Git Basics

Joseph Vantassel, The University of Texas at Austin

Sources:

Git Tutorial from Code Academy
Git Tutorial for Beginners: Command-Line Fundementals
Git Tutorial: Fixing Common Mistakes and Undoing Bad Commits
Why is Git always asking for my password? Committed a Large File by Accident?

Setup/Configuration

```
git --version                           # Determine version number
git config --global user.name John Doe      # Set up configuration with name
git config --global user.email jd@mail.com  # Set up configuration with email
git config --list                       # Confirm and view changes
```

Need help?

```
git help <verb>
git <verb> --help
# For Example
git help config
git config --help
```

Local Repositories

```
cd <myprojectdirectory>         # Move to project directory you want to track
ls -la                          # List files in directory
git init                        # Initialize git directory
ls -la                          # List files in directory -> Note .git folder!
```

What's going on at this point

```
git status                      # Shows what is and is not being tracked
```

What if we have a file(s) that we dont want to track. Use **.gitignore**

```
ls -la                          # List files in directory
touch .gitignore                # This creates a .gitignore file for us
ls -la                          # List files in directory -> Note .gitignore file!
```

But how does it know what we want to ignore.
For that, we need to edit the file. Since its just a text file edit any way you wish (e.g. vim, emacs, nano).

```
vim .gitignore                  # Use vim to open the file
BoringStuff.xlsx                # Ignore MS Excel file called BoringStuff
*.txt                           # Ignore all .txt files
```

Now that we have excluded the junk. Lets tell git to add the rest to the staging area

```
git status                      # Current state of our directory
git add -A                      # -A add everything (that isn't "gitignored")
git status                      # Git shows us what changes are ready to be committed
# OR -> add the files individually
git status                      # Current state of our directory
git add <filename>              # This will let you to add a file to staging area one at a time
git status                      # Git shows us what changes are ready to be committed
```

Added file to staging area by accident?

```
git status                      # Current state
git reset BadFile.py            # Remove BadFile.py from the staging area
git status                      # Confirm this worked
# OR -> remove all the files and start again
git status                      # Current state
git reset                       # Remove all the files from the staging area
git status                      # Confirm this worked
```

Commit files from the staging area.

```
git status                    # Get a baseline
git commit -m "Initial commit"  # Lets commit, -m is to give it a message, always do this!
git status                    # Check our staging area
git log                       # Now we can see our commit along with its message
```

Remote Repositories

Say we found a project on a remote (e.g., github) that we would like to contribute to.
Copy the remote to local repo.

```
git clone <url-source> <local-destination>   # Get url-source from repo repo
#OR
git clone <local-source> <local-destination> # It is rarely needed but you can setup a remote at a different location on your local machine
```

Info about our remote.

```
git remote -v                 # Provides information about our remote
git branch -a                 # Shows all branches (local and remote)
```

Making edits to a remote repository.
First, make changes to local repository, and commit.

```
git diff                      # Show changes on current copy of code
git status                    # Show the files that have been modified
git add -A                    # Add all the files to staging area
git commit -m "Commit It"     # Now commit the files on local
```

Second, we pull any new changes that were pushed to the remote while we were working on our local copy, merge our changes, and push the updated files to our remote.

```
git pull origin master        # Now we need to pull, in case someone else has made a change
git push origin master        # Now we push to origin-name of remote repository and branch - master
```

We have now successfully added our changes to the remote repository!

Branches

Branches allow us to make speculative changes on potential features without affecting our master file.
To create a branch, and begin working.

```
git branch                    # List out current branches, and shows current branch by (*)
git branch <branch_name>      # This creates a new branch
git branch                    # We can now see our new branch
git checkout <branch_name>    # This moves us from our current branch to the branch we just created
git branch                    # Now you can see we are on the new branch
```

We make our speculative changes, and things look good.
First, we need to commit the changes we made to current branch (directions above).
Then push our branch up to our remote.

```
git push -u origin <branch_name> # Now we have associated our local branch and remote branches
git branch -a                 # This will show our local and remote branches
```

Once we have pushed, we can merge our branch into master if we wish.

```
git checkout master           # Get to our local master branch
git pull origin master        # Lets pull any new changes
git branch --merged           # List out the branches we have merged so far
git merge <branch_name>       # Merge in our branch
git push origin master        # Push our changes to master
git branch --merged           # List out the branches, we should see branch_name here
git branch -d <branch_name>   # Remove the local branch
git branch -a                 # We can see the local branch is gone, but the remote is still there
git branch origin --delete <branch_name> # Now remove the remote branch
git branch -a                 # Now we can see the branches are cleaned up!
```

In certain rare cases, we may want to create an unrelated branch (i.e., an orphan branch)

```
git checkout --orphan newbranch  # This moves you to a new empty branch
git log                       # You will see there are no previous commits
git rm -rf .                  # Remove everything from the current directory and remove from git
# Do some work, so there is something to commit (e.g. touch README.md)
git add <file(s)>             # Add new files to staging
git commit -m "Initial Commit"   # Commit new files and establish commit history
git log                       # Now you can see the history
git branch                    # Now you can see the new orphan branch
git push -u origin <newbranch>   # Now we can push this new orphan branch up to our remote
```

