Cheatography

How to do stuff with Git Cheat Sheet

by nahuelsanchez via cheatography.com/20293/cs/3105/

What's going on?

git status

Shows the changes in the working directory.

git diff

Shows the changes on the specific files to indicate what you're about to commit

What's happened?

git log

Shows last commits in the working directory.

git log path/to/file

Show last commits in the specified file or directory.

How do I move between branches?

git branch

Shows all the branches

git checkout -b *branch-name*

Creates a new branch based on the one you currently are and moves you to it.

git checkout *branch-name*

Moves you to an existing specified branch.

git branch -D *branch-name*

Deletes a specified branch.

What if I need to take a pause?

git stash

Put the current files state in a "limbo" to retrieve it later, and makes a reset of the current branch to the last commit (like a *git reset HEAD*)

What if I need to take a pause? (cont)

git stash list

Shows the saved stashes with the ID number to retrieve them.

git stash pop

Retrieves the last made stash and removes it from the stash list.

git stash apply

Retrieves the last made stash and without removing it from the stash list.

git stash apply stash@{*stash-number*}

Retrieves the specified stash without removing it from the stash list.

git stash clear

Deletes all the stashes.

What if I regret something?

git checkout -- *path/to/file*

Discard changes in the file (for tracked files), so it's reverted to the last commit.

git reset *commit id*

Reverts changes in your working directory that haven't been comitted yet. It's a time machine to specified commit.

git reset HEAD

Same as above, where *HEAD* is the last commit.

git revert *commit id*

Same as *git reset* but it creates a new commit with the reverted changes.

How do I submit my work?

git add *path/to/file*

Add the file's changes under the *Changes to be committed* state so you can commit those changes.

git commit -m '*message here*'

Creates a new commit with all the changes under the *Changes to be committed* state.

git push *remote-name remote-branch*

Push the commits to the specified remote and branch (for example *git push origin develop*).

How do I grab the changes from the remotes?

git pull *remote-name remote-branch*

Pulls the commits from the specified remote and branch and merges them into the local branch (for example *git pull origin develop*).

git pull --rebase *remote-name remote-branch*

Same as above but instead of a merge at the end of grabbing commits it will do a rebase.

git fetch *remote-name*

Grabs what's on the remotes but it doesn't merge or rebase the commits on your local branches.

How do I manage the remotes?

git remote -v

Shows the remotes names and URLs.

git remote add *name url*

Adds a new remote using the specified name and url.

By **nahuelsanchez**

cheatography.com/nahuelsanchez/

Not published yet.
Last updated 22nd December, 2014.

Cheatography

How to do stuff with Git Cheat Sheet

by nahuelsanchez via cheatography.com/20293/cs/3105/

Possible states of a file

Changes to be committed

> Usually in green on the command line, shows the changes that are ready to be commited.

Changes not staged for commit

> Shows the changes made since the last commit, but in this state they are not going to be commited.

Untracked files

> Shows files in which Git ignores changes.

Unmerged paths

> Shows files with conflicts that needs to be manually fix it.

What can cause conflicts?

When you do a *git merge*, *git pull* or *git stash pop/apply* you can get conflicts because other commits affected the same files you worked on. When this happens you'll get the chance to fix this conflicts. The files with conflicts on it will be in a new state: **Unmerged paths**.
A conflict can be caused because a commit deletes a file and other commit add it or made changes on it, or even when two commits added the same file.
Remember to always commit all your work before doing a *git merge*, *git pull* or *git stash pop/apply*.

Conflicts because of a git stash pop/apply?

If a *git stash pop* or any other *git stash* command that grabs a stash and put it in the working directory produces a conflict, it will show all the files with conflicts at once under the *Unmerged paths* state.
You need to go through each file and resolve the conflicts on it.
When you fixed a conflict on a file, do a *git add path/to/file*. And when you're done fixing the conflicts on all files you can commit those fixes with *git commit*.

Conflicts because of a git merge?

If a *git merge* produces a conflict, it will show all the files with conflicts at once under the *Unmerged paths* state.
You need to go through each file and resolve the conflicts on it.
When you fixed a conflict on a file, do a *git add path/to/file*. And when you're done fixing the conflicts on all files you can commit those fixes with *git commit*.
In this case a *git commit* will create a new commit with the conflicts fixed.

Conflicts because of a git pull?

Same as the conflicts during *git merge*.

Conflicts because of a git pull --rebase?

If a *git pull --rebase* encounters a conflict between commits it will pause itself and move you to a temporary branch in order to let you fix the conflicts before continue.
In this case you can go through each file and resolve the conflicts on it.
When you fixed a conflict on a file, do a *git add path/to/file*. And when you're done fixing the conflicts on all files you can continue with the rebase by doing a *git rebase --continue*.
If you want to omit the commit that it's causing the conflicts you can do a *git rebase --skip*. It will skip the entire commit, not the files with conflicts.
If things gets a little bit complicated you can abort the operation by doing a *git rebase --abort*.

By **nahuelsanchez**

cheatography.com/nahuelsanchez/

Not published yet.
Last updated 22nd December, 2014.