Get Acquainted With Git[\*](#this-just-covers-some-basic-git-commands-doesnt-cover-remote-repos-push-or-pull)

=======================

Git Bash Command Console

> git --version

**Configuration:**

> git config --global color.ui true -- Add color

> git config --global user.name <your\_name> -- for tracking changes

> git config --global user.email <your\_email>

###Basic Commands

**Create Repo: -- Directory -- Create Repo Directory**

> git init

Creates a new Git repository in the current directory. In addition, a directory named \*\*.git\*\* is added to the folder. This folder holds all the Git history and information for the repo.

**To get rid of the repo**: just delete the .git folder

**.gitignore** -- A file which lists files and file extensions that Git should not track in the repo

Temp files, log files, .DS\_store, etc.

> git status

Show me the current status of the repository. Shows

1 \*\*Untracked files\*\* -- new files that have not be added to the staging or committed

2 \*\*Working files\*\* -- Files that are tracked, have been changed but have not been staged\*\*

3 \*\*Staged files\*\* -- files that have been added but not committed

4 \*\*Committed files\*\* -- file is added to repo and is now tracked by Git

**Commits: -- Commit changed files to the repository -- I want to remember the state of this file**

**Each commit is a record of something you did -- 10 changes and then commit**

**Add to Staging:**

> git add .

Adds all the files in the current directory to staging. “.” Every file where I am now and down

> git add FILENAME -- file added to staging area

**Commit:**

> git commit -m “Initial Commit - Message”

> git status

> nothing to commit, working directory clean -- all changes have been committed

**More add to Staging:**

> git add -A

Adds all files in the current repo (even new files that are not yet tracked)

> git add -u

Add all files that are already being tracked (ignore new files)

> git add -p

Add just parts of changes to a file. "p" stands for patch and lets you go through each section of a file that has changed and add (stage) just parts of the file

> git add <path/to/file>

Adds specified file to staging

> git commit -m "Add file to repo"

Commit staged files to the repo

> git commit --amend -m "New Message"

Changes the commit message for the last commit

> git commit -am "New Message"

Let's you add \*\*and\*\* commit all tracked, modified files in one step.

> git commit --amend -m "New commit message" Replace last commit. Use this if you forgot something that you wanted to inlcude in the last commit. Creates a new Git SHA.

**List of all commits done so far:**

> git log

Show a log of all commits.

> git log --oneline (short version)

Show a log of all commits, one line per commit. Can make this the default by ```git config format.pretty oneline```

**Recovering from a mess up:**

**Messed up Working File (file not yet staged)**

> git checkout <path/to/file>

**Messed up file and Staged**

**Unstage the file:**

> git reset HEAD <path/to/file> -- ‘HEAD’ is a reference to the last commit

And repeat the above command for not yet staged file:

> git checkout <path/to/file>

**Messed up file, Staged and Committed**

> git checkout HEAD^ <path/to/file> -- ^ refers to last commit - previous commit

**Undo an entire Commit -- go back to previous state -- ROLL BACK ENTIRE COMMIT**

> git reset --hard HEAD^

**Check what you have changed**

> git diff

Show differences between working tree and staging (or last commit)

> git diff -- staged

Show differences between staged changes and repository

> git diff -- <path to file/path to directory>

Show differences for specific file or directory

**Managing Files and Directories**

> git rm <path/to/file>

Remove a file that's being tracked in the repo. If you haven't yet added the file to staging, this will produce an error. You may need to force the removal if the file is staged but not committed:

> git rm -f <path/to/file>

> git rm -r <path/to/directory>

Remove a directory's worth of files. Also removes the directory. Directories themselves aren't tracked in Git. You may need to force the remove if a file in the directory is staged but not committed:

> git rm -rf <path/to/directory>

> git mv <path/to/file> <path/to/new-file-name>

Move a file that's been committed to the repo.

> git mv <path/to/directory> <path/to/new-directory-name>

Move a directory (and its files).

**Creating and Using and Merging Branches**

* **Master** branch -- main repo O -- O -- O --O 4 commits on master branch -- main working branch
* Create **Branch** to build new feature, fix bug, etc.
* **Merge**

> git checkout -b <name\_of\_branch>

Create a branch and check it out in one step

> git branch <name\_of\_branch>

Create a branch, but stay in current branch.

> git checkout <name\_of\_branch>

Check out an already created branch -- master or….

