15 Developer Tools: Git

Git helps you track changes in a project and undo them when something goes wrong. Git helps teams share and manage a project, especially as the team size grows.

> "Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later."
>
> — **Scott Chacon**, **Pro Git**

Git was built by software developers for software developers but can be used to share any type of project. In fact, Bloc uses Git to track content.

Installation

It is best to know how to use Git on the command line. Here's how to install Git for each major operating system:

Operating System	Guide
OS X	Download and install **Git for OS X**.
Windows	Git BASH (or Cloud9 for Rails students) comes with Git pre-installed, so there's no need to download it separately.
Linux	Refer to **this** installation guide.

Git Configuration, Your Identity

We need to set some global settings on our Git installation. Execute the following commands:

Terminal

```
$ git config --global user.name "Your Name"
$ git config --global user.email yourname@example.com
$ git config --global color.ui true
$ git config --global push.default simple
```

We recommend configuring Git to use the "simple" configuration for pushing which can be explained in more detail **here**.

These settings are reflected in the `.gitconfig` file in our `HOME` directory. We can verify this by inspecting the `.gitconfig` file:

Terminal

```
$ cat ~/.gitconfig
[user]
        name = Your Name
        email = yourname@example.com
[push]
        default = simple
```

Repositories

A repository is a collection of files under version control. We choose what to put in a repository (or under version control). Repositories are like a shopping cart at a grocery store. The grocery store contains all possible items we could potentially buy, but we only want certain things in our shopping cart. We can pick what to put in it. The grocery store in our example correlates to the directory on our computer's filesystem that is under version control. We get to choose what files or sub-directories we want to put in the repository.

Initializing a Repository

Start by creating a project that we will use as our practice repository. Create a directory named `bloc` if you don't already have one and a sub-directory in `bloc` named `shopping_cart`:

~

```
$ cd ~
$ mkdir -p bloc/shopping_cart
```

Change directories to the `shopping_cart` directory:

~

```
$ cd bloc/shopping_cart
$ pwd
/Users/<YOUR_USERNAME>/bloc/shopping_cart
```

We have nothing in our `shopping_cart` directory at this point. Create a file using `touch` and name it `apple.txt`:

~/bloc/shopping_cart

```
$ touch apple.txt
$ ls
apple.txt
```

Currently, `apple.txt` is not under version control, but we can use Git to track the changes to this file and our `shopping_cart` directory over time.

Initialize this directory as a repository by using `git init`:

~/bloc/shopping_cart

```
$ git init .
Initialized empty Git repository in /Users/yourusername/bloc/shopping_cart/.git/
```

We can verify that the repository was initialized properly by doing an `ls -al`. We see a hidden directory named `.git` in the `bloc/shopping_cart` directory:

~/bloc/shopping_cart

```
$ ls -al
total 0
drwxr-xr-x   4 yourusername  staff  136 Mar 17 16:10 .
drwxr-xr-x   5 yourusername  staff  170 Mar 17 16:02 ..
drwxr-xr-x  10 yourusername  staff  340 Mar 17 16:11 .git
-rw-r--r--   1 yourusername  staff    0 Mar 17 16:07 apple.txt
```

Git Commands

Git Status

We can also use the `git status` command to verify the repository was initialized properly. The output of the command shows current information about a repository:

~/bloc/shopping_cart

```
$ git status
-On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    apple.txt

nothing added to commit but untracked files present (use "git add" to track)
```

The output shows `apple.txt` is "untracked", developers will commonly refer to an untracked file as "not being checked in".

Git Add

The `shopping_cart` directory is now a repository but we still don't have any files checked in. Using our running analogy, we are at the grocery store and we have our shopping cart,

but we haven't put anything in it yet! We must tell Git which files we want to put under version control. Use the `git add` command to put `apple.txt` under version control:

~/bloc/shopping_cart

```
$ git add apple.txt
```

Run `git status`:

~/bloc/shopping_cart

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   apple.txt
```

We *staged* `apple.txt`. Using our grocery store example, the staging state would be analogous to putting an item in the shopping cart before checking out. We still have time to think about whether we really want to buy this item from the store before we actually pay for it. Anything that is not staged will not be committed, just as anything we didn't put in our shopping cart will not be bought.

Let's create another file named `banana.txt` using `touch`:

~/bloc/shopping_cart

```
$ touch banana.txt
$ ls
apple.txt  banana.txt
```

Run `git status` again:

~/bloc/shopping_cart

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   apple.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        banana.txt
```

Notice from the result of the `git status` command we issued, the `banana.txt` file is *unstaged*. The file is in the `shopping_cart` directory, but not under version control.

