Git and GitHub

What?

Git: software which keeps track of code changes

GitHub: a popular server for storing repositories

Why?

- Keeps a full history of changes
- Allows multiple programmers to work on the same codebase
- Is efficient and lightweight (records file changes, not file contents)
- Public repositories on GitHub can serve as a coding resume.

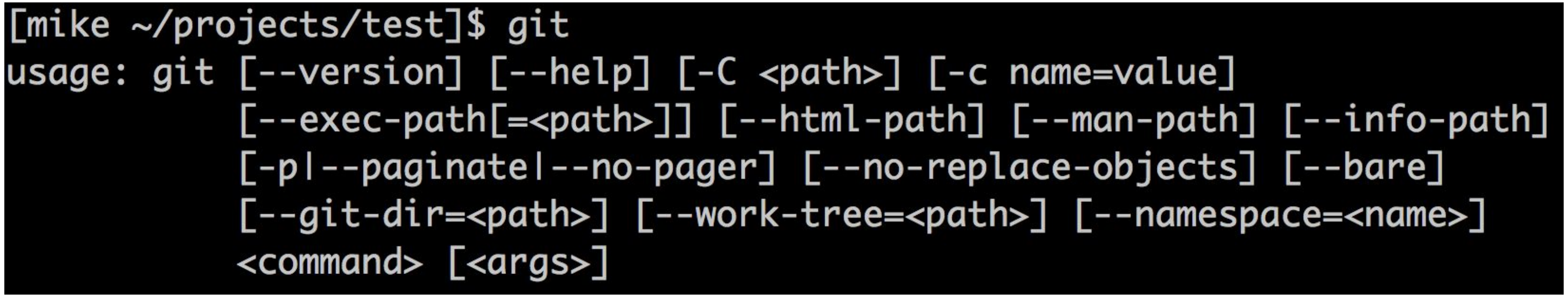
How?

That’s what this session is for!

