

# Команды для Git (v.2.0)

## Установка и настройка

### Базовые (основные) команды

#### Группа "Основные команды"

[st](#)  
[br](#)  
[co](#)  
[ci](#)  
[rb, rbi](#)  
[ch](#)  
[unstage](#)  
[amend](#)

#### Группа "Просмотр изменений"

[df](#)  
[df0](#)  
[dfc](#)  
[visual](#)

#### Группа "Просмотр истории и логов"

[sshov](#)  
[hist](#)  
[last](#)  
[hist2](#)

#### Группа "Информация о пользователе"

[myname](#)  
[myemail](#)  
[mynickname](#)  
[mycheck](#)  
[mylog, mylogsort](#)

#### Useful commands

[show-current-branch-name, cbr](#)  
[ign, ign-cd](#)  
[ignore, ignored, unignore](#)  
[merged, unmerged](#)  
[gi, gi-list](#)  
[fsckclear](#)

### Расширенные команды

#### Обозначения и пояснения

#### Часто используемые расширенные команды

[w-create-base, mcf-create-base](#)  
[w-rebuild-base, mcf-rebuild-base](#)  
[w-update, mcf-update](#)

[w-update2, mcf-update2](#)

[w-upload, mcf-upload](#)

[w-upload2, mcf-upload2](#)

[w-copy2tmp, mcf-copy2tmp](#)

[Вспомогательные расширенные команды](#)

[w-load-fix-from-repo, mcf-load-fix-from-repo](#)

[w-apply-fix, mcf-apply-fix](#)

[Конечная ветка: \\$fix.](#)

[w-send-fix, mcf-send-fix](#)

[w-backup-cfg, mcf-backup-cfg](#)

[w-fakecommit, mcf-fakecommit](#)

[w-update-extcmd, mcf-update-extcmd](#)

[Локальные переменные](#)

[Параметры MCF-схемы](#)

[Имена веток MCF-схемы](#)

[I-debug-level](#)

[Применение расширенных команд \(cookbook\)](#)

[Основная идея](#)

[Общий цикл разработки](#)

[Первичная настройка и настройка локальной конфигурации](#)

[Регулярная работа](#)

[Периодические работы](#)

[Различные примеры при разработке](#)

[Генерация тестового окружения](#)

[Первичное наполнение](#)

[Настройка локальной конфигурации](#)

[Работа внутри команды](#)

[Работа с дополнительным источником](#)

[Пример из реальной задачи](#)

[Предварительная настройка](#)

[Настройка веток разработки](#)

[Стандартная разработка](#)

[Разработка](#)

[Отправка результатов работы](#)

[Контрибуция](#)

[Рекомендации по устранению конфликтов при обновлении и загрузке](#)

[Стандартное решение](#)

[Другие способы](#)

[Дополнение](#)



## *Вступление*

Каждый разработчик, который использует git использует определенный пользовательский набор команд - алиасы и макросы.

Ниже будет приведен список команд, которые помогают в нашей работе. Они проверены на многих проектах. Основным положительным моментом, что при использовании этих команд используется единая схема работы с ветками для различных конфигураций схем работы с заказчиком, а также уменьшается количество ошибок при синхронизации и устранении конфликтов.

Команды разбиты по группам, все расширенные команды mcf-\* имеют дополнительные алиасы w-\*, чтобы можно было изменить поведение команд mcf-\* в локальном конфиге проекта не изменяя общее поведение, а также константы l-\*.

Многие команды из раздела "общие" - это просто алиасы стандартных команд, сделаны просто для удобства и по аналогии с SVN, а также чтобы было удобнее и быстрее их набирать.

## Установка и настройка

Естественно git 1.7.10+ должен быть уже установлен. Тестирование команд проводилась git 1.9.5 (Win7) и git 2.2.0 (Ubuntu 12.04).

1. Склонировать репозиторий системы команд в текущий каталог с нашего сервера <https://github.com/GiantLeapLab/myworkflow>. Например, в ~/mytest:

```
cd ~/mytest && git clone git@github.com:GiantLeapLab/myworkflow.git mcf
Cloning into 'mcf'...
...
Checking connectivity... done.
```

2. Входим в рабочий каталог для инсталляции проекта. Сделаем все скрипты в нем запускаемыми. Все дальнейшие действия будем описывать находясь в этом каталоге.

```
cd mcf/install && chmod 774 *.sh
```

3. Запускаем скрипт для предварительной настройки:

```
./00_prepare_install.sh
Check user info ...
Check the file '/home/user/mytest/mcf/install/my_git_name_info':
----- START CHECK -----
git config --global user.name      "your name"
git config --global user.email     "your@email.address"
git config --global user.nickname  "yournickname"
----- END CHECK -----
Edit user info:
vi "/home/user/mytest/mcf/install/my_git_name_info"
Please, edit the file '/home/user/mytest/mcf/install/my_git_name_info'
```

