | | |
| git add <файл>  git add <шаблон>  git add -r <папка> | додавання до етапу Stage (індексування файлів) |  |
| Усі файли з поточної директорії  git add .  Усі файли з поточної директорії і піддиректорій  git add --all |  |  |
| git commit -m <повідомлення> | збережння змін на зовнішньому ресурсі | ================================================================== |
| Додавання усіх індексованих файлів і збереження  git commit -a -m "comment"  git commit -am <повідомлення> | додавання усіх вже індексованих файлів і збереження у віддаленому репозитарії |  |
| ПРИКЛАД | | |
| Додаємо файл |  |  |
|  | | |
|  |  |  |
| git rm <файл>  Видалення з Stages  git rm --cached <файл> | Видалення | git rm --cached readme |
|  |  |  |
| git clone <шлях до репозитарію>  git clone  <шлях до репозитарію>  <шлях, де зберегти> | клонування репозитарію | git clone https://github.com/myuser/project.git myfolder |
|  |  |  |
|  |  |  |
| **Перегляд комітів** | | |
| git log  Коміт - один рядок  git log --oneline | перегляд усіх комітів |  |
| git show  git show  --name-only  (  допомога – h  вийти - q  ) | перегляд змін останнього коміта |  |
| **ПОРІВНЯННЯ комітів** | | |
| git diff | Порівняння робочої директорії і Stage\_area | git diff |
| git diff --staged | Порівняння Stage\_area і останнього коміта | git diff –staged |
| git diff **коміт1** **коміт2** | Порівняння двох комітів | git diff master branchB |
|  |  |  |
|  |  |  |
| git commit --amend | Зміна (перебудова) останнього коміта (зміни з індексоаної області будуть додані у останній коміт) | git commit --amend  -----------------------------  git commit -m 'initial commit'  git add forgotten\_file  git commit --amend |
|  | | |
| **Копіювання checkout**  **(поточна вітка не змінюється)**  Команда checkout используется для копирования файлов из истории или сцены в рабочую директорию. Также она может использоваться для переключения между ветками. | | |
| git checkout -- <file> | Зчитування файлу з Stage\_area | git checkout -- CONTRIBUTING.md  Важно понимать, что git checkout -- [file] — опасная команда. Любые изменения соответствующего файла пропадают — вы просто копируете поверх него другой файл. Ни в коем случае не используйте эту команду, если вы не убеждены, что файл вам не нужен. |
|  | | |
| git checkout HEAD <файл> | Зчитування файлу з поточного коміта |  |
| git checkout HEAD~ <файл> | Зчитування файлу з попереднього коміта | git checkout HEAD~ foo.c |
| git checkout HEAD~2 <файл> | Зчитування файлу з другого з кінця коміта | git checkout HEAD~2 foo.c |
|  | | |
| git checkout <вітка> | Зчитування файлів з вітки (можлива зміна HEAD) | git checkout maint |
|  | | |
| git checkout <вітка>**~**<№предка> | Зчитування файлів з предків за порядковим номером (можлива зміна HEAD) | git checkout master~3 |
|  | | |
|  |  |  |
| **Відновлення з копії reset**  **(покажчик поточної вітки змінюється)** | | |
| git reset | Відновлення стану вітки з копії (за замовчуванням HEAD) | git reset  -----------------------------------  git reset --hard |
|  | | |
|  |  |  |
| Если коммит указан без имён файлов, указатель ветки будет перемещён на этот коммит, а затем сцена приведётся в соответствие с этим коммитом. Если мы используем ключ --soft, то сцена не будет изменена. Если мы используем ключ --hard, то будет обновлена и сцена, и рабочая директория.  =================================================================================================== | | |
|  | | |
| git reset -- <файли> | Відновлення файлів за іменем у Stage\_area | git reset -- Fist.txt |
| Если в команде указано имя файла (и/или ключ -p), то команда работает так же, как [checkout](https://marklodato.github.io/visual-git-guide/index-ru.html?no-svg" \l "checkout) с именем файла, за исключением того, что только сцена (но не рабочая директория) будет изменена. Если вы подставите имя коммита на место двойной черты, вы сможете получить состояние файла из этого коммита, тогда как в случае с двойной чертой вы получите состояние файла из коммита, на который указывает HEAD. | | |
| **РОБОТА З ВІТКАМИ** | | |
| git branch нова\_вітка  git checkout нова\_вітка  або  git checkout -b <нова\_вітка> | Створення вітки у позиції HEAD (вітка одразу стає активною) |  |
| git checkout <вітка> | Переключення на іншу вітку |  |
| Git branch -m нове\_ім»я | Перейменування вітки | git branch -m other |
| git branch -D <вітка> | Видалення вітки | git branch -D testing |
| git branch | Виведення віток | git branch  iss53  \* master  testing |
| git push --set-upstream origin вітка | Публікація вітки | git push --set-upstream origin testing |
| ОБ’ЄДНАННЯ merge | | |
| git merge <коміт> | Перемотка (присвоєння) покажчика предка до нащадка | Текущий коммит **является предком другого** коммита: в этом случае происходит быстрая перемотка (fast-forward). Ссылка текущей ветки будет просто перемещена на новый коммит, а сцена и рабочая директория будут приведены в соответствие с новым коммитом. |
|  | Об’єднання поточної вітки (поточний коміт не є предком) з іншою | Во всех других случаях выполняется «настоящее» слияние. Вы можете изменить стратегию слияния, но по умолчанию будет выполнено «рекурсивное» слияние, для которого будет взят текущий коммит (ed489 ниже на схеме), другой коммит (33104) и их общий предок (b325c); и для этих трех коммитов будет выполнено трёхстороннее слияние. Результат этого слияния будет записан в рабочую директорию и на сцену, и будет добавлен результирующий коммит со вторым родителем (33104). |
| git merge --abort | Відміна  abort |  |
| РОБОТА З МІТКАМИ | | |
| git tag | перегляд міток | git tag  v0.1  v1.3 |
| git tag -a <мітка> -m 'коментар' | створення міток анотованих | git tag -a v1.4 -m 'my version 1.4'  ---------------------------  git tag  v0.1  v1.3  v1.4 |
| git tag <мітка> | створення міток неанотованих | git tag v1.4-lw  --------------------------------------  $ git tag  v0.1  v1.3  v1.4  v1.4-lw  v1.5 |
| git show <мітка> | перегляд властивостей мітки | **git show** v1.4  tag v1.4  Tagger: Ben Straub <ben@straub.cc>  Date: Sat May 3 20:19:12 2014 -0700  my version 1.4  commit ca82a6dff817ec66f44342007202690a93763949  Author: Scott Chacon <schacon@gee-mail.com>  Date: Mon Mar 17 21:52:11 2008 -0700  changed the version numbe |
| git push origin <мітка> | збереження міток | git push origin v1.5 |
|  | збереження усіх міток | git push origin --tags |
| git checkout <мітка> | перехід до мітки | git checkout v2.0.0 |
| **git checkout -b <нова\_вітка> <мітка>** | створення вітки на основі мітки | В Git вы можете переключится на версию метки. То есть скопировать все файлы отмеченного меткой коммита в рабочую директорию и создать из этой метки новую ветку.  **git checkout -b version2 v2.0.0** Switched to a new branch 'version2' |
| Застосування деяких змін іншого коміта до поточного коміта cherry-pick | | |
|  |  | Команда cherry-pick («вишенка в тортике») создаёт новый коммит на основе только одного сладкого «коммита-вишенки», применив все его изменения и сообщение. |
| Зміна основи поточної вітки rebase  (основа – базовий коміт поточної вітки (предок самого першого коміта, як окремої вітки (на мал. нижче a47c3 ) | | |
| git rebase <базовий\_коміт> |  | =========================================================================================    Перебазирование (rebase) — это альтернатива слиянию для задач объединения нескольких веток. Если слияние создаёт новый коммит с двумя родителями, оставляя нелинейную историю, то перебазирование применяет все коммиты один за одним из одной ветки в другую, оставляя за собой линейную историю коммитов. По сути это автоматическое выполнение нескольких команд cherry-pick подряд.  На схеме вы видите как команда берёт все коммиты, которые есть в ветке topic, но отсутствуют в ветке master (коммиты 169a6 and 2c33a), и воспроизводит их в ветке master. Затем указатель ветки перемещается на новое место. Следует заметить, что старые коммиты будут уничтожены сборщиком мусора, если на них уже ничего не будет ссылаться. |
| git rebase –onto <цільовий\_базовий\_коміт> <почтаковий\_коміт\_поточної> | Перенесення чистини вітки | Используйте ключ --onto чтобы ограничить глубину захвата объединяемой ветки. На следующей схеме вы можете увидеть как в ветку master приходят лишь последние коммиты из текущей ветки, а именно коммиты после (но не включая) 169a6, т. е. 2c33a. |
| git rebase --interactive <базовий\_коміт> | Настроювана зміна основи можливістю вибору потрібних комітів та їх порядку | Есть также интерактивный режим перебазирования git rebase --interactive, с помощью которого вы сможете сделать вещи похитрее простого линейного применения коммитов, а именно сбрасывание (dropping), изменение порядка (reordering), правка (modifying) и выдавливание (squashing) коммитов. Нет наглядной схемы, чтобы показать эти возможности; за описанием лучше обратиться к справке по [git-rebase(1)](http://www.kernel.org/pub/software/scm/git/docs/git-rebase.html" \l "_interactive_mode) |
|  |  |  |
| git clean -df | видалення усіх файлів, які не збережені у репозитарії |  |

