All in one Git Cheat Sheet

By Rushikesh Mashidkar

Set up a Repository

git init

Create a new repository in the path where the command is executed

git clone [url]

clone a repository to the current directory

git clone [url] [new directory]

clone a repository to a new directory

Collaboration and Synchronization with GitHub

These commands are used to work collaboratively with the remote repository

Set up Repository Access on GitHub

Access your repository on GitHub and click on the "Settings" tab

On the settings page, click the "Collaborators" tab

In the "Search by username" field , enter the username of the person you want to add as a collaborator

Select the username from the list of results and click "Add collaborator"

The person you added will receive a collaboration request in their GitHub email. When you accept it, you can start collaborating in the repository

To add contributors to a repository on GitHub, you must first have repository owner permissions or be a user with write permissions .

Types of access permissions for collaborators

Read access: Contributors with this type of access can view and download code from the repository, but cannot make changes or submit pull requests.

Types of access permissions for collaborators (account)

Write access: Contributors with this type of access can make changes and submit pull requests, but they cannot delete the repository or change repository settings .

Administrator access : Contributors with this type of access have full access to the repository and can make changes, submit pull requests, delete the repository, and Change the repository settings .

You can change a contributor's access type at any time on the repository settings page in the "Contributors" section .

Git push

git push [remote repo] [branch name]

Public commits from the local repository to the remote repository. You can skip the repository if you want to continue using the same repository. Example: git push origin master

git push [remote repo] --all

Publishes commits from all branches to the remote repository.

git push [remote repo] --tags

Public tags local to the remote repository

- Tags of the not automatically synchronize with the remote repository with other push commands. https://www.atlassian.com/git/tutorials/syncing#git-push

Git pull

git pull [remote repo]

Bring changes from remote repository. Almost forward merge. You can skip repo if it has already been configured.

git pull [remote repo]/[branch name]

Brings changes from a branch and merges them to the local repository.

- Git pull command = git fetch + git merge
- More information and examples: https://www.atlassian.com/git/tutorials/syncing#git-pull

All in one Git Cheat Sheet

By Rushikesh Mashidkar

Git fetch

git fetch [remote repo name]

Displays changes for all branches in the remote repository

git fetch [remote repo name] [branch]

Displays commits in a branch of the repository.

git fetch --dry-run

View changes from a remote repository before pulling

- Used to see what other collaborators have worked on
- Fetched content is represented as a remote branch and does not affect the local repository. https://www.atlassian.com/git/tutorials/syncing#git-fetch

Git remote

git remote

Check if you have a remote repository. If done on a cloned repository, it returns the original repository

git remote -in

Displays the full path of the remote repository

git remote add origin [github url]

Add a remote repository

git remote [url] [branch name]

Point remote branch to the correct url

git remote rm [remote repo name]

Remove connection to specified repository

Branches and Repositories

git switch [remote branch]

Brings changes from a remote branch to the local repository and creates the branch locally.

git push --set-upstream [repository] [new branch name]

This command is used to push a new branch to a remote repository and set the upstream for the new branch.

Example:- ***git push –set-upstream origin new-feature***

Time travel with Git

These commands are used to undo changes, roll back states, and retrieve code to any point in Git history.

Git restore

git restore [file name]

Retrieves the state of the file in question according to the HEAD

git restore [commit id] [file name]

Retrieves the state of the file in question according to a specific commit

git restore --staged [file name]

Pulls a file from the staged area to unstaged

This is a new command that helps undo operations. Since it is new most tutorials do not mention it, its functions can be replaced with git checkout.

Git checkout

git checkout [commit id]

Returns the project to a specific commit in a detached HEAD state

git checkout HEAD~ [number]

Returns the project by moving [number] positions backwards from HEAD. HEAD~1 refers to the pre-HEAD commit. HEAD~2 refers to 2 commits before.

This command is like a wild card, or the Swiss Army knife of git. It is often said that it is overloaded which led to the addition of the git switch and git restore commands.