4. Редактируем файл с user info, вставляя корректную информацию о себе:

```
vi "my_git_name_info"
... редактируем ...
```

5. Еще раз проверяем введенную информацию:

```
./00_prepare_install.sh
...
----- START CHECK -----
git config --global user.name      "Valerii Savchenko"
git config --global user.email     "valerii.s@giantleaplab.com"
git config --global user.nickname  "wellic"
----- END CHECK -----
...
```

6. Если user info ок, генерируем скрипт инсталляции, который берет шаблоны из папки *install.template*:

```
./01_create_install.sh
...
The install program was prepared.
```

```
You have to run installation after checked user info correct:
bash "/home/user/Dropbox/github/myworkflow/install/02_install.sh"
```

7. Инсталлируем .gitconfig в систему. При установке будет создана копия существующего файла .gitconfig:

```
./02_install.sh
Installing ...
```

*Please check your new '.gitconfig':*

*less ~/.gitconfig*

8. Проверяем содержимое установленного .gitconfig:  
**less ~/.gitconfig**
9. Убираем сгенерированный каталог, чтобы не занимал место, т.к. для повторной инсталляции достаточно выполнить только шаги 6 и 7:  
**./99\_clear.sh**

## Базовые (основные) команды

### Группа "Основные команды"

**st**

Алиас: status

**br**

Алиас: branch

**co**

Алиас: checkout

**ci**

Алиас: commit

**rb, rbi**

Алиас: rebase и rebase -i

**ch**

Алиас: cherry-pick

**unstage**

Алиас: reset HEAD --

**amend**

Базируется на commit. Используется, когда нужно в предыдущий коммит добавить изменения, чтобы сохранился заголовок коммита.

**#Добавить незакомиченные изменения в последний коммит на текущей ветке**

**git amend -a**

### Группа "Просмотр изменений"

**df**

Алиас: diff

**df0**

Алиас: diff -U0

Делает тоже, что и diff, но показывает только измененные строки.

**dfc**

Алиас: diff --cached

Делает тоже, что и diff, но берет файлы не из рабочей папки, а из stage area.

## visual

Алиас: `gitk --all &`

## Группа "Просмотр истории и логов"

### sshow

Базируется на команды *show*. Команда позволяет посмотреть какие файлы менялись в указанном коммите. Удобно находить, какие файлы были использованы в указанном коммите.

```
>git sshow ae39542
* ae39542 2015-02-13 | Improved config [Valerii Savchenko]|
| .gitconfig_sample          | 106 ++-
| tools/fix_win.bat          |   2 +-
| tools/setup_git_testing_env.sh |   2 +-
| 3 files changed, 53 insertions(+), 57 deletions(-)
```

### hist

Базируется на команде *log*. Команда позволяет в удобной форме посмотреть список коммитов. Удобно использовать вместе с именем ветки и количеством коммитов для просмотра. Выводит по-умолчанию все записи.

```
>git hist
* 14ce503 2015-02-16 (HEAD, origin/master, master) | Added config #2 [wellic]
* a5bf1fe 2015-02-16 | Fixed create-base command [Valerii Savchenko]
... commits ...
* 87363d5 2014-12-03 | Add workflow [Valerii Savchenko]
* c50b514 2014-12-03 | Initial commit [wellic]
```

### last

Базируется на команде *hist*, но выводит, по-умолчанию, последние 20 коммитов. Удобно использовать вместе с именем ветки и количеством коммитов для просмотра.

```
>git last master -2
* 14ce503 2015-02-16 (HEAD, origin/master, master) | Added config #2 [wellic]
* fba755e 2015-02-16 (gll/master, gh/master) | Added config [Valerii Savchenko]
```

### hist2

Базируется на команде *hist*, но выводит, список коммитов без показа имен веток, тегов и т.д.

```
>git hist2
* 14ce503 2015-02-16 | Added config #2 [wellic]
... commits ...
* c50b514 2014-12-03 | Initial commit [wellic]
```



## Группа "Информация о пользователе"

### myname

Показывает имя текущего пользователя. Использует переменную `user.name`, Можно переопределять в локальном конфиге.

```
>git myname
Valerii Savchenko
>vim /dir/to/project/.git/config
#Change default name
[user]
name = wellic
>git myname
wellic
```

### myemail

Показывает email текущего пользователя. Использует переменную `user.email`. Можно переопределять в локальном конфиге.

```
>git myemail
valerii.s@giantleaplab.com
>vim /dir/to/project/.git/config
#Change default email
[user]
email = new@address
>git myemail
new@address.com
```

### mynickname

Показывает nickname текущего пользователя. Использует переменную `user.nickname`. Можно переопределять в локальном конфиге.

```
>git mynickname
wellic
>vim /dir/to/project/.git/config
#Change default nickname
[user]
nickname = wellic2
>git mynickname
wellic2
```

