Git Cheatsheet

Git, the magical version control

Git is an open-source version control system (meaning, it's free!) that allows developers to track changes made on their code files throughout the lifetime of a project.

Git is a great version control tool because it's lightweight and straightforward to use, and it provides incredible compression and speed.

Download & Install Git

Download Git for your operating system here: http://git-scm.com/downloads

When you install Git, nothing will happen. After you install Git, open your terminal application, type `git` and hit enter. You should see details about Git, including common commands.

NOTE: All Git commands are run in the command line or terminal application, unless you download one of Github's applications.

Sign up for GitHub

A great place to store your repositories is on GitHub. Setting up an account is free, and you can have as many public repositories as you need.

GItHub is also a great way to build a resume for yourself. Often, when developers are applying for jobs, they simply share their GitHub profile link to show potential employers what they've done.

Git Cheatsheet

Gitting Started

git init FIRST STEP

Before you do anything you have to initialize, or create, your Git repository. You do this with the *git init* command. It says to your computer "Hey this folder I am in, it is now a Git repository."

```
$~ git init
```

git add . SECOND STEP

When you are ready to commit your changes you first need to stage them. Use the period (.) to add all directories and files. With your very first commit, you almost always want to add all of your directories and files, not just some of them.

```
$~ git add .
```

git commit -m "Write a message here" MESSAGE REQUIRED

When you have added all of your files, it's time to commit. Committing a collection of changes is like taking a snapshot of your work. These snapshots are what get pushed up to your remote repository. Commits are also handy for rolling back your work to a previous version.

```
$~ git commit -m "Sometimes its hard to come up with git messages"
```

Handy Commands & Files

git status

As you're working on your project, it's a good idea to periodically run *git status* to see the changes you've made. This will show you all files that have been edited, any new files or directories, and any files or directories that have been deleted. It's a *very* good idea to run git status before you add files and commit changes.

```
$~ git status
```

Git Cheatsheet

git diff path/to/file

This command shows the changes you've made to the specified file by diffing your local version with the upstream version. It will show you code that has been added and deleted. It's good practice to run *git diff* before you add a file for a commit.

```
$~ git diff www/sample/index.html
```

git add path/to/file

git add path/to/directory

You already know how to add all files and directories with *git add .*, but you can also add files and directories one-by-one.

```
$~ git add www/sample/index.html
```

```
$~ git add www/sample/css/
```

git checkout path/to/file

If you've made a complete mess of a file, and it's past the point of saving, you can easily revert back to the remote version of the file by checking it out.

```
$~ git checkout www/sample/index.html
```

git stash

If you have a bunch of edits that you need to move out of the way so you can work on other things, you can use stash. Stash is like moving all of your edits to the back burner for awhile. You can run this several times and store many stashes.

```
$~ git stash
```

git stash apply

When you're ready to move your code back to the front burner, use git stash apply. This will move everything that has been stashed away, so if you've run the git stash command multiple times, everything will come along.

```
$~ git stash apply
```

Git Cheatsheet

git stash clear

This empties everything out of your stash and gives you a clean slate for a new stash.

```
$~ git stash clear
```

git mv path/to/oldfilename /path/to/newfilename

Renames a file. This change has to be committed and pushed to the repo.

```
$~ git mv www/sample/home.html www/sample/index.html
```

git rm path/to/file

git rm -r path/to/directory

Removes a file or a directory (and everything in it). This change has to be committed and pushed to the repo.

```
$~ git rm www/sample/home.html
```

```
$~ git rm -r www/sample/images/
```

.gitignore

Sometimes you may have files or entire directories that you want to keep out of the remote repository while still keeping them in your local repository. The .gitignore file is used to track these files and directories. Add one file or directory per line.

```
.htaccess
```

```
/app/config.rb
```

Remote repositories & GitHub

git clone git@github.com:username/reponame.git path/to/local/dir

Clones a remote repository to your local environment, into the specified directory.

```
$~ git clone git@github.com:addabjork/sample.git www/sample
```

Git Cheatsheet

---

git remote add origin git@github.com:username/reponame.git

Adds a remote repository to your existing local repository. This allows you to push code up to the remote.

```
$~ git remote add origin git@github.com:addabjork/sample.git
```

---

git pull origin master BRANCHNAME REQUIRED

git pull origin *branchname*

Pulls code from the remote specified branch and merges it into the current local branch.

```
$~ git pull origin refactor
```

---

git push origin master BRANCHNAME REQUIRED

git push origin *branchname*

Pushes code up to the remote specified branch.

```
$~ git push origin refactor
```

---

git remote -v

Checks what remote you are working with and tells you the address.

```
$~ git remote -v
origin  git@github.com:addabjork/sample.git (fetch)
origin  git@github.com:addabjork/sample.git (push)
```

---

git remote rm master

git remote rm *nameofremote*

Removes the remote specified branch.

```
$~ git remote rm refactor
```

Git Cheatsheet

Branches!

Git is designed such that it views your code files like a “tree” and allows you to do cool things like create a “branch” where you work on some of the files without affecting the “trunk” code base until you are sufficiently convinced that the changes you are making are good and won’t break the rest of the tree.

git checkout -b *branchname*

Does checkout look familiar? In addition to checking out files, *git checkout* can be used to create and checkout branches. This command creates a new local branch and automatically checks it out. The new branch is in essence a copy of whatever branch you were on before you ran the command.

```
$~ git checkout -b v2.0
```

git checkout *branchname*

Checks out an existing branch.

```
$~ git checkout v2.0
```

git merge *branchname*

Merge an existing branch into the current branch. First checkout the branch you want to merge into.

```
$~ git checkout master
$~ git merge v2.0
```

git branch -a

Shows all existing branches and highlights the current branch.

```
$~ git branch -a
* master
  v2.0
  remotes/origin/HEAD -> origin/master
```

Git Cheatsheet

git branch -d *branchname*

Deletes the specified branch. Danger! Any changes will be lost if you do not first merge this branch into another one.

```
$~ git branch -d v2.0
```

More Information about Git & GitHub

For more information about Git, check out the Git documentation.

For more information about GitHub, check out GitHub's help guide.