# git Primer

## History

git became the de facto VCS for Crew since 2008 when DDS (the second? third? attempt) was under way.

## Setup

Before you do any thing, you need to write down your information somewhere. That place is `~/.gitconfig`.

Mine looks like this:

```
[core]
    editor = vim
[user]
    email = johnny@example.com
    name = Johnny Appleseed
[alias]
    br = branch
    ci = commit
    co = checkout
    cl = clone
    st = status
```

## Downloading the Source

The generalized form for downloading a repository is like this:

```
$ git clone <GIT_URL>
```

A GIT_URL may look like:

```
/path/to/a/repo/.git
git@crew-git.ccs.neu.edu:backend.git
git://crew-git.ccs.neu.edu/backend.git

rsync://host.xz/path/to/repo.git/
https?://host.xz[:port]/path/to/repo.git/
git://host.xz[:port]/path/to/repo.git/
ssh://[user@]host.xz[:port]/path/to/repo.git/
```

## Using the Source

### Basic Use.

|  |  |  |
|---|---|---|
| 10 | Edit | Edit your source code |
| 20 | **`git status`** | See what files have been changed/removed/added |
| 30 | **`git add <NEW_FILES/CHANGED_FILES>`** | Add the files to be committed (staging). |
| 30 B | **`git add -u`** | Adds all files to the staging area that are currently tracked. |
| 40 | **`git diff HEAD`** | See what you are going to commit. If error, goto 10 |
| 45 | Test your code | Or goto 10 and write tests. |
| 50 | **`git commit`** | Commit your code. Write a detailed, beautiful log message.<br>NOTE that log messages should be in the form<br>"This commit will...**X**"<br>(where **X** is the changes)<br>Example commit message: "Update the command line arguments." |
| 60 | **`git push`** | Push your code to the remote server (branch).* |
| 70 | goto 10 |  |

For the first time you may have to do `git push origin master` (replace master with the branch you want to push)

## Normal Use.

### Using branches.

Okay, this is what you actually want. What git became famous for – cheap (and dirty) branches.

First, look at what you have with `git branch`

You can fork from the current branch you are on by `git checkout -b <BRANCH_NAME>`

Delete a branch `git branch -D <BRANCH_NAME>`

Delete a remote branch `git push origin :<REMOTE_BRANCH_NAME>`

### Checking out a remote branch.

First, ~~switch into~~ checkout the remote branch `git checkout origin/unstable`

Create a branch off the remote branch by `git checkout -b unstable`

## Advanced Use.

### git rebase

If you are making a bunch of small changes, you might want to try rebasing. Some people may call you a nut for writing one line of code, committing, writing one line of code, committing, etc., but we don't like to judge.

NO PUSHING

It is critical that you DO NOT PUSH to your remote branch or publish the branch that is going to be rebased (it fucks up the other person's history).

So, make a bunch of commits, then `git rebase -i HEAD~<NUMBER_OF_REVISIONS_BACK>`

Your choice of editor should open with something like this:

**Rebasing**

```
pick 74a889d typo 1
pick 5baaf52 typo 2
pick 4d02941 typo 3
pick 0126d87 typo 4
pick 0e52c27 typo 5

# Rebase 563a038..0e52c27 onto 563a038
#
# Commands:
#  p, pick = use commit
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

`pick` the commits you want.

Be careful when you delete lines, as those commits will become unreachable (technically).

`squash` obvious, small bugfixes. Who knew history could be fun! It will merge into the previous commit.

`edit` will stop the rebase and allow you to change the commit message (even change the commit itself, but don't do that unless you know what you are doing).

```
pick 74a889d typo 1
s 5baaf52 typo 2
s 4d02941 typo 3
s 0126d87 typo 4
s 0e52c27 typo 5

# Rebase 563a038..0e52c27 onto 563a038
#
# Commands:
#  p, pick = use commit
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Save, close the editor and we get:

```
# This is a combination of 5 commits.
# The first commit's message is:
typo 1

# This is the 2nd commit message:

typo 2

# This is the 3rd commit message:

typo 3

# This is the 4th commit message:

typo 4

# This is the 5th commit message:

typo 5

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   README
#
```

Save, close the editor.

### git fetch

TODO

### git merge

TODO

## Putting your own crap in a repository.

So, it might be a good idea to put your own source code (for homework, fun, etc.) under version control. How else can you muse about your failures as a programmer?

First, setup a bare repository in a location (path) that's not likely to move.

```
$ mkdir your_crap.git
$ cd your_crap.git
$ git init --bare
```

Now you should have a bare repository primed for pushing/pulling. Create a working copy by cloning it.

```
$ git clone /path/to/your_crap.git
```

It should create a directory your_crap. Voilà!

## Links

- A presentation by MIT's SIPB group on understanding git http://web.mit.edu/cluedumps/slides/understanding-git-2008.pdf. Very well done.
- Git Snippets, a good place to get bootstrapped (Check out "pushing and pulling" under beginner)
- http://cheat.errtheblog.com/s/git
- http://git-scm.com/
- This site has an illustrative picture of git's structure