### mycheck

Базируется на команде `git grep`. Команда проверяет на наличие в исходниках наличия `nickname`, по-умолчанию, но можно задавать и свое выражение. Например, удобно в отладочных целях поставить в исходниках метки `'#DEBUG your_nickname'`, и проверить, чтобы перед коммитом не забыть убрать не нужные метки. Выводит имена файлов и номера строк, где есть метки.

```
>git mycheck '#DEBUG'
Find '#DEBUG' in sources
```

```
README.md:9:if ($debug) echo '1'; #DEBUG wellic
```

## **mylog, mylogsort**

Базируются на hist2. Показывает коммиты текущего пользователя, как есть или отсортированные по дате.

## **Useful commands**

### **show-current-branch-name, cbr**

Показывает имя текущей ветки. Полезно для написания скриптов.

### **ign, ign-cd**

Показывает список файлов, которые будет игнорироваться при коммите полный или для текущего каталога.

### **ignore, ignored, unignore**

Добавление, просмотр, удаление файлов, которые во временном списке игнорирования. Может быть полезным, чтобы не добавлять в .gitignore

### **merged, unmerged**

Базируется на команде branch. Показывает сmerged и несmerged ветки. Можно указывать конкретное имя ветки. Полезно, если нужно удалить уже сmerged ветки.

### **gi, gi-list**

Получить с сайта gitignore.io список существующих правил и получить правила. Используется для заполнения .gitignore стандартными правилами

```
>git gi-list
```

```
...,bower,...,laravel,linux,...,netbeans,...,symfony,...,vim,...,wordpress,...yii...
```

```
>git gi yii
```

```
### Yii ###
```

```
assets/*
```

```
!assets/.gitignore
```

```
protected/runtime/*
```

```
!protected/runtime/.gitignore
```

```
protected/data/*.db
```

```
themes/classic/views/
```

### **fsckclear**

Чистит, лечит, убирает коммиты на которые нет ссылок, сжимает локальную репу. Использовать только профилактически, желательно не часто.

# Расширенные команды

## Обозначения и пояснения

1. Все команды имеют основную версию команды (mcf-\*) и дополнительную с префиксом (w-\*). Эти команды равнозначны, пока они не переопределены в локальном конфиге. Если необходимо, то переопределять лучше команды w-\*, т.к. именно эти команды вызываются в расширенных командах. Команды **mcf-\*** нужно использовать как команды с поведением по-умолчанию.
2. Расширенные команды выполняют работы с git согласно 3-веточной схемы. Все команды можно переопределять в локальных конфигах, чтобы изменить стандартное поведение.

**# П р и м е р**

**# Отключить бекап локального конфига (ветка \$cfg)**  
**на удаленный сервер**  
**# для команд w-upload и w-upload2.**  
**>vim /dir/to/project/.git/config**

...

**[alias]**

**w-backup-cfg= "! : "**

3. При описании команд ниже запись \$param = value обозначает, что \$param с параметром по-умолчанию value.
4. Все команды используют параметры по-умолчанию:

**\$fix = l-fix** - имя ветки правок fix в схеме MCF

**\$cfg = l-cfg** - имя ветки конфигурации cfg в схеме MCF

**\$master = l-master** - имя основной ветки master в схеме MCF

**\$src1 = l-src1** - имя основного источника

**\$src1\_rbranch = l-src1-rbranch** - имя удаленной (remote) ветки в основном источнике

**\$src2 = l-src2** - имя вспомогательного источника

**\$src2\_rbranch = l-src2-rbranch** - имя удаленной (remote) ветки вспомогательного источника.

**\$showlog = show** - показывать или не показывать (hide|off) log после выполнения команды

**\$rebuild\_base = on** - делать(on) или не ltkfm(off) после выполнения команды переустановку базовых веток

**\$method = rebase** - метод (rebase|merge), который использовать при получении новых изменений с удаленных (remote) источников

5. Локальные константы l-\*, которые находятся в секции [mcf], предназначены, для инициализации параметров команд по-умолчанию, а также для переопределения значений параметров в локальных конфигах.

**# П р и м е р**

```

#Изменить умолчательные имя локальных веток
$master в work, $fix в dev
# для текущего проекта.
# vim /dir/to/project/.git/config.
# ...
# [mcf]
#     l-master = work
#     l-fix    = dev

```

6. Все *mcf,w*-\* команды можно адаптировать для работы с git-svn.
7. Далее в примерах мы используем команды с индексом *w*\*, и они могут быть переопределены в локальных конфигах, так же удобнее и быстрее набирать на клавиатуре.

