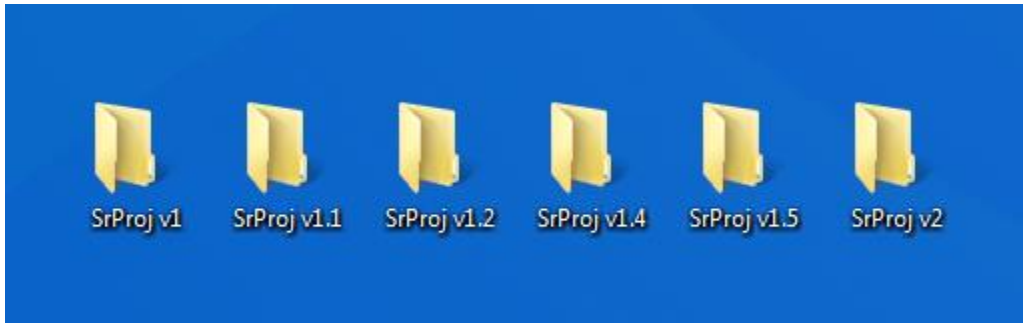
Revision Control with git

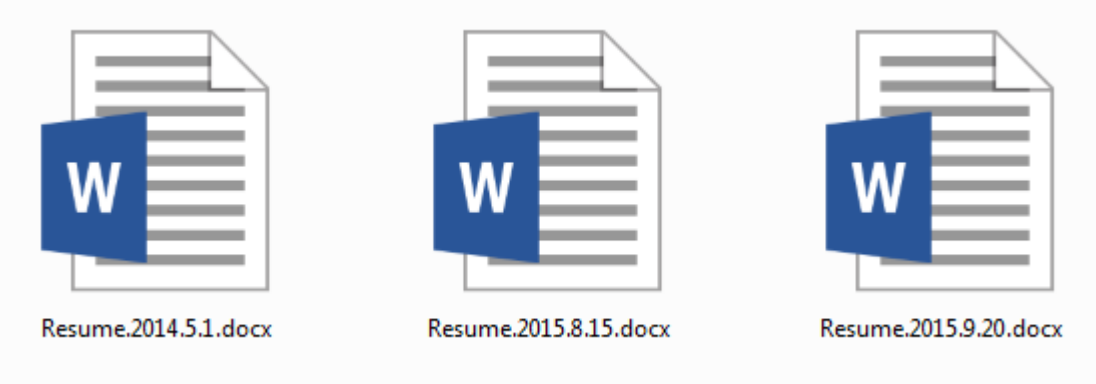
Blake Bourque

10/16/15

3pm-6pm

The “Save As…” Method





This is a problem!

What is Git?

- Noun
  - an unpleasant or contemptible person.

- Proper Noun
  - a widely used version control system for software development. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. Git was initially designed and developed by …

What does git do?

- Time Travels
- Droboxes
- Forking and Un-forking
- Oops. Ctrl-Z
- Track Changes & lay blame
- Backup and Restore
- Synchronization & Collaboration
- Branching and merging
- Undo
- Track Changes & Ownership