Fixing Common Mistakes

Go back to previous commit.

```
git status                      # Can see the files have changed
git diff                        # Can see the changes
git checkout <filename>         # Lets reset our file to last commit
git status                      # Can see file has been rolled back
git diff                        # Can see there are no changes in file
```

Note: these changes affect the history
Bad commit message.

```
git log                             # Current log
git commit --amend -m "New Message"   # Change commit message
git log                             # New Log, see changed message, but new hash!
```

Forgot to include file.

```
git add <forgotten file>        # Place forgotten file in staging file
git commit --amend              # Going to amend
git log                         # See no new commit
git log --stat                  # See files that were changed in commit
```

Making changes on master rather than feature branch that we had created but forgot

```
git log                         # Lets get the hash of the commit we want to copy
git checkout branch_name        # Go to the branch we were supposed to have committed to
git log                         # Can see we don't have the commit
git cherry-pick hashmaster      # Brings commit onto feature branch
git log                         # See the copied commit

#Now we need to remove the bad commit from master

git checkout master             # Go back to the master branch
git log                         # Get hash of the commit we want
git reset --soft hash           # Commit is gone, files still has changed and are in staging area
git reset hash                  # Mixed (default). Commit gone, files still have changes, but are not in staging area
git reset --hard hash           # Commit gone, tracked files set back to commit
git clean -df                   # This cleans out all untracked (d for directories, f for files)
```

Ran `git reset --hard` *by accident. There may be hope if its <30 days.*

```
git reflog                      # This keeps track of all changes, might see hash
git checkout hash               # This brings you back, but in a "detached head state" (i.e. not on a branch)
git log                         # Can now see the changes are back
git branch branchname           # Add a branch to save that work, if we dont it will get deleted
git branch                      # We can see our current branch and new branch
git checkout master             # Go to master
git branch                      # See all our branches, note that "detached head state" branch is gone
```

These will not modify or delete the history
People have already pulled the file.

```
git log                       # See our commits, get hash
git revert hash               # Allows us to revert,
git log                       # Now can see an additional commit undoing changes
git diff hash1 hash2          # Let us see what git did to maintain history.
```

Setting up SSH keys (Stop Git from asking for my password)

Tired of typing your password? ... Set up an SSH key.

```
ls -al ~/.ssh                 # Check for existing SSH keys
```

You may see some defaults, such as:

- id_dsa.pub
- id_ecdsa.pub
- id_ed25519.pub
- id_rsa.pub

You can either use one of these (if they exist) or *generate a new SSH key*

```
ssh-keygen -t rsa -b 4096 -C "your_email@email.com"   # Generates a new key, using the provided email
```

Enter a file in which to save the key Enter passphrase - A passphrase enables an additional level of security, if your system is compromised. You will need to enter this passphrase every time you work with Git.

Add the key to the ssh-agent

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/<newkeyname>
```

Add key to GitHub account follow this.

You may also need to *switch from HTTPS to SSH* for this to work:

```
git remote -v                                          # This will show the current remotes (you will see HTTPS or SSH)
  # HTTPS look like: https://github.com/<USERNAME>/<REPO>.git
  # SSH look like: git@github.com:<USERNAME>/<REPO>.git
git remote set-url origin git@github.com:<USERNAME>/<REPO>.git   # You will get the SSH from the remote's page
git remote -v                                          # Should see the updated remotes
```

Renaming a GitHub Repository

On github

```
Your Repo > Settings > Repository name
```

Links to the previous name of your repository will be forwarded to your new repository. However, for clarity its recommend that you update your links by:

```
git remote set-url origin <newurl>
```

Commited a Large File by Accident

If you accidentally committed a large file to your git history, but are now having an issue pushing your commit due to Github's maximum file size of 100MB.

When you push you will see an error like this:

```
remote: error: GH001: Large files detected. You may want to try Git Large File Storage - https://git-lfs.github.com.
remote: error: Trace: b310dac0473fe3f34910f9b68bd885fd
remote: error: See http://git.io/iEPt8g for more information.
remote: error: File path/to/your/file is 427.54 MB; this exceeds GitHubs file size limit of 100.00 MB
To github.com:username/repository.git
 ! [remote rejected] master -> master (pre-receive hook declined)
error: failed to push some refs to git@github.com:username/repository.git
```

Note that if you add this file to .gitignore and try committing this will not solve your problem as the issue is with your very first commit (the one before you realized you have that large file).

To fix this issue you need to not only .gitignore the large file, but also purge this file from your commit history so you can push your work and be up to update on your remote. You do this by going back in your tree and forcefully removing the file.

```
git filter-branch --tree-filter 'rm -rf path/to/your/large/file' HEAD
```

If you need to do this for more than one file, you will need to add a -f, to tell github to force overwrite the backup it made when you performed the previous filter and removal, like this:

```
git filter-branch -f --tree-filter 'rm -rf path/to/your/other/large/file' HEAD
```