## Часто используемые расширенные команды

### w-create-base, mcf-create-base

**w-create-base [ \$fix \$cfg \$master \$showlog ]**

Создать ветки \$cfg с базой от \$master и \$fix с базой от \$cfg для работы со схемой MCF. Если ветки уже были созданы ранее, то будет просто переход на них без изменения базовых коммитов.

Конечная ветка: \$fix.

### w-rebuild-base, mcf-rebuild-base

**w-rebuild-base [ \$fix \$cfg \$master \$showlog ]**

Переустановить базовые коммиты на \$master для \$cfg и yf и \$cfg для \$fix, т.е. подготовить для работы со схемой MCF.

Обычно делается всегда после правки конфигурации, или после обновления \$master, чтобы внести изменения в базовые ветки.

Конечная ветка: \$fix.

### w-update, mcf-update

**w-update [ \$fix \$cfg \$master \$src1 \$src1\_rbranch \$src2 \$src2\_rbranch \$rebuild\_base \$showlog ]**

Залить обновление в \$master из 2-х источников.

Обновить \$master из основного источника \$src1/\$src1\_rbranch методом *git pull --rebase*.

Если первый и второй источник одинаковые (\$src1 = \$src2 && \$src1\_rbranch = \$src2\_rbranch), то обновление из второго источника пропускается.

Если первый и второй источник разные, то обновить \$master из дополнительного источника \$src2/\$src2\_rbranch методом *git merge*.

Параметр \$rebuild\_base=on|off указывает, нужно ли после обновления делать переустановку базовых веток.

Конечная ветка: \$fix.

### w-update2, mcf-update2

**w-update2 [ \$src2 \$src2\_rbranch \$fix \$cfg \$master \$src1 \$src1\_rbranch \$rebuild\_base \$showlog ]**

Команда аналогична w-update, но другой порядок параметров.

Первые два параметра определяют второй источник, потом указываются остальные параметры или берутся по-умолчанию, как для w-update.

Более удобна при работе с 2-мя источниками, чем w-update, если используются параметры по-умолчанию, а нужно указать только параметры второго источника.

Конечная ветка: \$fix.

### **w-upload, mcf-upload**

**w-upload [ \$fix \$cfg \$master \$src1 \$src1\_rbranch \$src2 \$src2\_rbranch \$showlog ]**

Выгружает сделанные изменения из ветки \$fix в ветку \$master и на сервер \$src1.

Сначала происходит обновление \$master из источников \$src1 и \$src2 (см. w-update).

Затем изменения из \$fix заливаются в \$master и отправляются в \$src1/\$src1\_rbranch.

Конечная ветка: \$fix.

### **w-upload2, mcf-upload2**

**w-upload2 [ \$src2 \$src2\_rbranch \$fix \$cfg \$master \$src1 \$src1\_rbranch \$showlog ]**

Команда аналогична w-upload, но другой порядок параметров.

Первые два параметра определяют второй источник, потом указываются остальные параметры или берутся по-умолчанию, как для w-upload.

Более удобна при работе с 2-мя источниками, чем w-upload, если используются параметры по-умолчанию, а нужно указать только параметры второго источника.

Конечная ветка: \$fix.

### **w-copy2tmp, mcf-copy2tmp**

**w-copy2tmp [ \$src1 \$showlog ]**

Сделать временный бекап состояния текущей ветки.

Создается текущее состояние рабочего каталога в репу \$src1 с именем ветки

'\_{nickname}\_{name\_of\_current\_branch}\_tmp'. При бекапе все несохраненные изменения также отправятся в репу.

После завершения бекапа в рабочем каталоге состояние файлов будет восстановлено, как до выполнения команды.

Эта операция полезна, если вы не готовы коммитить изменения, но нужно сделать копию текущего состояния.

Конечная ветка: не меняется.

## **Вспомогательные расширенные команды**

### **w-load-fix-from-repo, mcf-load-fix-from-repo**

**w-load-fix-from-repo [ \$master \$src1 \$src1\_rbranch \$method \$showlog ]**

Залить новые изменения в ветку \$master из источника \$src1 \$src1\_rbranch используя \$method=rebase|merge.

Если \$method=rebase, то используется *pull --rebase*, а иначе merge.

Конечная ветка: \$master.

### **w-apply-fix, mcf-apply-fix**

**w-apply-fix [ \$fix \$cfg \$master \$showlog ]**

Загрузить свои изменения из ветки \$fix в ветку \$master, исключая коммиты ветки \$cfg.  
Конечная ветка: \$fix.

### **w-send-fix, mcf-send-fix**

**w-send-fix [ \$fix \$cfg \$master \$src1 \$src1\_rbranch \$showlog ]**

Применить изменения (mcf-apply-fix) \$fix к \$master, и отправить их в \$src1/\$src1\_rbranch.  
Конечная ветка: \$fix.