Git Commit

Now that `apple.txt` is under version control, we have the ability to take snapshots of the file at any given time. To take a snapshot of the `apple.txt` file we use the `git commit` command:

~/bloc/shopping_cart

```
$ git commit -m "Create the apple.txt file"
[master (root-commit) 5b1d5c8] Create the apple.txt file
 1 file changed, 0 insertions(++), 0 deletions(-)
 create mode 100644 apple.txt
```

By issuing the commit command, we have instructed Git to save the current state of the `apple.txt` file. Committing is like checking out at the grocery store in that once the item is committed it is a part of the permanent collection of items in our shopping cart when we leave the store. Also take note of the tense we used for the commit message, "Create the apple.txt file". We use the *imperative mood*. This philosophy is drawn directly from the **Git guidelines**:

> Describe your changes in imperative mood, as if you are giving orders to the codebase to change its behavior; for example, "make xyz do abc" instead of "[This patch] makes xyz do abc" or "[I]

> changed xyz to do abc". Make sure your explanation can be understood without external resources.

Let's stage and commit `banana.txt`:

~/bloc/shopping_cart

```
$ git add banana.txt
$ git commit -m "Add banana.txt"
```

Git Log and Git Show

We can visualize the commit by using another command named `git log`:

~/bloc/shopping_cart

```
$ git log
commit 70914a2f007f13fa0b27676fb8e9512157b324d4
Author: yourusername <yourusername@example.com>
Date:   Mon Jun 1 15:30:51 2015 -0700

    Add banana.txt

commit 30fef231aa765caaac7bd0d84d69c09aa82ab03b
Author: yourusername <yourusername@example.com>
Date:   Mon Jun 1 15:23:08 2015 -0700

    Create the apple.txt file
```

The Git log shows our two commits. Inspect the latest commit by issuing the Git show command:

~/bloc/shopping_cart

```
$ git show --pretty="format:" --name-only HEAD

banana.txt
```

The output shows `banana.txt` as the only file in the commit. The gnarly strings `30fef231aa765caaac7bd0d84d69c09aa82ab03b` and

`70914a2f007f13fa0b27676fb8e9512157b324d4` are the unique identifiers for each commit called the `hash`. The `hash` that you see will be different from this one, but the important concept is that each commit has a unique identifier. The `hash` is similar to the receipt you get when you check out of the grocery store. It is an identifier for the groceries that you have decided to buy that has meta information like date and time.

Using .gitignore to Ignore Files

A grocery store has many items that we don't want to bring home such as tomatoes. Let's imagine that when we got our shopping cart, it already had tomatoes in it. We're too lazy to remove the tomatoes from our cart so we ignore them when we checkout. Similarly, in Git, you can harness the power of the `.gitignore` file to ignore any files that you may not want under version control. Create a file named `tomatoes.txt` in the `shopping_cart` directory:

~/bloc/shopping_cart

```
$ touch tomatoes.txt
```

Run `git status`:

~/bloc/shopping_cart

```
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        tomatoes.txt

nothing added to commit but untracked files present (use "git add" to track)
```

`tomatoes.txt` is untracked, but we never want to add it and we don't want Git to complain. Create a `.gitignore` file to do just that:

~/bloc/shopping_cart

```
$ touch .gitignore
$ echo "tomatoes.txt" >> .gitignore
```

Perform another `git status`:

~/bloc/shopping_cart

```
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

Git no longer complains about `tomatoes.txt` being untracked, but it does see there's a new file named `.gitignore`. Add the `.gitignore` file to the repository:

~/bloc/shopping_cart

```
$ git add .gitignore
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore

$ git commit -m "Add .gitignore"
[master 5888285] Add .gitignore
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

Git now knows to both ignore `tomatoes.txt` and also add the `.gitignore` file to our repository.