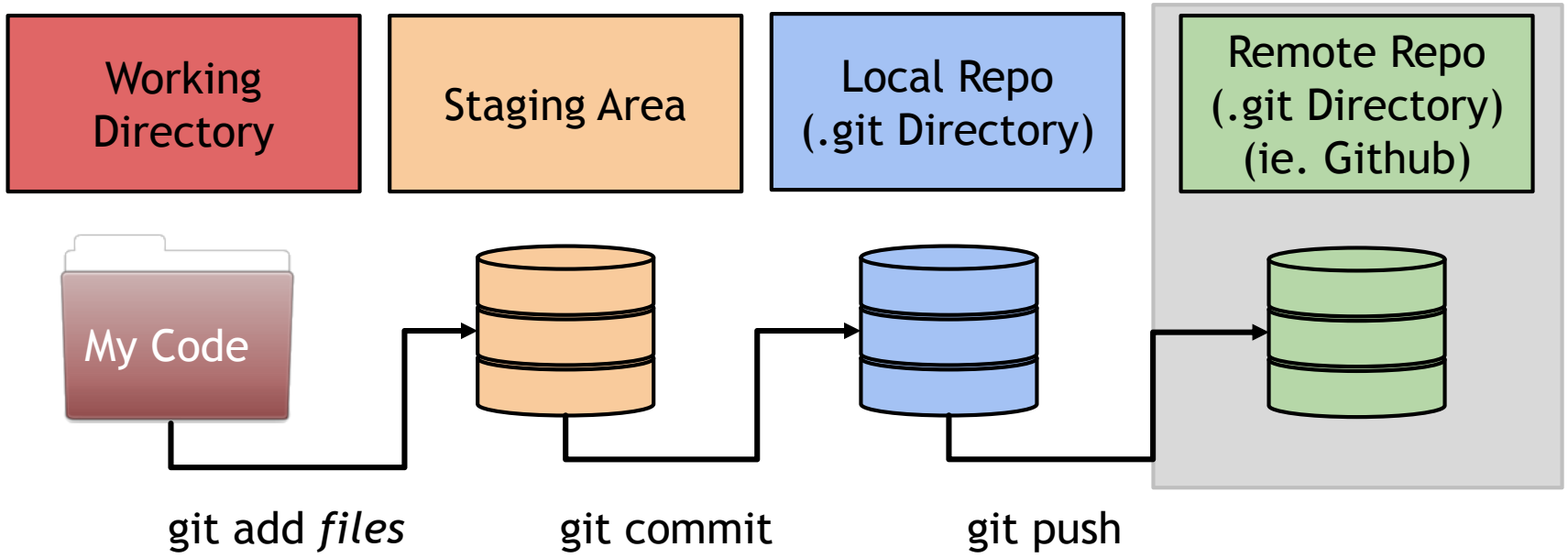
How is git better than […]?

- Cheap Local Branching
- Everything is local
- git is fast
- git is small
- Staging area
- Distributed
- Any Workflow
- Github
- Easy to learn

http://LearnStemLabs.com/articles/whygitisbetter

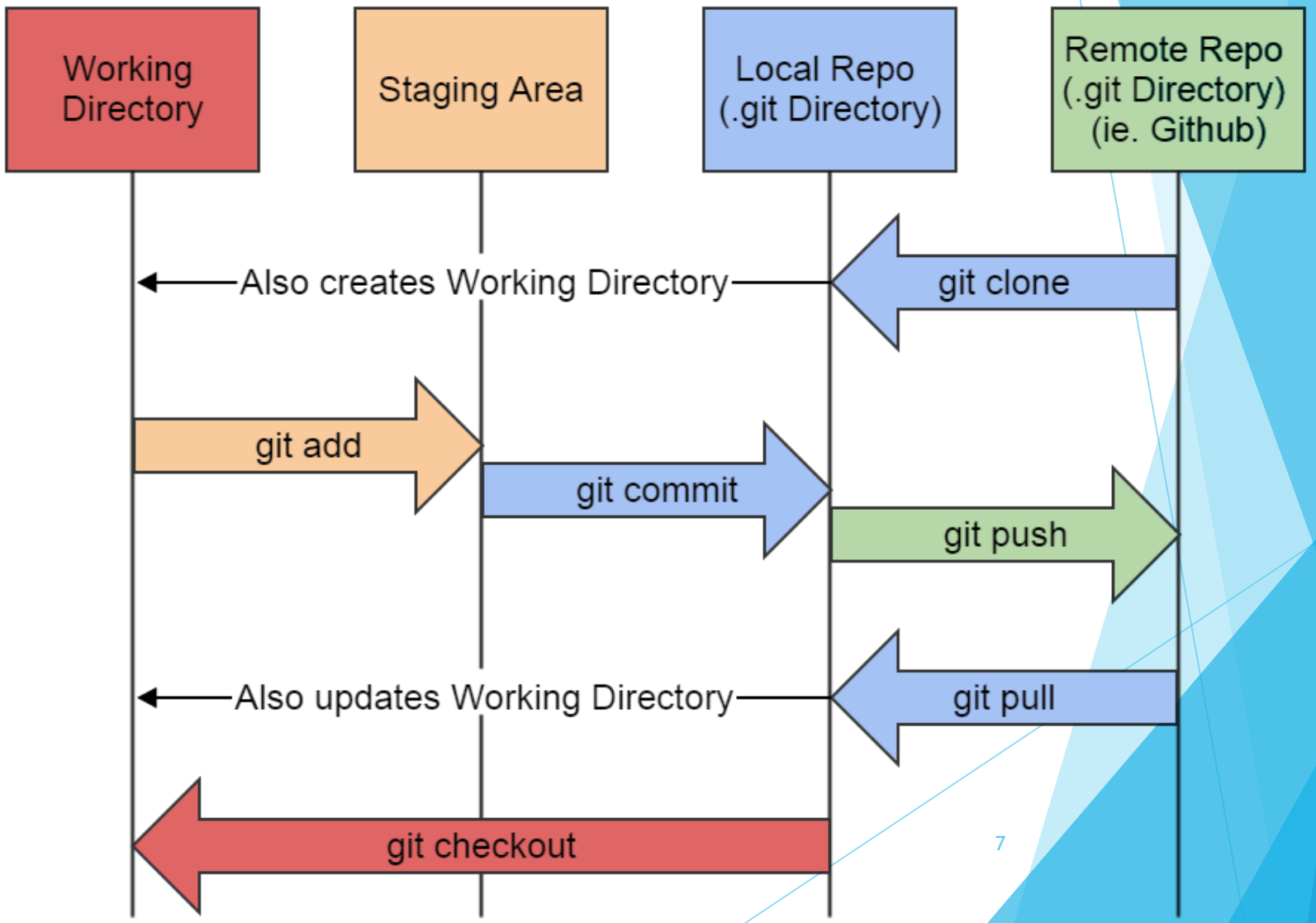
git “areas”