Колективна розробка

<https://habr.com/post/125999/>

|  |  |  |
| --- | --- | --- |
| git clone <репозитарій> | створюємо копію репозитарію  (клонований репозитарій має прив’язку до віддаленого репозитарію origin,  яка посилається на копію, а не оригінальний репозитарій) | git clone [git@github.com:username/Spoon-Knife.git](mailto:git@github.com:username/Spoon-Knife.git)  Для відстеження змін у оригінальному репозитації  cd Spoon-Knife  git remote add upstream git://github.com/octocat/Spoon-Knife.git  git fetch upstream |
| git checkout -b <вітка> | створюємо тематичну вітку і робимо її активною | git checkout -b feature |
| git push origin <вітка> | зберігаємо зміни у вітці у своїй копії репозитарію | git push origin feature |
| git pull | завантаження з репозитарію  (локальні файли не змінені) |  |
| git fetch --all git reset --hard github/master | завантаження з репозитацію  (локальні файли могли бути змінені) |  |
| git push | запис змінених файлів у репозитарій |  |
| git push -f | запис у репозитарій, якщо версія у локальній копії відрізняється (НЕ РОБИТИ ТАК!) |  |
| git stash | тимчасове збереження змінених файлів (хочемо попрацювати з копією з репозитарію) |  |
| git stash pop | повртаємо змінені файли, які раніше були збережені з використанням команди  git stash |  |