Setting up Git and GitHub

Installing Git on your machine

Git may already be installed. Check to see by running `git` in a terminal.



```
[mike ~/projects/test]$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

Otherwise:

Mac:

`brew install git`

Ubuntu:

`apt-get install git`

Windows:

http://git-scm.com/downloads

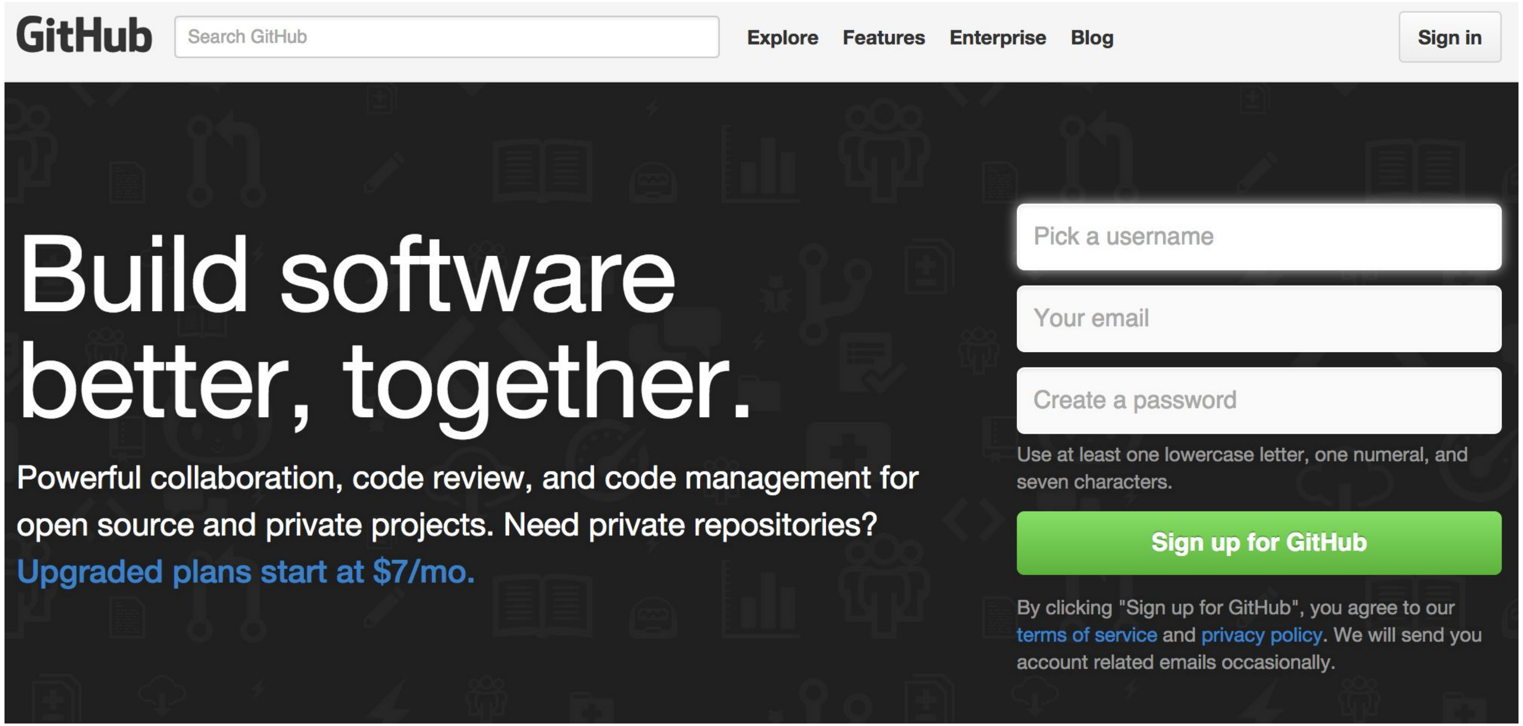
Default install options.

Open the installed Git Bash.

If these don't work for any reason, try the other options at:
http://git-scm.com/book/en/Getting-Started-Installing-Git

Creating a GitHub account: https://github.com/

(choose a job-appropriate user name if possible)



GitHub

Search GitHub

Explore Features Enterprise Blog

Sign in

Build software better, together.

Powerful collaboration, code review, and code management for open source and private projects. Need private repositories? Upgraded plans start at $7/mo.

Pick a username

Your email

Create a password

Use at least one lowercase letter, one numeral, and seven characters.

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.

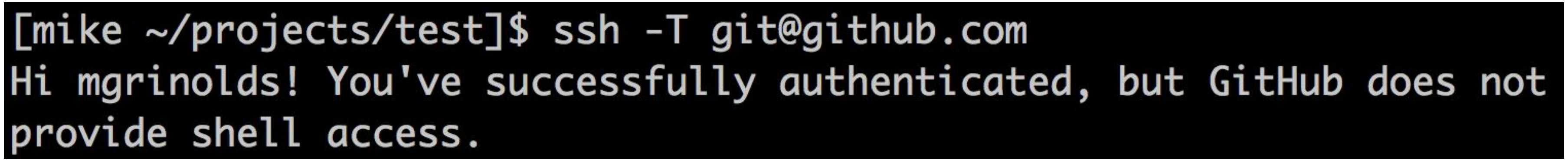
Creating a pair of SSH keys

- SSH keys are one way of securely communicating with GitHub.
- There'll also be used for logging into your eventual web-app servers.

Step-by-step directions at:

https://help.github.com/articles/generating-ssh-keys

When all is well:



```
[mike ~/projects/test]$ ssh -T git@github.com
Hi mgrinolds! You've successfully authenticated, but GitHub does not
provide shell access.
```

And you're ready to start using Git and GitHub.

Basic git configuration

User info, so GitHub recognizes your commits:

```
git config --global user.name “Mike Grinolds”

git config --global user.email “mgrinolds@gmail.com”
```

These global config settings live in ~/.gitconfig and can also be manually edited there.

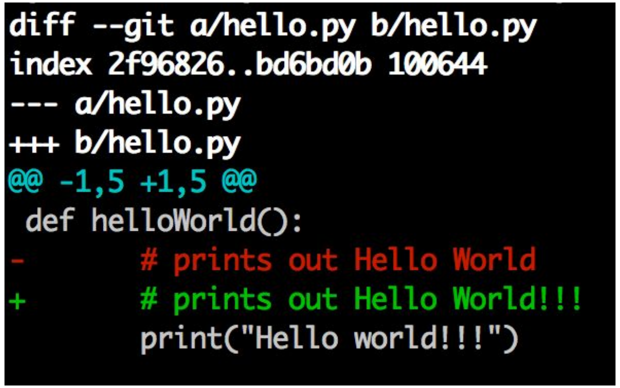
Git concepts and vocabulary

Change, Commit, Branch, Repository

aka Diff

One of two things:

1. File creation, renaming, or deletion.
2. Insertion or deletion of a line in a file (a modified line is both an insertion and a deletion)