Where your code lives



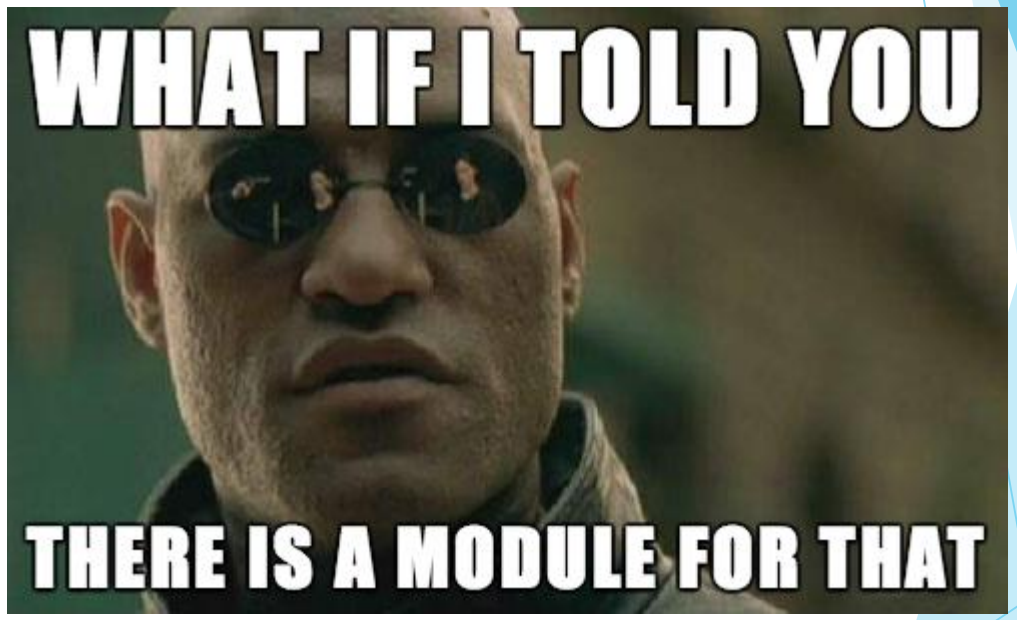
Understanding the a git flow

- $ git clone -or- git init
- $ <make changes>
- $ git add
- $ git commit
- $ git push
- $ git pull



Modules

1. Getting setup
2. Time Travel
3. Type less. Do more.
4. Branching & Merging
5. Collaborating
6. Forking



Module 1: Getting Setup

software, accounts and credentials

Overview

1

1. Installing git
2. Installing gcc
3. Installing an editor
4. Configuring git
5. Configuring vim
6. Configuring Line Endings
7. Create a GitHub account
8. Forge your new identity

Installing git

1.1

- Who does not have git installed? Raise your hand in shame!
  - Linux
    - $ apt-get install git -y
    - $ yum install git
  - Mac OSX (use homebrew)
    - $ brew install git
  - Windows (use Cygwin)
    - http://learnstemlabs.com/articles/

Installing gcc
how else would you compile your code?