### **w-backup-cfg, mcf-backup-cfg**

**w-backup-cfg [ \$cfg \$src1 \$showlog ]**

Сделать бекап ветки конфигурации  
Отправить копию ветки \$cfg в репу \$src1 с именем ветки '\_{nickname}\_{\$cfg}\_backup'.  
Конечная ветка: не меняется.

### **w-fakecommit, mcf-fakecommit**

**w-fakecommit [ \$mess \$showlog ]**

Создает фейковый коммит.  
Создается коммит с пустым файлом и комментарием 'Fake: \$mess'. По-умолчанию, комментарий - "Fake: Added \_fakefile\_...".  
Команда используется для тестов и при исследовании git команд.  
Конечная ветка: не меняется.

### **w-update-extcmd, mcf-update-extcmd**

**w-update-extcmd [ \$fix \$cfg \$master \$src1 \$src1\_rbranch \$src2 \$src2\_rbranch \$rebuild\_base \$showlog ]**

Эта команда выполняется после успешного выполнения обновления w-update.  
Эту команду нужно переопределить в локальном конфиге, чтобы при обновлении можно было выполнить дополнительные действия. Например, выполнить тесты, дополнительную синхронизацию, дополнительные системные программы и т.п.  
Конечная ветка: зависит от переопределенной команды.

**# П р и м е р : д о п о л н и т е л ь н а я с и н х р о н и з а ц и я с 3-м сервером server3**

```
>vim /dir/to/project/.git/config
...
[alias]
w-update-extcmd-default = "! f(){ \
    fix=$1 && cfg=$2 && master=$3\
    && git push server3 $master:custom_branch \
};}; f"
```

## **Локальные переменные**

Локальные переменные l-\* описываются в секции *[mcf]* файла конфигурации.

Они задают умолчательные параметры команд и уровень детализации отладочной информации.

#### Параметры MCF-схемы

```
l-fix = fix
l-cfg = cfg
l-master = master
```

#### Основной источник обновлений

```
l-src1 = origin
l-src1-rbranch = master
```

#### Дополнительный источник обновлений

```
l-src2          = origin
l-src2-rbranch = master
```

#### Отладка

##### l-debug-level

```
l-debug-level = "! echo $level"
```

```
$level : -1 is normal mode. Hide all diagnostic messages.
```

```
$level : 0 is debug mode 1. Show some diagnostic message in w-* functions.
```

```
$level : 1 is debug mode 2. Show all diagnostic message in w-* functions.
```

Локальный метод, позволяет управлять выводом сообщений при работе w-\* методов. Необходим при отладке методов и при изучении работы команд.

# Применение расширенных команд (cookbook)

## Основная идея

В пакете с командами идет небольшая утилита для изучения и применения команд. Часто необходимо промоделировать некоторые ситуации, возникшие в ходе работ. Чтобы не испортить рабочую копию, можно потренироваться на тестовой. Далее рассмотрим набор основных действий и пример использования некоторых команд `w-*`.

В качестве примера рассмотрим ситуацию когда есть 2 группы разработчиков:

- 1 группа - репозиторий `server1`, разработчики `dev1` и `dev2`.
- 2 группа - репозиторий `server2`, разработчик `dev3`.

В примере, для упрощения, мы рассмотрим работу по схеме MCF только разработчика `dev2`.

Его работа с репозиторием сводится к одной или двум командам:

- `git w-upload` - если он работает только со своей командой (`w-upload` позволяет работать одновременно с 2-мя источниками, см. справочник).
- `git w-wupload2` - если нужно внести код второй команды в свои исходники (например, это код некоторой деvelopepской версии CMS).

## Общий цикл разработки

Обычно, весь цикл выглядит приблизительно так:

### Первичная настройка и настройка локальной конфигурации

- Первичная настройка делается 1-й раз:  
`git checkout master`  
`git w-create-base`  
Последняя команда создает ветки `cfg` от `master` и потом `fix` от `cfg`.
- настройка локальной конфигурации делается редко, обычно после разворачивания некоторой системы.  
`git checkout cfg`  
Настраиваем систему под себя, коммитимся  
`... Edit configs ...`  
`git add ...`  
`git commit ...`  
Переустановим базовые коммиты на рабочих ветках MCF  
`git w-rebuild-base`

### Регулярная работа

- Работаем всегда на ветке `fix`. Стандартный цикл разработки - пишем код и делаем коммиты:  
`git checkout fix;`  
`{ ... Create code ... ; git add ... ; git commit ... } много раз`



- Отправляем свой код в репозиторий, при этом произойдет подтягивание кода других разработчиков из основного источника, автоматическое перестроение веток, отправка своих изменений, бекап конфигурации и т.д. (см. справочник команд):

```
git w-upload
```

### Периодические работы

- Иногда нужно подтянуть код из стороннего источника к себе в ветку fix, но не отправлять свою работу до полного ее окончания:

```
git w-update2 source2 local_name_branch remote_name_branch
```

а если нужно отправить в свое хранилище:

```
git w-upload2 source2 local_name_branch remote_name_branch
```

### Различные примеры при разработке

Далее все диагностические и информационные сообщения будут опущены для упрощения вывода.

