Шпаргалка по Git, в якій представлені основні команди

cinex.ho.ua/shpargalka-po-git-v-yakij-predstavleni-osnovni-komandy

Cinex 16 квітня 2019 р.

Git сьогодні - це дуже популярна система контролю версій. Тому шпаргалка по Git, що складається з основних команд - це те, що може вам стати в нагоді.

Шпаргалка по основним командам

git add

Команда *git add* додає вміст робочої директорії в індекс (staging area) для подальшого комітів. За замовчуванням *git commit* використовує лише цей індекс, так що ви можете використовувати *git add* для збірки зліпка вашого наступного комітів.

git status

Команда *git status* показує стан файлів в робочій директорії і індексі: які файли змінені, але не додано в індекс; які очікують комітів в індексі. Додатково до цього виводяться підказки про те, як змінити стан файлів.

git diff

Команда *git diff* використовується для обчислення різниці між будь-якими двома Git деревами. Це може бути різниця між вашою робочою директорією і індексом (власне *git diff*), різниця між індексом і останнім комітом (*git diff --staged*), або між будь-якими двома комітами (*git diff* master branchB).

git difftool

Команда *git difftool* просто запускає зовнішню утиліту порівняння для показу відмінностей в двох деревах, на випадок якщо ви хочете використовувати що-небудь відмінне від вбудованого переглядача *git diff*.

git commit

Команда *git commit* бере всі дані, додані в індекс за допомогою *git add*, і зберігає їх зліпок у внутрішній базі даних, а потім зрушує покажчик поточної гілки на цей зліпок.

git reset

Команда *git reset*, як можна здогадатися з назви, використовується в основному для скасування змін. Вона змінює покажчик **HEAD** і, опціонально, стан індексу. Також ця команда може змінити файли в робочій директорії при використанні параметра **--hard**, що може привести до втрати напрацювань при неправильному використанні, так що переконаєтеся в серйозності своїх намірів перш ніж використовувати його.

git rm

Команда *git rm* використовується в Git для видалення файлів з індексу і робочої директорії. Вона схожа на *git add* з тим лише винятком, що вона видаляє, а не додає файли для наступного комітів.

git mv

Команда *git mv* - це всього лише зручний спосіб перемістити файл, а потім виконати *git add* для нового файлу і *git rm* для старого.

git clean

Команда *git clean* використовується для видалення сміття з робочої директорії. Це можуть бути результати збирання проекту або файли конфліктів злиття.

Шпаргалка по розгалуженню і злитті

git branch

Команда *git branch* - це свого роду "менеджер гілок". Вон вміє перераховувати ваші гілки, створювати нові, видаляти і перейменовувати їх.

git checkout

Команда *git checkout* використовується для перемикання гілок і вивантаження їх вмісту в робочу директорію.

git merge

Команда *git merge* використовується для злиття однієї або декількох гілок в поточну. Потім вона встановлює покажчик поточної гілки на результуючий коміт.

git mergetool

Команда *git mergetool* просто викликає зовнішню програму злиттів, в разі якщо у вас виникли проблеми злиття.

git log

Команда *git log* використовується для перегляду історії комітів, починаючи з самого свіжого та йдучи до витоків проекту. За замовчуванням, вона показує лише історію поточної гілки, але може бути налаштована на висновок історії інших, навіть декількох відразу, гілок. Також її можна використовувати для перегляду відмінностей між гілками на рівні комітіов.

git stash

Команда *git stash* використовується для тимчасового збереження всіх незакомміченних змін для очищення робочої директорії без необхідності комітити незавершену роботу в нову гілку.

git tag

Команда *git tag* використовується для завдання постійної мітки на будь-який момент в історії проекту. Зазвичай вона використовується для релізів.

Шпаргалка по спільній роботі і оновленню проектів

Не так вже й багато команд в Git вимагають мережевого підключення для своєї роботи, практично всі команди оперують з локальною копією проекту. Коли ви готові поділитися своїми напрацюваннями, всього кілька команд допоможуть вам працювати з віддаленими репозиторіями.

git fetch

Команда *git fetch* зв'язується з віддаленим репозиторієм і забирає з нього всі зміни, яких у вас поки немає і зберігає їх локально.

git pull

Команда *git pull* працює, як комбінація команд *git fetch* і *git merge*, тобто Git спочатку забирає зміни із зазначеного віддаленого сховища, а потім намагається злити їх з поточної гілкою.

git push

Команда *git push* використовується для встановлення зв'язку з віддаленим репозиторієм, обчислення локальних змін відсутніх в ньому, і власне їх передачі в вищезгаданий репозиторій. Цій команді потрібно право запису до головного сховища, тому вона використовує аутентифікацію.

git remote

Команда *git remote* служить для управління списком віддалених репозиторіїв. Вона дозволяє зберігати довгі URL репозиторіїв у вигляді зрозумілих коротких рядків, наприклад «origin», так що вам не доведеться забивати голову всякою нісенітницею і набирати її кожен раз для зв'язку з сервером. Ви можете використовувати кілька віддалених репозиторіїв для роботи і *git remote* допоможе додавати, змінювати і видаляти їх.

git archive

Команда *git archive* використовується для упаковки в архів зазначених комітів або всього сховища.

git submodule

Команда *git submodule* використовується для управління вкладеними репозиторіями. Наприклад, це можуть бути бібліотеки або інші, які використовуються не тільки в цьому проекті ресурси. У команди *submodule* є кілька під-команд - *add, update*, *sync* і ін. - для управління такими репозиторіями.

Шпаргалка з огляду та порівняно

git show

Команда git show відображає об'єкт в простому і зрозумілому вигляді для людини. Зазвичай вона використовується для перегляду інформації про мітку або коміт.

git shortlog

Команда *git shortlog* служить для підведення підсумків команди *git log*. Вона приймає практично ті ж параметри, що і *git log*, але замість простого лістингу всіх коммітов, вони будуть згруповані по автору.

git describe

Команда *git describe* приймає на вхід що завгодно, що можна трактувати як Комміт (гілку, тег) і виводить більш-менш зрозумілий для людини рядок, який не зміниться в майбутньому для даного коміта. Це може бути використано як більш зручна, але як і раніше унікальна, заміна SHA-1.

Шпаргалка з налагодження

У Git є кілька команд, які використовуються для знаходження проблем в коді. Це команди для пошуку місця в історії, де проблема вперше проявилася і власне винуватця цієї проблеми.

git bisect

Команда *git bisect* - це надзвичайно корисна утиліта для пошуку комітів в якому вперше проявився баг або проблема з допомогою автоматичного бінарного пошуку.

git blame

Команда *git blame* виводить перед кожним рядком файлу SHA-1 коміту, останній раз мінявшого цей рядок і автора цього коміта. Це допомагає в пошуках людини, яка хоче задавати питання про проблемний шматку коду.

git grep

Команда *git grep* використовується для пошуку будь-якого рядка або регулярного виразу в будь-якому з файлів вашого проекту, навіть в більш ранніх його версіях.

Якщо ви тільки починаєте працювати з Git, або переходите на Git з іншого VCS, то така шпаргалка може вам дуже стати в нагоді.

Оригінал на: proglib.io