1.2

- Who does not have git installed? Raise your hand in shame!
  - Linux
    - $ apt-get install gcc -y
    - $ yum install gcc
  - Mac OSX (already installed)
    - $ gcc
  - Windows (use Cygwin)

Installing Sublime Text 3

or some other less new-age editor … vim anyone?

1.3

- http://www.sublimetext.com/3
- You need package control
  - https://packagecontrol.io/installation
- Highly Recommended Packages
  - SideBarEnchancements
  - MarkdownEditing
- Installing a package
  - Ctrl + Shft + P or Cmd + Shift + P
  - Type: “PCIP” for Package Control Install Package
  - <package name>

git Configuration

1.4

- Tell git who you are
  - $ git config --global user.name "Your Name"
  - $ git config --global user.email your_email@whatever.com
- Tell git what editor you want to use for commit messages
  - $ git config --global core.editor vim
- Tell git that you would like to see pretty colors
  - $ `git config --global color.ui auto`

vim Configuration

1.5

.vimrc

http://git.io/vCIBR

```
"enable syntax highlighting
syntax on

"I don't like it when vim auto-wraps my code.
set nowrap

"only needed in cygwin
set nocompatible

" these setup tabs and indents
set ts=4
set smartindent
set tabstop=4
set shiftwidth=4

"use w!! when you forget sudo
cmap w!! w !sudo tee > /dev/null %

"turn on spellchecking
setlocal spell spelllang=en_us
```

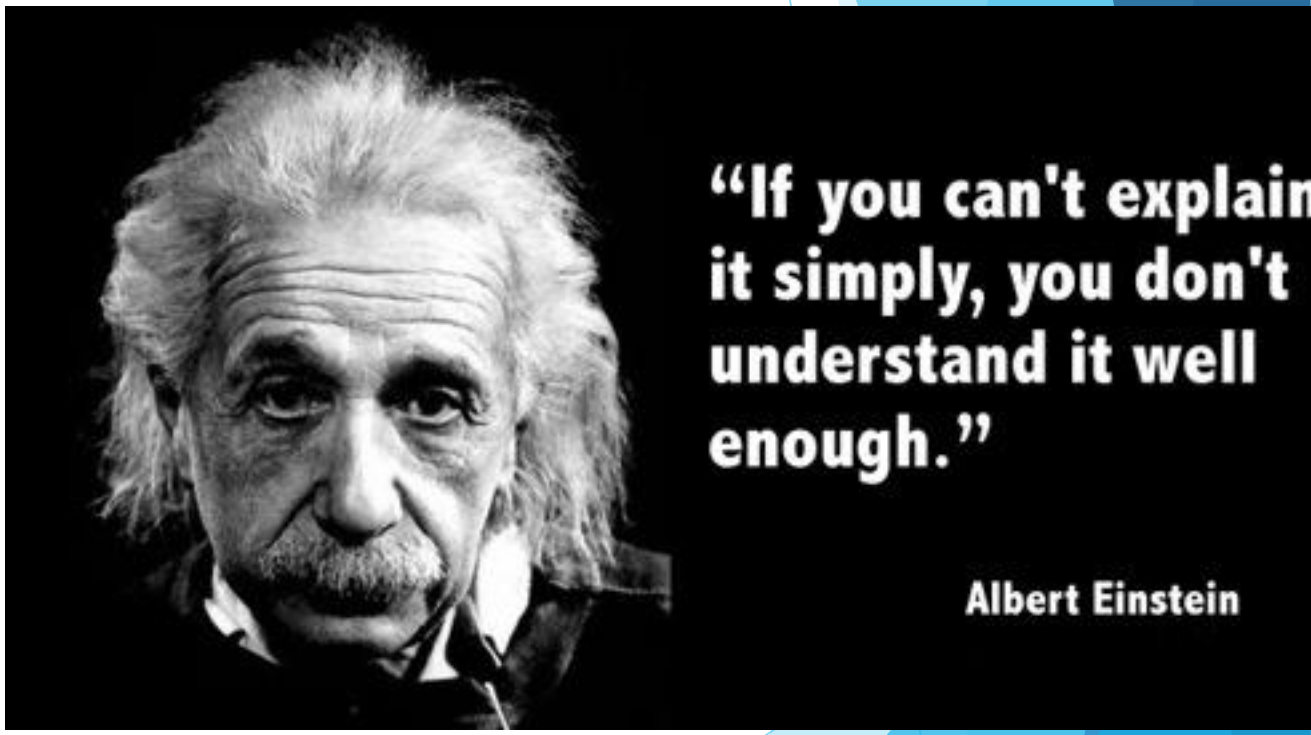
Line Ending Preferences

why can’t we all just agree and be friends

1.6

- Unix/Mac users:
  - `git config --global core.autocrlf input`
  - `git config --global core.safecrlf true`