### Генерация тестового окружения

- Допустим у нас программа скопирована в *~/mytest/mcf*. Переходим в каталог с генератором тестового окружения и запускаем его:

```
cd ~/mytest/mcf/testenv/  
./setup_git_testing_env.sh ~/mytest  
...
```

*For testing go to:*

```
cd "/home/user/mytest/_testgit_/devs"
```

- Скрипт сгенерит набор каталогов, которые позволяют эмулировать работу нескольких серверов и нескольких разработчиков. По-умолчанию: 2 сервера и 3 разработчика. (В скрипте генерации можно изменить эти параметры.). Перейдем в каталог разработчиков. Каждая папка в этом каталоге эмулирует работу рабочего места отдельного разработчика

```
cd ~/mytest/_testgit_
```

```
cd devs
```

```
ls
```

```
dev1 dev2 dev3
```

```
cd ../servers
```

```
ls
```

```
server1 server2
```

### Первичное наполнение

- Первый разработчик dev1, начинает выполнять работу делает 3 коммита и отправляет их в совместный с разработчиком dev2 репозиторий origin (при генерации у всех разработчиков origin привязан к server1):

```
(div3): cd ../dev1
```

```
(div1/master): git remote -v
```

```
(div1/master): git w-fakecommit 'Initial dev1 c1'
```

```
(div1/master): git w-fakecommit 'dev1 c2'
```

```
(div1/master): git w-fakecommit 'dev1 c3'
```

```
(div1/master): git push -u origin master:master


```

- Второй разработчик dev2, сначала подтягивает к себе изменения, которые уже существует и включается в работу:

```
(div1): cd ../dev2


```

- Создаем базовые ветки:

```
(div2/master): git w-create-base
* efabcd1 2015-02-19 (origin/master, master, cfg, fix) | Fake: dev1 c3
...
```

## Настройка локальной конфигурации

- Настраиваем конфиг:

```
(div2/fix): git checkout cfg


```

## Работа внутри команды

- Второй разработчик делает полезную работу:

```
(div2/fix): git w-fakecommit 'dev2 fix1'


```

- Параллельно с ним работает dev1 и отправляет свои правки в хранилище:

```
(div2): cd ../dev1


```

- dev2 решил отправить свои правки в хранилище без локального конфига:

```
(div1): cd ../dev2


```

```
* bade39a 2015-02-19 (HEAD, origin/_wellic_cfg_backup, fix, cfg) | Fake: cfg2...
* 01c27a4 2015-02-19 | Fake: dev2 cfg1 [Valerii Savchenko]
* e2b8702 2015-02-19 (origin/master, master) | Fake: dev2 fix2 [Valerii...]
* 490ff3d 2015-02-19 | Fake: dev2 fix1 [Valerii Savchenko]
* e26301f 2015-02-19 | Fake: dev1 c5 [Valerii Savchenko]
* 07d741a 2015-02-19 | Fake: dev1 c4 [Valerii Savchenko]
* efabcd1 2015-02-19 | Fake: dev1 c3 [Valerii Savchenko]
...
```

- dev1 обновляет свою локальную версию:

```
(div2): cd ../dev1
```

```
(div1/master): git pull --rebase
```

```
(div1/master): git last
```

```
* e2b8702 2015-02-19 (HEAD, origin/master, master) | Fake: dev2 fix2 [Val...]
* 490ff3d 2015-02-19 | Fake: dev2 fix1 [Valerii Savchenko]
* e26301f 2015-02-19 | Fake: dev1 c5 [Valerii Savchenko]
* 07d741a 2015-02-19 | Fake: dev1 c4 [Valerii Savchenko]
* efabcd1 2015-02-19 | Fake: dev1 c3 [Valerii Savchenko]
* 3c7ea62 2015-02-19 | Fake: dev1 c2 [Valerii Savchenko]
* 6c3aae9 2015-02-19 | Fake: Initial commit [Valerii Savchenko]
```

## Работа с дополнительным источником

- dev3 работает самостоятельно и ничего не знает про первую группу разработчиков. Его код автономный. Например, библиотека, фреймворк, и т.д. Ее позже захотят использовать разработчики из первой группы dev1 и dev2. Разработчик dev3 сохраняет свои результаты на server2:

```
(div1): cd ../dev3
```

```
(div3/master): git w-fakecommit 'dev3 c1'
```

```
(div3/master): git w-fakecommit 'dev3 c2'
```

```
(div3/master): git w-fakecommit 'dev3 c3'
```

```
(div3/master): git push -u server2 master:master
```

```
(div3/master): git last
```

```
* bb77be2 2015-02-19 (HEAD, server2/master, master) | Fake: dev3 c3 [Val...]
* f8b6a14 2015-02-19 | Fake: dev3 c2 [Valerii Savchenko]
* 8d7c094 2015-02-19 | Fake: dev3 c1 [Valerii Savchenko]
...
```