```
diff --git a/hello.py b/hello.py
index 2f96826..bd6bd0b 100644
--- a/hello.py
+++ b/hello.py
@@ -1,5 +1,5 @@
 def helloWorld():
-       # prints out Hello World
+       # prints out Hello World!!!
        print("Hello world!!!")
```

Change, **Commit**, Branch, Repository

A series of changes that records incremental updates to a series of files.

Has a global unique hash (calculated from contents of file) that serves as an identifier.

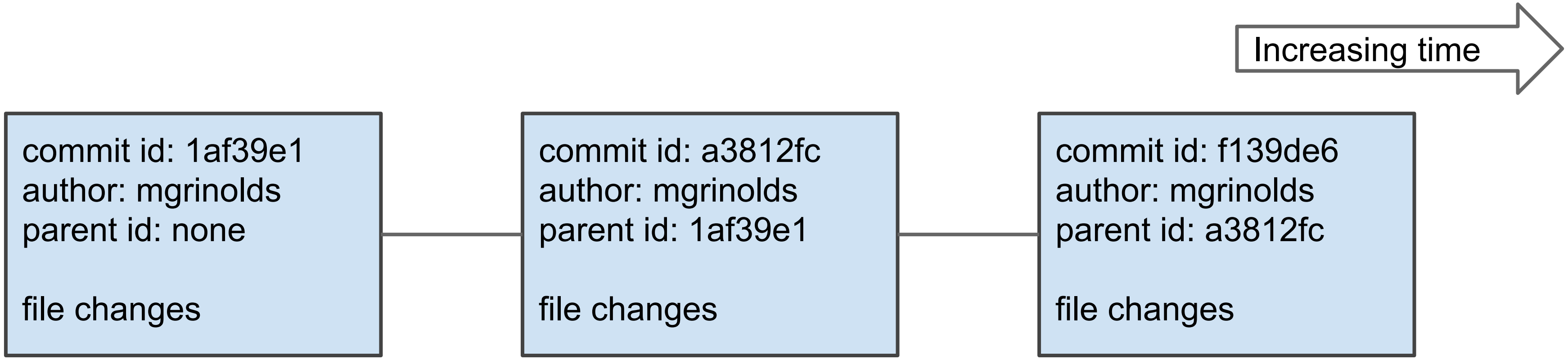
commit id: 1af39e1
author: mgrinolds

file changes

Change, Commit, **Branch**, Repository

A linear series of commits.

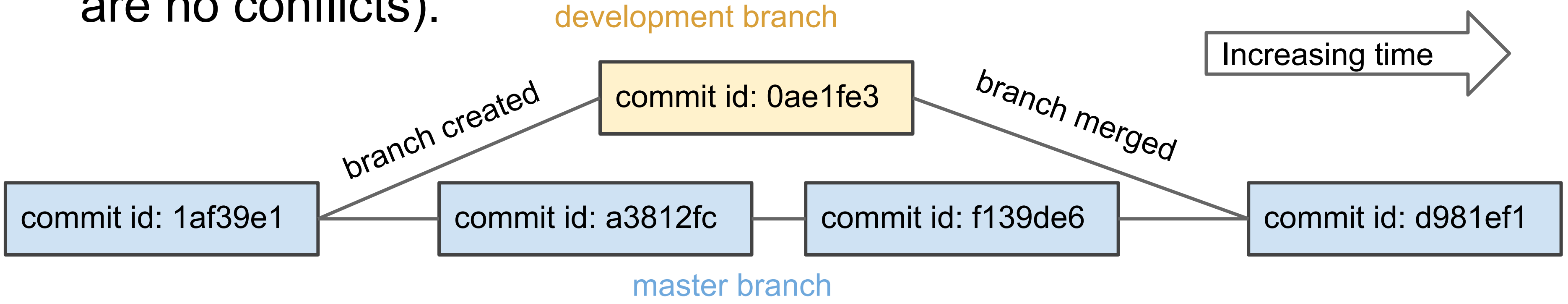
A codebase can be calculated by applying changes to files in each commit in succession.



Change, Commit, Branch, **Repository**

A tree structure containing many branches. Each branch represents a different state of the code.

Branches can be formed at any commit, and two branches can be merged together by summing their changes (assuming there are no conflicts).



Basic Git tutorial

Initializing a local repository

Create a folder with the project name (`git-demo`)