- And for Windows users:
  - `git config --global core.autocrlf true`
  - `git config --global core.safecrlf true`



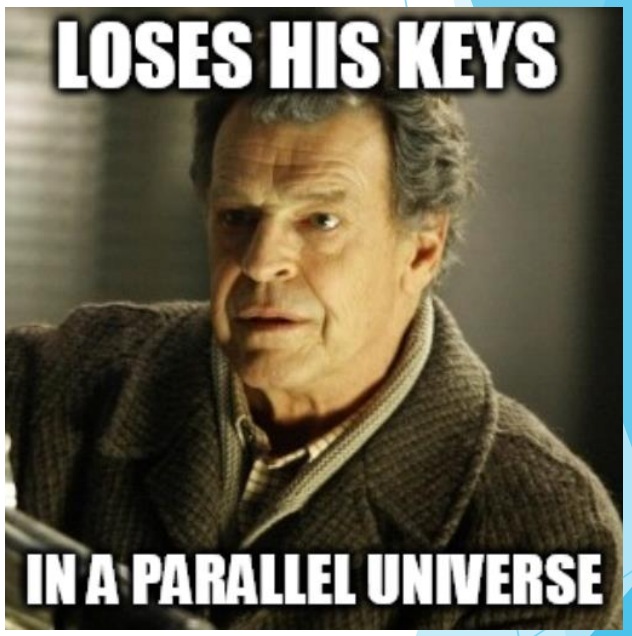
Create a GitHub account
or bitbucket or gitlab

1.7

- Go to github.com
- Create account
- Profit!

