Git Cheat Sheet

The essential Git commands every developer must know

This cheat sheet covers all of the Git commands I've covered in my Ultimate Git Mastery course.

- ✓ Creating snapshots
- ✓ Browsing history
- ✓ Branching & merging
- ✓ Collaboration using Git & GitHub
- ✓ Rewriting history

Hi! My name is Mosh Hamedani. I'm a software engineer with two decades of experience. I've taught millions of people how to code or how to become a professional software engineer through my YouTube channel and online coding school. It's my mission to make software engineering simple and accessible to everyone.

Check out the links below to master the coding skills you need:

https://codewithmosh.com

https://youtube.com/user/programmingwithmosh

https://twitter.com/moshhamedani

https://facebook.com/programmingwithmosh/

Want to master Git?

Stop wasting your time memorizing Git commands or browsing disconnected tutorials. If you don't know how Git works, you won't get far.

My **Ultimate Git Mastery** course teaches you everything you need to know to use Git like a pro.

- ✓ Learn & understand Git inside out
- ✓ Master the command line
- ✓ Version your code and confidently recover from mistakes
- ✓ Collaborate effectively with others using Git and GitHub
- ✓ Boost your career opportunities

Click below to enroll today:

https://codewithmosh.com/p/the-ultimate-git-course/

Table of Content

Creating Snapshots

修改默认设置:
git config --global -e

Initializing a repository

git init

git每个版本都会保留全部文件
（为什么我印象中只保留改动？）

Staging files

git add file1.js # Stages a single file
git add file1.js file2.js # Stages multiple files
git add *.js # Stages with a pattern
git add . # Stages the current directory and all its content

Viewing the status

git status # Full status
git status -s # Short status

第一列是stage area,
第二列是working directory

Committing the staged files

git commit -m “Message” # Commits with a one-line message
git commit # Opens the default editor to type a long message

Skipping the staging area

git commit -am “Message”

git ls-files 显示当前stage区的文件

Removing files

git rm file1.js # Removes from working directory and staging area
git rm --cached file1.js # Removes from staging area only

以处理gitignore没有及
时忽略的文件

[-r] 递归删除文件夹下的所有文件

Renaming or moving files

git mv file1.js file1.txt

同时重命名working directory和staging are的文
件

Viewing the staged/unstaged changes

```
git diff                    # Shows unstaged changes
git diff --staged           # Shows staged changes
git diff --cached           # Same as the above
```

在commit之前，stage area的改动

```
git config --global diff.tool vscode
git config --global difftool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"
```

Viewing the history

```
git log                     # Full history
git log --oneline           # Summary
git log --reverse           # Lists the commits from the oldest to the newest
```

Viewing a commit

```
git show 921a2ff            # Shows the given commit
git show HEAD               # Shows the last commit
git show HEAD~2             # Two steps before the last commit
git show HEAD:file.js       # Shows the version of file.js stored in the last commit
```

Unstaging files (undoing git add)

```
git restore --staged file.js  # Copies the last version of file.js from repo to index
```

如果file.js在work directory也做了修改，那么执行这个指令后，如果从staged area回退的文件和本地的文件有冲突怎么办？
首先，在git add之后，本地文件和stage area文件一样，所以restore --staged的作用仅仅是改变stage area的文件状态，而不会影响工作区

Discarding local changes

```
git restore file.js         # Copies file.js from index to working directory
git restore file1.js file2.js  # Restores multiple files in working directory
git restore .               # Discards all local changes (except untracked files)
git clean -fd               # Removes all untracked files
```

Restoring an earlier version of a file

```
git restore --source=HEAD~2 file.js
```

Browsing History

Viewing the history

```
git log --stat              # Shows the list of modified files
git log --patch             # Shows the actual changes (patches)
```

Filtering the history

```
git log -3                  # Shows the last 3 entries
git log --author=“Mosh”
git log --before=“2020-08-17”
git log --after=“one week ago”
git log --grep=“GUI”        # Commits with “GUI” in their message
git log -S“GUI”             # Commits with “GUI” in their patches
git log hash1..hash2        # Range of commits
git log file.txt            # Commits that touched file.txt
```

Formatting the log output

```
git log --pretty=format:”%an committed %H”
```

Creating an alias

```
git config --global alias.lg “log --oneline"
```

Viewing a commit

```
git show HEAD~2
git show HEAD~2:file1.txt   # Shows the version of file stored in this commit
```

Comparing commits

```
git diff HEAD~2 HEAD        # Shows the changes between two commits
git diff HEAD~2 HEAD file.txt # Changes to file.txt only
```

Checking out a commit

```
git checkout dad47ed        # Checks out the given commit
git checkout master         # Checks out the master branch
```

Finding a bad commit

```
git bisect start
git bisect bad              # Marks the current commit as a bad commit
git bisect good ca49180     # Marks the given commit as a good commit
git bisect reset            # Terminates the bisect session
```