> git branch

See a list of all branches, highlights the currently checked out branch

> git checkout master

Checkout master branch

> git branch -d <name\_of\_branch>

**Delete** branch locally, but only if you've merged branch.

> git branch -D <name\_of\_branch>

**Delete** branch even if you haven't merged changes.

> git checkout master

> git merge <name\_of\_branch>

Return to master branch and merge changes from <name\_of\_branch> branch

###Getting Out of Changes

####File Fixes

```git reset HEAD <path/to/file>``` Unstage a file. (HEAD represents the current commit)

```git reset HEAD``` Unstage \*\*all\*\* staged files.

```git checkout <path/to/file>``` To revert to the last committed version of a file but only if a) the file has been committed and b) is not currently in staging

```git checkout HEAD^ <path/to/file>``` Revert to version of file from prior commit (HEAD^ represents the prior commit). Careful: this overwrites changes to files in your working branch.

```git checkout <sha-of-commit> <path/to/file>``` Revert to version of file from specific commit

####Undo Commits

Be careful with these commands when working on a shared repository -- for example with Github. Resetting commits changes the "history" of the repo -- so only use it to back out of local commits that haven \*\*not\*\* been pushed to a shared repository.

```git reset --soft HEAD^``` Undo last commit of entire repo, but leave files staged.

```git reset --hard HEAD^``` Completely blow away last commit. Changes files to state of previous commit.

```git reset --hard HEAD^^``` Completely blow away last two commits. Changes files to state of previous commit.

```git reset --hard HEAD^^^``` Completely blow away last three commits. Changes files to state prior to last third commit.

```git reset --hard <sha-of-commit>``` Returns files to state they were in after specificed commit

###Finding Differences Between Versions

```git diff``` View differences between current working files and staging area (or if files aren't staged compare working with last commit).

```git diff <path/to/file>``` View differences between current working file and staging area (or if file isn't staged compare working with last commit).

```git diff --staged``` View differences between staged files and last commit.

**A Simple Workflow**

\*\*Don't work on the master branch.\*\*<br> Master branch should hold your working, production files. Don't mess with them. When you need to fix something, or add a new feature to your project, create a new working branch. Make changes to that branch, then merge them into the master branch when done. You can then deploy your master branch (push it up to a web server, for example)

1. Make sure master is up-to-date.

Add and commit files, if there are any.

2. Create and checkout a new working branch:<br>

```git checkout -b <working\_branch\_name>```

3. Make changes to this branch. Make sure to add files and make commits along the way.

4. When done with branch:<br>

```git status```

Just to check and make sure that there are no outstanding changes that have yet to be committed. \*\*If there are, add and commit files.\*\*

5. Switch back to master<br>

```git checkout master```

6. Merge changes from working branch<br>

```git merge <working\_branch\_name>```

7. Remove branch<br>

```git branch -d <working\_branch\_name>```

8. Deploy master (push to web server for example).

9. Repeat steps 1-8 for your next feature/set of changes.

If things gets TOTALLY messed up in your working branch, you can just switch back to the master branch and delete the working branch:

1. ```git checkout master```

2. ```git branch -D <working\_branch\_name>```

Then just follow steps 1-7 again.

###Useful Configuration Options

```git config --global alias.st status``` Let's you just type ```git st``` whenever you want to see the status of the repo

```git config --global alias.co checkout``` Let's you just type ```git co``` whenever you want to checkout a branch

```git config --global alias.ci commit``` Let's you just type ```git ci``` whenever you want run a commit

####\*This just covers some basic Git commands. Doesn't cover remote repos, ```push``` or ```pull```

**GitHub Collaborative Development**

**Add REMOTE GitHub repository to a LOCAL project and push LOCAL project to GitHub Repo**

> git remote add origin https://github.com/danielniclas/express-basics.git

> git push -u origin master

(‘origin’ can be changed to anything you want -- ‘home’, ‘github’ etc.)

**Add Collaborator and use code in GitHub:**

‘SSH clone URL’

> git clone <SSH clone URL>

**Change REMOTE ORIGIN:**

git remote -v

# View existing remotes

# origin https://github.com/user/repo.git (fetch)

# origin https://github.com/user/repo.git (push)

git remote set-url origin https://github.com/user/repo2.git

# Change the 'origin' remote's URL

git remote -v

# Verify new remote URL

# origin https://github.com/user/repo2.git (fetch)

# origin https://github.com/user/repo2.git (push)