- Разработчик dev1 продолжает в это время тоже работать

```
(div3): cd ../dev1
```

```
(div1/master): git pull --rebase
```

```
(div1/master): git w-fakecommit 'dev1 c6'
```

```
(div1/master): git w-fakecommit 'dev1 c7'
```

```
(div1/master): git push
```

```
(div1/master): git last
```

```
* 1e9f606 2015-02-19 (HEAD, origin/master, master) | Fake: dev1 c6 [Val...]
* b42b4ca 2015-02-19 | Fake: dev1 c6 [Valerii Savchenko]
* e2b8702 2015-02-19 | Fake: dev2 fix2 [Valerii Savchenko]
...
```

- Разработчик dev2 также работает, кроме того ему понадобилось поправить конфигурацию:

```
(div1): cd ../dev2
```

```
(div2/fix): git w-fakecommit fix3
```

```
(div2/fix): git checkout cfg
```

```
(div2/cfg): git w-fakecommit cfg3
```

```
(div2/cfg): git w-rebuild-base
```

```
* 853c6b4 2015-02-19 (HEAD, fix) | Fake: fix3 [Valerii Savchenko]
* 1922ef8 2015-02-19 (cfg) | Fake: cfg3 [Valerii Savchenko]
* bade39a 2015-02-19 | Fake: dev2 cfg2 [Valerii Savchenko]
* 01c27a4 2015-02-19 | Fake: dev2 cfg1 [Valerii Savchenko]
* e2b8702 2015-02-19 (origin/master, master) | Fake: dev2 fix2 [Val...]
...
```

- Чтобы подтянуть код из второго источника server2, разработчик dev2 создает локальную ветку master\_s2, которые учитываются при отправке его работы в свое хранилище:

```
(div2/fix): git fetch server2 master
```

```
(div2/fix): git checkout -b master_s2 server2/master
```

```
(div2/master_s2): git w-upload2 server2 master_s2 master
```

```
* 2fc40f8 2015-02-19 (HEAD, origin/_wellic_cfg_backup, fix, cfg) | Fake: cfg3 ...
* 0e475c2 2015-02-19 | Fake: dev2 cfg2 [Valerii Savchenko]
* acfc138 2015-02-19 | Fake: dev2 cfg1 [Valerii Savchenko]
* 2cfeda5 2015-02-19 (origin/master, master) | Fake: fix3 [Valerii Savchenko]
* 4f056f0 2015-02-19 | Merged with server2/master [Valerii Savchenko]
/\
| * bb77be2 2015-02-19 (server2/master, master_s2) | Fake: dev3 c3 [Va...]
| * f8b6a14 2015-02-19 | Fake: dev3 c2 [Valerii Savchenko]
| * 8d7c094 2015-02-19 | Fake: dev3 c1 [Valerii Savchenko]
* 1e9f606 2015-02-19 | Fake: dev1 c6 [Valerii Savchenko]
* b42b4ca 2015-02-19 | Fake: dev1 c6 [Valerii Savchenko]
...
```

## Пример из реальной задачи

В качестве примера возьмем CMS PrestaShop. Эту CMS мы используем в своих проектах, но также, вносим правки в нее, т.е. являемся для нее контрибьютерами. Для этого мы используем 3 репозитория:

1. presta - источник оригинального кода Prestashop
2. gll - форк оригинала PrestaShop, необходим для контрибуции
3. origin - наш репозиторий для текущего проекта.

Стандартные задачи, которые нужно выполнять регулярно:

- загружать обновления с presta;
- синхронизировать presta с gll;
- обновление локальной разработки origin обновлениями с presta;
- вносить замеченные ошибки и улучшения в PrestaShop;
- решение текущей задачи с PrestaShop;
- бекап копии конфигурации для локальной разработки.

## Предварительная настройка

Настройка репозиторий для работы:

```
#origin - our project
```

```
git clone git@gitserver.giantleaplab.com:OurProject
```

```
# presta - dev branch
git remote add presta https://github.com/PrestaShop/PrestaShop.git
```

```
# gll - fork presta
git remote add gll git@github.com:GiantLeapLab/PrestaShop.git
```

## Настройка веток разработки

Для работы используется несколько веток.

Настроим окружение.