```
mkdir git-demo
cd git-demo
git init
```

Create an example code file

Create a python file `hello.py`

(`nano hello.py`, `vim hello.py`, or using your favorite application)

Write some (very simple) code

Commit the changes to the git repo:

```
git add .
git commit -m “Initial commit”
```

`hello.py`

```
def hello():
    print (“Hello World”)

if __name__ == “__main__”:
    hello()
```

Pushing the repo to GitHub

Create a repo with the same name on GitHub: https://github.com/new, don't initialize with a Readme.

Copy the ssh remote link (`git@github.com:mgrinolds/git-demo.git`)

Configure remote connection locally:

```
git remote add origin git@github.com:mgrinolds/git-demo.git
git push origin master
```

Check to see if the file is added on GitHub.

A couple conventions worth noting:

`origin` : the name for a remote repository
`master` : the name of the primary branch

Ignoring files in the repository

Make a file named `.gitignore`.

Files that match expressions in this file are not revision controlled. Add the line `credentials.json` to this file.

```
git add .
git commit -m “Adding .gitingore”
git push
```

Now create `credentials.json`, and it’s not tracked.

Making a (new) branch

`git branch add-excitement`

`git checkout add-excitement`

(`git checkout -b add-excitement` does both of these)

Modify hello.py

`git add .`

`git commit`

For saving work:

`git push --set-upstream origin add-excitement`

`hello.py`

```
def hello():
    print (“Hello World!!!”)

if __name__ == “__main__”
    hello()
```

Merging a branch (fast forward)

Go back to the master branch:

```
git checkout master
```

```
git merge add-excitement
```

hello.py

```
def hello():
    print (“Hello World”)

if __name__ == “__main__”
    hello()
```

hello.py

```
def hello():
    print (“Hello World!!!”)

if __name__ == “__main__”
    hello()
```

Merging a branch (resolvable conflict)

Make a change in `master`:

(and commit)

`hello.py`

```
def hello():
    # prints “Hello World!!!”
    print (“Hello World!!!”)

if __name__ == “__main__”
    hello()
```

Checkout other branch:

```
git checkout add-excitement
```

Make a change in `add-excitement`:

(and commit)

`hello.py`

```
def hello():
    print (“Hello World!!!”)

if __name__ == “__main__”
    hello()
    hello()
```

Merging a branch (resolvable conflict)

```
git checkout master
git merge add-excitement
```

hello.py

```
def hello():
    # prints “Hello World!!!”
    print (“Hello World!!!”)

if __name__ == “__main__”
    hello()
    hello()
```

Merging a branch (unresolvable conflict)

Make a change in `master`:

(and commit)

Checkout other branch:

```
git checkout add-excitement
```

Make a change in add-excitement:

(and commit)

`hello.py`

```
def hello():
    # prints “Hello World”
    print (“Hello World”)

if __name__ == “__main__”
    hello()
    hello()
```

`hello.py`

```
def hello():
    # prints “Hello World!!!!”
    print (“Hello World!!!!”)

if __name__ == “__main__”
    hello()
    hello()
```

Merging a branch (unresolvable conflict)