Finding contributors

```
git shortlog
```

Viewing the history of a file

```
git log file.txt            # Shows the commits that touched file.txt
git log --stat file.txt     # Shows statistics (the number of changes) for file.txt
git log --patch file.txt    # Shows the patches (changes) applied to file.txt
```

```
git log --oneline --all --graph
```

Finding the author of lines

```
git blame file.txt          # Shows the author of each line in file.txt
```

Tagging

```
git tag v1.0                # Tags the last commit as v1.0
git tag v1.0 5e7a828        # Tags an earlier commit
git tag                     # Lists all the tags
git tag -d v1.0             # Deletes the given tag
```

Branching & Merging

branch的命名可以随意，比如加斜杠或者-， 如fixbug/namebug

Managing branches

git branch bugfix	# Creates a new branch called bugfix	
git checkout bugfix	# Switches to the bugfix branch	
git switch bugfix	# Same as the above	prefer to use this command, because it is much clear
git switch -C bugfix	# Creates and switches	
git branch -d bugfix	# Deletes the bugfix branch	

Comparing branches

git log master..bugfix	# Lists the commits in the bugfix branch not in master
git diff master..bugfix	# Shows the summary of changes

两个点

当当前分支的本地变动不想stage,但是又需要切换到另一个分支的时候，需要临时stash一下当前的变动。
但是git stash不会保存未track的文件，所以需要加上-all选项。
git stash push -am

Stashing

git stash push -m “New tax rules”	# Creates a new stash
git stash list	# Lists all the stashes
git stash show stash@{1}	# Shows the given stash
git stash show 1	# shortcut for stash@{1}
git stash apply 1	# Applies the given stash to the working dir
git stash drop 1	# Deletes the given stash
git stash clear	# Deletes all the stashes

Merging

git merge bugfix	# Merges the bugfix branch into the current branch	
git merge --no-ff bugfix	# Creates a merge commit even if FF is possible	
git merge --squash bugfix	# Performs a squash merge	
git merge --abort	# Aborts the merge	merge之后不想解决冲突，就终止merge

配置禁止fast-forwar merge:
1. git config ff no
2. git config --global ff no

Viewing the merged branches

git branch --merged	# Shows the merged branches
git branch --no-merged	# Shows the unmerged branches

应该及时删除已经merge的分支，git branch -d branch_name

Rebasing

git rebase master	# Changes the base of the current branch

Cherry picking

git cherry-pick dad47ed	# Applies the given commit on the current branch

当git merge产生conflict时，需要手动解决conflict,但是此时不要添加额外的代码。因为当前的commit只用于merge这两个分支。

Collaboration

Cloning a repository

git clone url

Syncing with remotes

git fetch origin master	# Fetches master from origin
git fetch origin	# Fetches all objects from origin
git fetch	# Shortcut for “git fetch origin”
git pull	# Fetch + merge
git push origin master	# Pushes master to origin
git push	# Shortcut for “git push origin master”

只是把代码从github取回本地版本库，但是HEAD依然指向当前的commit

git pull时，如果origin和local都有commit，则先通过git fetch在本地的master之外建立一个新的分支，然后在进行three-way merge，建立一个新的commit，最后将master指向这个commit

Sharing tags

git push origin v1.0	# Pushes tag v1.0 to origin
git push origin —delete v1.0	

先通过git tag v1.0 添加这个tag,github将自动在这个tag处打包源码

git push也会出现本地和远程分支都有新的commit的问题。
这种情况，首先使用git pull，然后进行three-way merge,最后再push

Sharing branches

git branch -r	# Shows remote tracking branches
git branch -vv	# Shows local & remote tracking branches
git push -u origin bugfix	# Pushes bugfix to origin
git push -d origin bugfix	# Removes bugfix from origin

Managing remotes

git remote	# Shows remote repos
git remote add upstream url	# Adds a new remote called upstream
git remote rm upstream	# Remotes upstream

将仓库回退到上一个commit
git reset --hard HEAD~1

Rewriting History

Undoing commits

只改变版本区，不改变工作区和缓存区

```
git reset --soft HEAD^       # Removes the last commit, keeps changed staged
git reset --mixed HEAD^      # Unstages the changes as well
git reset --hard HEAD^       # Discards local changes
```

改变版本区，且将缓存器回退

彻底回退到上一版本

Reverting commits

```
git revert 72856ea           # Reverts the given commit
git revert HEAD~3..          # Reverts the last three commits
git revert --no-commit HEAD~3..
```

Recovering lost commits

```
git reflog                   # Shows the history of HEAD
git reflog show bugfix       # Shows the history of bugfix pointer
```

Amending the last commit

```
git commit --amend
```

Interactive rebasing

```
git rebase -i HEAD~5
```