Git Remove

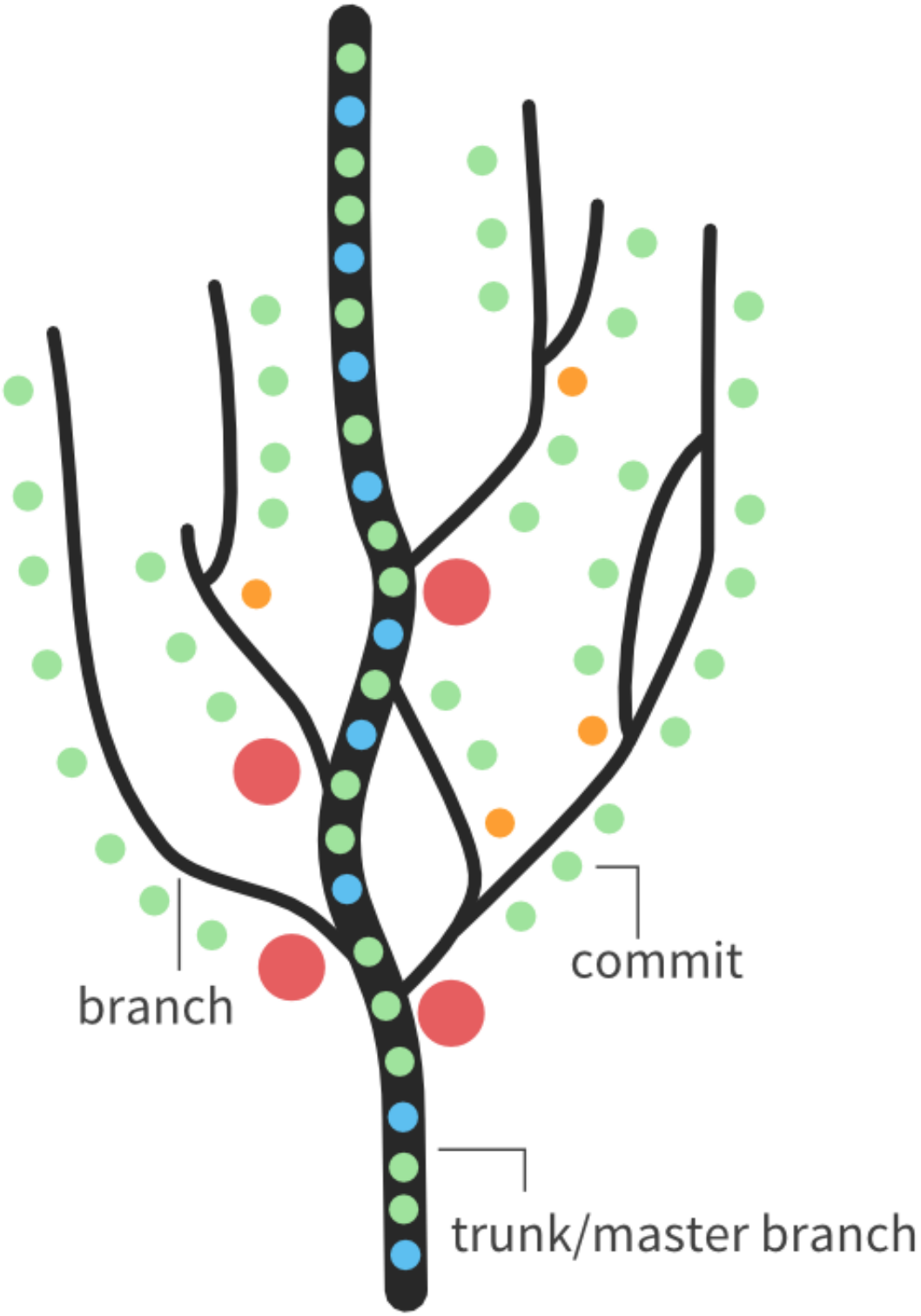
Git's `rm` command can be used to remove items from our repository. For example, if we wanted to remove a file named `rotten_apples.txt`, we would run the following command (don't actually run this command):

~/bloc/shopping_cart

```
$ git rm rotten_apples.txt

On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    rotten_apples.txt
```

Branches

A simple way to visualize a Git repository is to imagine a tree. Our main branch, usually master, is the trunk of the tree.



Repositories default to the `master` branch. It is the official record of production-ready code and shouldn't include any bugs or dysfunctional code. When introducing a new feature or experimenting with code, it is a good idea to work on a separate branch. A branch provides an isolated environment to work in without disturbing existing code.

To create a new branch, use the Git checkout command with the `-b` option to specify the

branch. This command will create the specified branch, and switch to it at the same time. Create a new branch on our local repository named `new_grocery_list`:

~/bloc/shopping_cart

```
$ git checkout -b new_grocery_list
Switched to a new branch 'new_grocery_list'
```

We can view all the existing branches by issuing the `git branch` command:

~/bloc/shopping_cart

```
$ git branch
  master
* new_grocery_list
```

We created a branch named `new_grocery_list` as well as "checked it out", which effectively means we switched to that branch. By switching to the `new_grocery_list` we are operating on the branch without affecting the `master` branch, it is completely independent. When created, the `new_grocery_list` branch contains the same files as the `master` branch. Execute the `ls` command on the command line to illustrate this:

~/bloc/shopping_cart

```
$ ls
apple.txt  banana.txt  tomatoes.txt
```

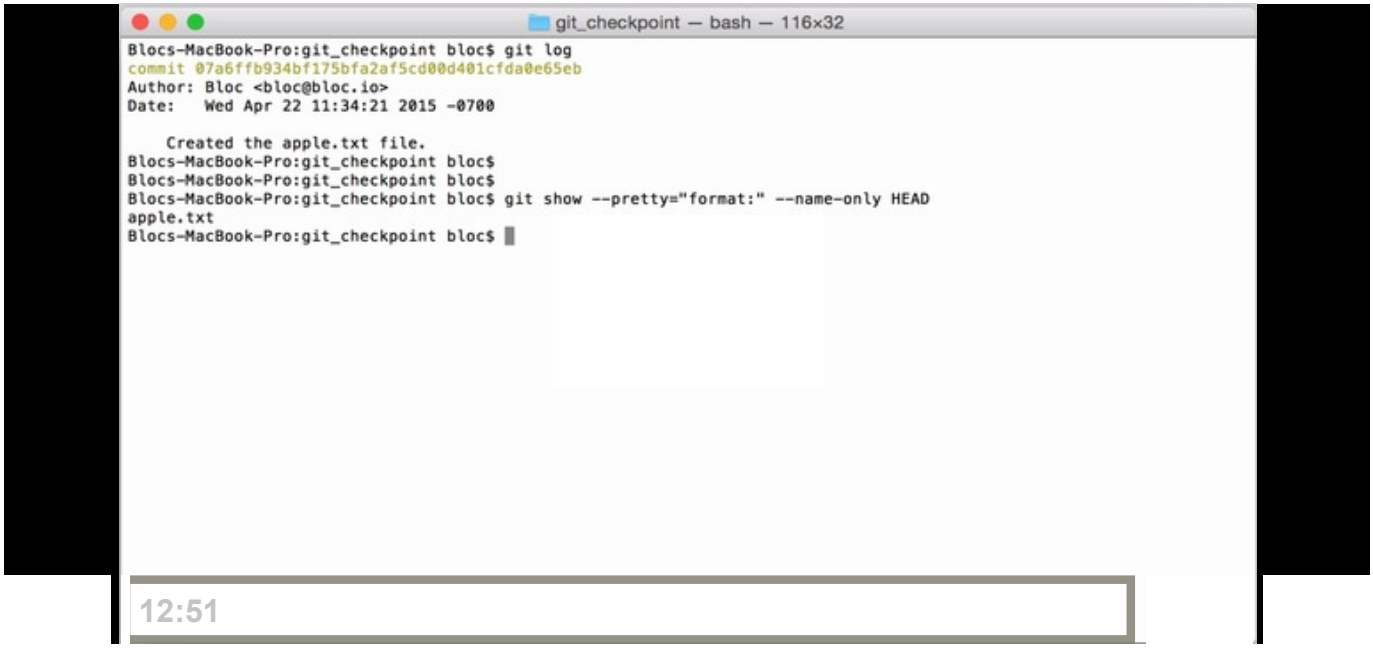
Note that the contents of the directory are unchanged. Use the following commands to create a file named `orange.txt` in the `new_grocery_list` branch, add it, and then commit it:

~/bloc/shopping_cart

```
$ touch orange.txt
$ git status
On branch new_grocery_list
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        orange.txt

nothing added to commit but untracked files present (use "git add" to track)
$ git add orange.txt
$ git status
On branch new_grocery_list
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   orange.txt

$ git commit -m "Add orange.txt on new_grocery_list"
[new_grocery_list 1064da0] Add orange.txt on new_grocery_list
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100755 orange.txt
```

Help

- **GitHub has extensive resources**
- **Interactive command line on GitHub**
- Tutorial Video Below



15. Developer Tools: Git

Assignment	Discussion	Submission

- From master, create a new branch named `shopping_cart_assignment`:

~/bloc/shopping_cart

```
$ git checkout master
$ git checkout -b shopping_cart_assignment
```

- Remove `apple.txt` from the repository using `git rm`:

~/bloc/shopping_cart

```
$ git rm apple.txt
```

- Add the text "Git Checkpoint Assignment." to `banana.txt`.
- Create a file named `lettuce.txt` and add the output of the `git diff`

command as the contents:

~/bloc/shopping_cart

```
$ git diff > lettuce.txt
```

- Commit the changes you have made.
- Run `cat lettuce.txt` to print the contents of `lettuce.txt`.
- Copy the output into a Markdown code block and send it to your Mentor for review.

assignment completed

?