Git Cheat Sheet

```
git init localRepositoryName
```
initializes (creates) a local repository

```
git status
```
displays the status of the current repository

```
git log
```
logs the commits on the repo

```
git log --oneline
```
logs the commits in a shorter format

```
git log --stat
```
displays extra info like the files that were changed and the number of lines added or deleted

```
git log -p          /          git log -patch
```
displays the changes the the file (file patch)

```
git log -p sha
```
goes directly to the specified commit called by its SHA

```
git log -p --stat       /         git log --stat -p
```
logs both the stats followed by the file patch (changes)

```
git log -p -w
```
ignores white/blank space changes

```
git log --decorate
```
also displays tags

Note: since Git 2.13, this is shown by default, without the --decorate flag

```
git log --oneline --graph --all
```
displays all branches and all commits in repository

```
git show
```
display the most recent commit only

Note: it can be combined with all the other flags mentioned above for git log

prepared by
/kh-m

Git Cheat Sheet

```
git show sha
```
displays the commit called by its SHA in the final argument

```
git add fileName
```
add the file in the current directory to the staging area (‘stages’)[can take multiple files as arguments]

```
git add .
```
stages all the files/directories in the current directory

```
git rm --cached fileName
```
removes the file from the staging area (‘unstages’)

```
git commit
```
commits the files that have been staged for a commit

```
git commit -m "yourCommitMessage"
```
commits like the previous command, but you bypass opening the editor by placing the commit message (comment) directly [you can only provide a commit message, but not a description]

```
git commit --amend
```
lets you provide a new commit message & you can make changes to the files, run *git add* and then run the *git commit --amend* command to change the files in the most recent commit

```
git commit revert sha
```
undoes the changes from the provided commit and creates a new commit to record that change

```
git diff
```
displays the difference between the last committed files and the ones that have only been saved but not committed

```
touch .gitignore
```
creates a (text) file in which you can list all files to be ignored when using *git add*

e.g.	**.txt*	adds all txt files
	video.*	adds all files called video
	video*	adds all files that start with video
	temp/*	adds temp folder and all its contents
	/temp/	adds any temp folder within any folder in the current directory

prepared by
/kh-m

Git Cheat Sheet

```
git tag
```

displays all current tags

```
git tag yourTag
```

adds a tag to the most recent commit

```
git tag -a yourTag
```

adds an annotated tag (in which editor will open) that can include extra info like person who made it, date, message

[if you don't use an annotated tag, you'd create what's called a 'lightweight tag']

```
git tag -a yourTag tagSha
```

adds a tag to a specified commit by referencing its SHA

```
git tag -d tagToBeDeleted        /    git tag -delete tagToBeDeleted
```

deletes the tag specified

```
git branch
```

shows you how many branches you have

```
git branch branchName
```

creates a new branch with the provided name

```
git branch branchName sha
```

creates a branch with the provided name pointing at the commit with the provided SHA

```
git checkout branchName
```

switches (head) to the branch provided

```
git checkout -b newBranchName
```

creates a new branch with the provided name and switches to it

```
git branch -d branchName         /         git branch -delete branchName
```

deletes the provided branch

[you cannot delete a branch that you are presently working on; you'll have to change the head to another branch. You also cannot delete a branch which you have branched further off of, you'd need git branch -D branchName]

```
git branch -D branchName
```

forcibly deletes the provided branch

```
git merge nameOfBranchToMergeIn
```

merges the current branch into the one branch with the branch provided

```
git reset commitReference        /        git reset --mixed commitReference
```

moves it to the working directory *e.g. git reset HEAD~1* will move the head back to its parents and move where the head was at to the working directory

```
git reset --soft commitReference
```

moves the commit to the staging index

```
git reset --hard commitReference
```

moves the commit to trash