GitHub

GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name "[firstname lastname]"`

set a name that is identifiable for credit when review version history

`git config --global user.email "[valid-email]"`

set an email address that will be associated with each history marker

`git config --global color.ui auto`

set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`

initialize an existing directory as a Git repository

`git clone [url]`

retrieve an entire repository from a hosted location via URL

git clone [url] --branch [branch]

clone a branch of repository

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`

show modified files in working directory, staged for your next commit

`git add [file]`

add a file as it looks now to your next commit (stage)

`git reset [file]`

unstage a file while retaining the changes in working directory

`git diff`

diff of what is changed but not staged

`git diff --staged`

diff of what is staged but not yet committed

`git commit -m "[descriptive message]"`

commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`

list your branches. a * will appear next to the currently active branch

`git branch [branch-name]`

create a new branch at the current commit

`git checkout [branch]` | **git checkout -b [branch-name]** A

switch to another branch | create & switch to branch

`git merge [branch]` B

merge the specified branch's history into the current one

`git log`

show all commits in the current branch's history

git branch -d [branch]

delete a branch

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`

show the commit history for the currently active branch

`git log branchB..branchA`

show the commits on branchA that are not on branchB

`git log --follow [file]`

show the commits that changed file, even across renames

`git diff branchB...branchA`

show the diff of what is in branchA that is not in branchB

`git show [SHA]`

show any object in Git in human-readable format

git show [commit id] ⟼ Changes of given commit

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`

delete the file from project and stage the removal for commit

`git mv [existing-path] [new-path]`

change an existing file path and stage the move

`git log --stat -M`

show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`

system wide ignore pattern for all local repositories

SHARE & UPDATE

git remote ⟼ show all remotes | **git remote -v** ⟼ show all remote with url

`git remote add [alias] [url]`

add a git URL as an alias

`git fetch [alias]`

fetch down all the branches from that Git remote

`git merge [alias]/[branch]`

merge a remote branch into your current branch to bring it up to date

`git push [alias] [branch]`

Transmit local branch commits to the remote repository branch

`git pull` (it's done for 'current alias' and 'current branch')

fetch and merge any commits from the tracking remote branch

git pull [alias] [branch] ⟼ above action for given branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`

apply any commits of current branch ahead of specified one

`git reset --hard [commit]`

clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`

Save modified and staged changes

git stash show [stash id]

show stash changes

`git stash list`

list stack-order of stashed file changes

git stash

clear stash list

`git stash pop` **[stash id]**

apply stash with deleting

git stash apply [stash id]

apply stash without deleting

`git stash drop`

discard the changes from top of stash stack

git tag

all tags list

git tag [name]

create a tag with name (like v1.0.0)
and message is last commit

git tag -a [name] -m "Message"

create a tag with name (like v1.0.0)
and message

git push -u [remote] [branch] [tag name]

push a tag

git push -u [remote] [branch] --tags

push all tags

نکته A :

قبل از سوییچ حتما یه

```
git status
```

بزن باید کلین باشه

اگه نبود برای کامیت دستور

```
git add .
git commit -m "Message"
```

بزن تا تغییرات در برنچ فعلی ذخیره بشه سپس بری به یه برنچ دیگه

نکته : اگه کارت تکمیل نشده که کامیت کنی ولی میخای سوییچ کنی به یه برنچ دیگه میتونی

```
git add .
git stash
```

بزنی تا تغییرات رو ذخیره کنه سپس سوییچ کنی و وقتی خواستی تغییرات رو برگردونی ابتدا با

```
git stash list
```

میتونی لیست فایل های ذخیره شده به همراه اسم استش رو ببینی

و با دستور

```
git stash show *
```

و بجای ستاره عبارت پشت دونقطه میتونی جزئیات استش ذخیره شده رو ببینی

و با دستور

```
git stash apply *
git stash pop *
```

استش ذخیره شده رو برگردونی اپلای فقط استش ذخیره شده رو اعمال میکنه و استش رو نگه میداره ولی پاپ علاوه بر اعمال اون استش رو حذف میکنه

نکته : دستور

```
git stash clear
```

لیست استش هارو پاک میکنه

نکته B :

این دستور خودش تغییرات رو ذخیره میکنه و نیاز به

```
git add . & git commit -m "Message"
```

نیست

نکته : اگر برای مرج کردن ادیتوری باز شد و کامنت خواست با انتخاب

esc+:i

میتونی اینسرت کنی کامنتت رو و سپس با انتخاب

esc+:wq

میتونی ذخیرش کنی.

ادیتوره اکثرا

vim

هست

نکته : اگر زمان مرج کردن

CONFLICT

رخ داد یعنی تداخل داشته و قسمتی که تداخل داشته رو نوشته داخل کد هم متمایز کرده برو به قسمتی که گفته تداخل داشته و دستی چیزی که میخای باشه رو درست کن سپس سیو کن و بیا به گیت و دستورات

```
git add .
git commit -m "fix-conflict"
```

رو بزن