It can be used to create branches, switch between branches, retrieve files, and travel in history, but it is recommended to use it alone

Git Exploratory Commands

git status

lists files that have been staged, unstaged, commited, and tracked

git show

Shows changes made in the last commit

git show [commit id]

Displays changes made to a specific commit

All in one Git Cheat Sheet

By Rushikesh Mashidkar

Git Exploratory Commands (account)

git shortlog

Alphabetical list of the names and messages of the commits made by each person in the repository

git shortlog -s -n

Shows the number of commits made by each person

git log

Displays the complete history of all commits made

git log -n [number]

Displays only the specified number of commits in the history

git branch

List the branches in the repository

git branch -a

Lists branches in the repository, including remote branches

Git Day

Displays existing tags in the repository

Save Changes

This set of commands is used to add changes to Git history

Git add

git add [file name]

Add files to the staging area

git add .

Add all files to the staging area. This includes those found to be *untracked*

git add '*[file type]'

Add all files from a ti on of the po l to the staging area. Example: git add . .txt

git add [directory]

Add all changes from a directory to the staging area

More information about this command: https://www.atlassian.com/git/tutorials/saving-changes#git-add

Git commit

git commit

Open a word processor to add a commit message on the top line. Add staging area changes to a commit.

git commit -m ["commit message"]

Creates a commit with the changes to the staging area and the specified message.

git commit -a -m ["commit message"]

Create a commit with all changes to the files in state *tracked*. No Adds to the commit *untracked* files.

Guide to writing commit messages: http://udacity.github.io/git-styleguide/

Git diff

git diff

Displays changes to the local directory (not staged)

git diff --staged

Shows changes from the staging area

git diff HEAD

Displays changes between staged and unstaged files

Git stash

git stash

Add all current staging changes to a special area called stash, where they can be retrieved later.

git stash pop

Gets the changes from the stash and takes them to the current branch

The stashing area is very useful for saving changes temporarily to change branches or bring remote changes without making unnecessary commits.

More about stash:

https://git-scm.com/docs/git-stash

All in one Git Cheat Sheet

By Rushikesh Mashidkar

Git Branches

git branch

List the branches of a local repository

git branch -a

Lists all branches in a repository , including those in the remote repository

git branch -r

Lists only remote branches in a repository

git branch [new branch name]

Create a new branch with the specified name

git branch [new branch name] [commit id]

Creates a new branch from the specified commit

git branch -d [branch name]

Deletes a branch. You cannot delete a branch if it is in it or has not been made merge of the changes.

git branch -D [branch name]

It removes a branch by force (does not require merge), but it cannot be done if it is on the branch.

git switch [branch name] or git checkout [branch name]

Serves to change to the branch in question

git switch -c [branch name] or git checkout -b [branch name]

Creates a new branch and changes the HEAD pointer to it.

git branch -m [new name]

Rename a branch.

git merge [branch name]

Brings the contents of the specified branch to the current branch. It does not delete the branch we are combining, it still exists. If it is No longer needed it must be removed manually.

Undo Commits

git reset [commit id]

Undo the commit in question and the changes are left in the working directory as unstaged

gear reset --hard [commit id]

Undo the commit and revert the files to the previous state. The changes contained in the commit are lost forever. Use with caution

git revert [commit id]

Undo the changes to the commit in question. Unlike reset, it does not delete the commit, but creates a new commit by reverting the changes.

If you want to undo changes that other collaborators already have on their machines, it is recommended to use revert.
If you haven't yet pushed the changes, use reset and no one ever You'll know.

Options in Detached HEAD state

Discard changes

To discard changes make **git switch** to an existing branch

Create a new branch with the changes

To create a new branch from the current state , just of the git switch -c [new branch] this branch will be performed from the current state and will include the work done in the local directory.

Make commits, changes, etc...

You can make changes and commit to this state and review the files, but these changes will not be persistent unless you create a new branch. If you switch to an existing branch, they will be lost forever.