1. '1.6' - ветка из presta, содержит последние изменения Prestashop. Эта же ветка используется в gll для дальнейшей контрибуции
2. `git fetch presta 1.6`  
`git checkout 1.6 presta/1.6`
3. 'work' - ветка в origin, порождается от 1.6, содержит наши разработки
4. `git checkout -b work`
5. 'cfg' - локальная ветка, порождается от work, содержит настройки локальной конфигурации для разработки
6. 'fix' - локальная ветка, порождается от cfg, в ней ведется разработка
7. `git w-create-base fix cfg work`
8. удаляем локальную ветку master, т.к. мы ее использовать не будем, а также чтобы она нас не запутывала
9. `git branch -D master`

## Стандартная разработка

Считаем, что все синхронизировано, все ветки созданы.

### Разработка

Работы выполняем на ветке fix:

```
git checkout fix
```

```
git commit -m 'Issue 1'
```

```
git commit -m 'Issue 2'
```

```
git commit -m 'Issue 3'
```

### Отправка результатов работы

Теперь нам нужно все перенести на ветку work и отправить в хранилище. Но за время нашей работы в presta/1.6 или в origin/work могли внести изменения. Поэтому нужно сделать достаточно много этапов для внесения новых изменений по веткам. А также синхронизировать репозитории presta и gll, т.е. нам нужно выполнить следующий список работ:

- загрузить новые обновления, которые могли сделать коллеги из origin/work в work
- загрузить новые обновления из presta/1.6 в ветки 1.6 и work
- восстановить связи cfg и fix с work после обновления
- залить изменения из ветки fix в ветку work, исключая настройки локальной конфигурации из cfg
- отправить изменения в origin
- сделать бэкап локальной конфигурации в origin/cfg\_backup

Это все может быть выполнено следующей командой:

```
git w-upload fix cfg work origin work presta 1.6 1.6 show
```

Чтобы не набирать в командной строке большое количество параметров, учитывая, что вместо ветки master используется ветка work, удобно в локальном конфиге проекта в файле .git/config внести вспомогательные алиасы:

```
vi .git/config
```

```
...
```

```
[alias]
```

```
w-update-extcmd = = "! git push gll 1.6:1.6 "
```

```
w-upload = "! f() { \
```

```
fix=${1:-fix} && cfg=${2:-cfg} && master=${3:-work} \
```

```
&& src1=${4:-origin} && rbranch1=${5:-$master} \
```

```
&& src2=${6:-gll} && lbranch2=${7:-1.6} && rbranch2=${8:-1.6} \
```

```
&& showlog=${9:-show} \
```

```
&& git mcf-upload $fix $cfg $master $src1 $rbranch1 $src2 $lbranch2 $rbranch2 $showlog \
```

```
};; f"
```

И тогда, все выше описанные действия выполняются, в дальнейшем, одной командой:

```
> git w-upload
```

## Контрибуция

После исправления ошибки, достаточно отправить в репозиторий gll ветку, и уже на сайте гитхаб сделать pull-request:

```
git checkout cfg
```

```
git checkout -b fix_issue_presta
```

```
...
```

```
git commit -m 'Fixed issue in prestashop'
```

```
git push gll fix_issue_presta:fix_issue_presta
```

Когда правку примут, то нужно удалить лишние ветки:

```
git checkout cfg
```

```
git branch -D fix_issue_presta
```

```
git push gll :fix_issue_presta
```

```
git remote prune gll
```

## Рекомендации по устранению конфликтов при обновлении и загрузке

При выполнении команд `w-update` и `w-upload`, в случае появления конфликта, нужно посмотреть на какой ветке произошел конфликт и какая команда была последней.

### Стандартное решение

Обычно применяется следующая схема:

- Посмотреть на какой операции произошел конфликт. Можно найти Видно в выводе команд.
- Устранить конфликт.  
`git mergetool`  
`...resolve...`
- Убедиться, что все ок.  
`git status`  
`git last`
- Если менялся `$master` - отправить его в собственное хранилище.  
`git push origin $master:$remote_master`
- Если команда не закончена, то закончить (`rebase`, `merge`, `pull`).  
`git rebase/pull/merge`
- Повторить начальную команду (`w-update`, `w-upload`).  
`git w-update/w-upload`

### Другие способы

Конфликты могут возникнуть между:

- 1) `master <-> cfg`, `cfg <-> fix`, `master <-> master_source2`: устраняется по стандартной схеме.
- 2) `master <-> origin/master`: такой конфликт может возникнуть, если каким-то образом на локальную ветку `$master` попали коммиты, которые конфликтуют с `origin/$remote_master`. Решений несколько: либо попробовать стандартное решение, либо найти причину появления лишних коммитов на ветке `$master`, если они существуют на другой ветке:
  - удалить с этой ветки  
`git reset --hard SHA1_FIRST_BAD_COMMIT`
  - повторить начальную команду:  
`git w-update/w-upload`
- 3) `fix <-> master` : обычно не возникает.

### Дополнение

Исходники можно скачать <https://github.com/giantleaplab/myworkflow>