```
git checkout master
git merge add-excitement
```

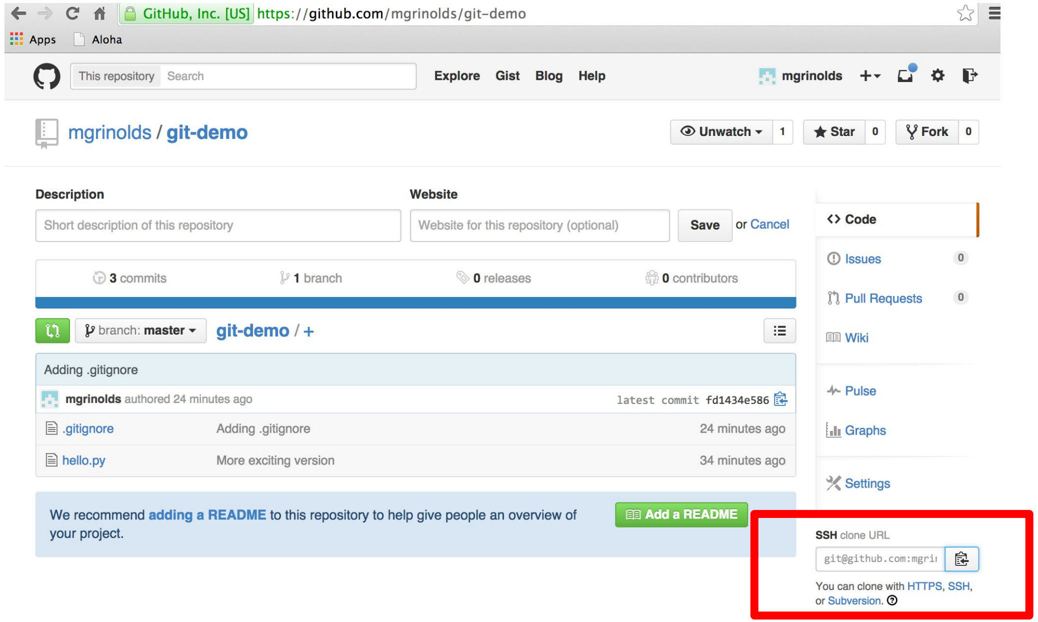
Resolve the conflict by manually editing the file and choosing code from the branch (or branches) you want to use.

Git use `<<<<<<` characters to denote conflicts. Delete these.

```
git add .
git commit -m “Fixing merge conflict in hello.py”
```

Cloning other GitHub repositories

Find the clone SSH link for a repo: Example at https://github.com/mgrinolds/git-demo



Adding .gitignore
mgrinolds authored 24 minutes ago — latest commit fd1434e586

.gitignore	Adding .gitignore	24 minutes ago
hello.py	More exciting version	34 minutes ago

We recommend adding a README to this repository to help give people an overview of your project.

You can clone with HTTPS, SSH, or Subversion.

Cloning other GitHub repositories

Go to your projects directory (`cd ..` from your repo)

For example at https://github.com/mgrinolds/git-demo:

```
git clone git@github.com:mgrinolds/git-demo.git

cd git-demo
```

The files from the remote repo should be there.

Pulling updates from a repository

After a remote repository is modified:

```
git fetch origin
git merge origin/master
```

(experts can consider `git rebase origin/master` as well)

(`git pull origin/master` does both commands together)

Pushing updates to a repository

`git push origin` works if the remote branch has no extra commits (and can be fast forwarded).

If the remote is ahead of local, first merge in remote:

```
git fetch origin
git merge origin/master
git push origin
```

FAQ

General advice

I’m worried I might do something that will lose files/changes, what should I do?

Make a copy of the entire repository directory, then feel free to experiment :-).

for complex changes I like following: http://sethrobertson.github.io/GitFixUm/fixup.html

I have uncommitted changes and I want to move to a new branch, can do I do this without committing?

Sort of. There is a repository-global stash, which is a stack (first-in last-out) storage for uncommitted changes. There are a few situations where it’s easy to lose work; however the following is okay:

```
git stash  # stores all uncommitted changes in the stash
git checkout <branch-name> # this is now doable
# do whatever work needs to be done
git checkout <original-branch>
git stash pop # original save changes applied to branch
```

You *can* have multiple diffs stashed; however I wouldn’t recommend it. You’ll likely forget the order they’ll need to be popped off of, and you could pop the stash in the wrong branch. Also for safety, only stash pop if there are no uncommitted changes - the changes will be merged, and there’s good way to undo that.

Undoing things

How do I undo uncommitted changes to a file?

```
git checkout <filename>
```

How do I undo all uncommitted changes?

```
git checkout .
```

How do I unstage changes?

```
git reset HEAD
```

How should I undo changes in a commit?

You do a second commit which undoes the first commit: `git revert <commit-hash>`

How do I return to a to a previous commit?

Checkout the commit you want, then create a new branch.

```
git checkout <commit-hash>
git checkout -b <my-branch-name>
```

Best practices

When should I commit?

Ideally you commit “working” code, so that you can return to a working state if necessary. Also, each commit should have one logical task that can be summarized in one phrase.

When should I push?

Always right after you commit. Why wouldn’t you want to have an immediate back up?

Do I have to have commit messages?

You at least need a descriptive title. This is an important part of code documentation (especially in a multi-developer environment).

Tools

What’s a good tool for viewing the Git tree?

http://www.sourcetreeapp.com/

gitk (should be installed with git)

What’s a good tool for resolving merge conflicts?

http://kdiff3.sourceforge.net/