Forge your new identity
ssh key identity that is

1.8

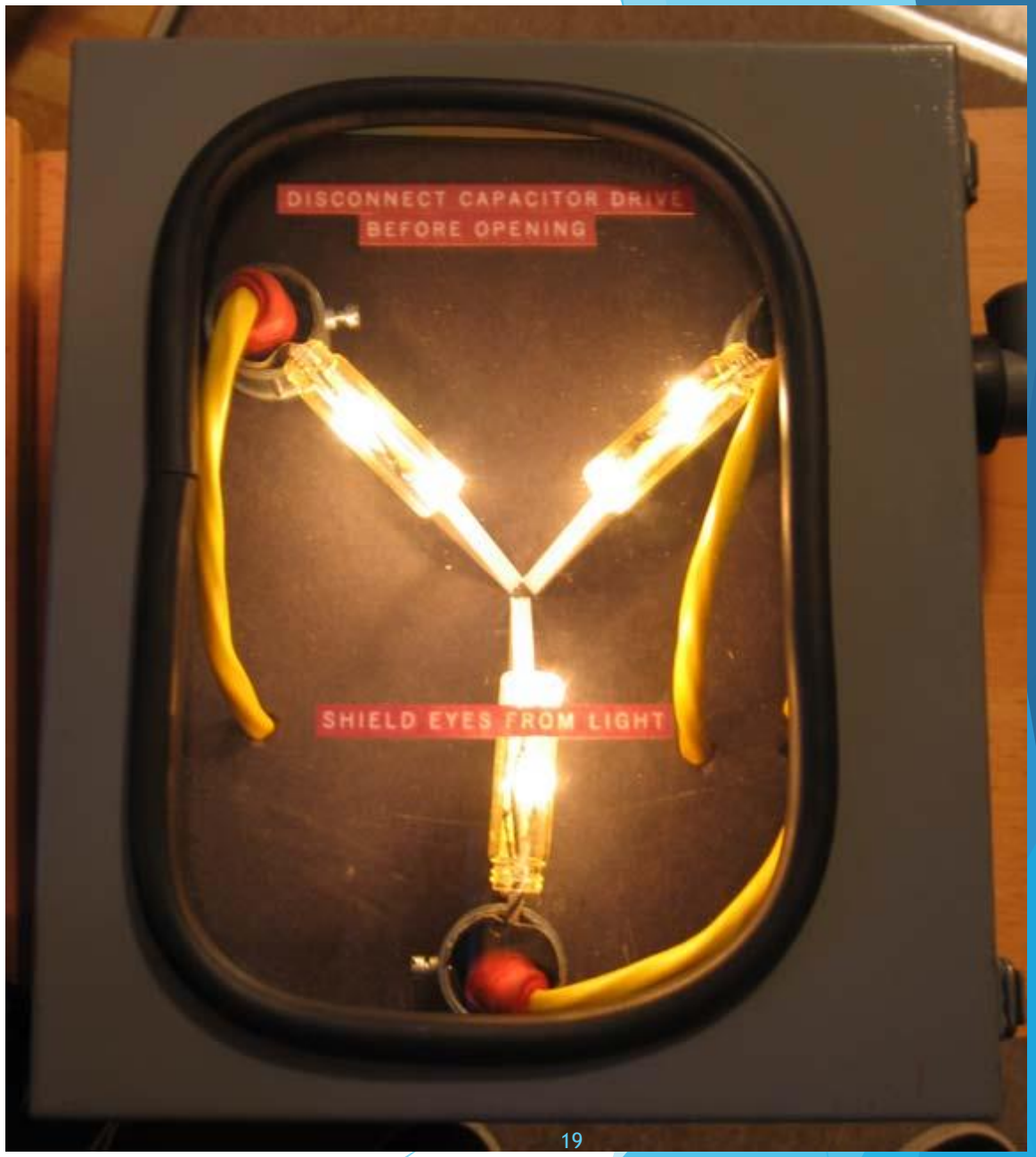


- https://help.github.com/articles/generating-ssh-keys/
  - Google: github ssh
- Make sure to add your public key to your account

- Understanding public key cryptography
  - Public / Private key pair
  - The public key decrypts that encrypted by the private key
  - The private key decrypts that encrypted by the public key
  - Everyone can encrypt with the public key, only the private key holder can decrypt
  - That encrypted with the private key can be decrypted with the public key, since the private key is private, the decryptor can verify the payload is from a known encryptor

Module 2: Time Travel

backup, restore and view code versions



Creating History
Overview

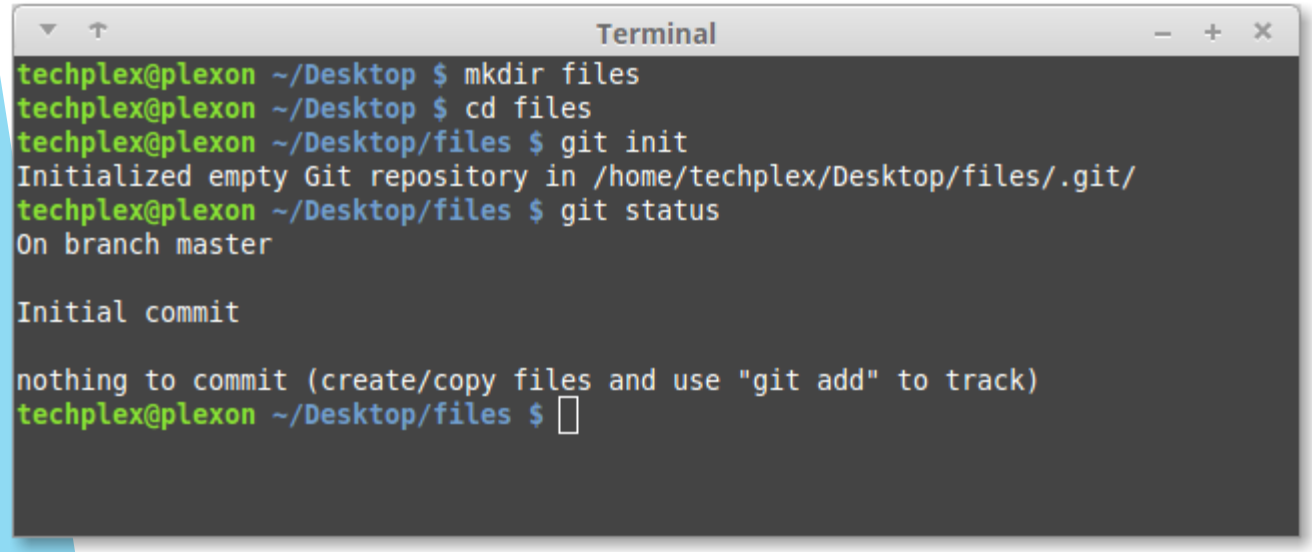
2

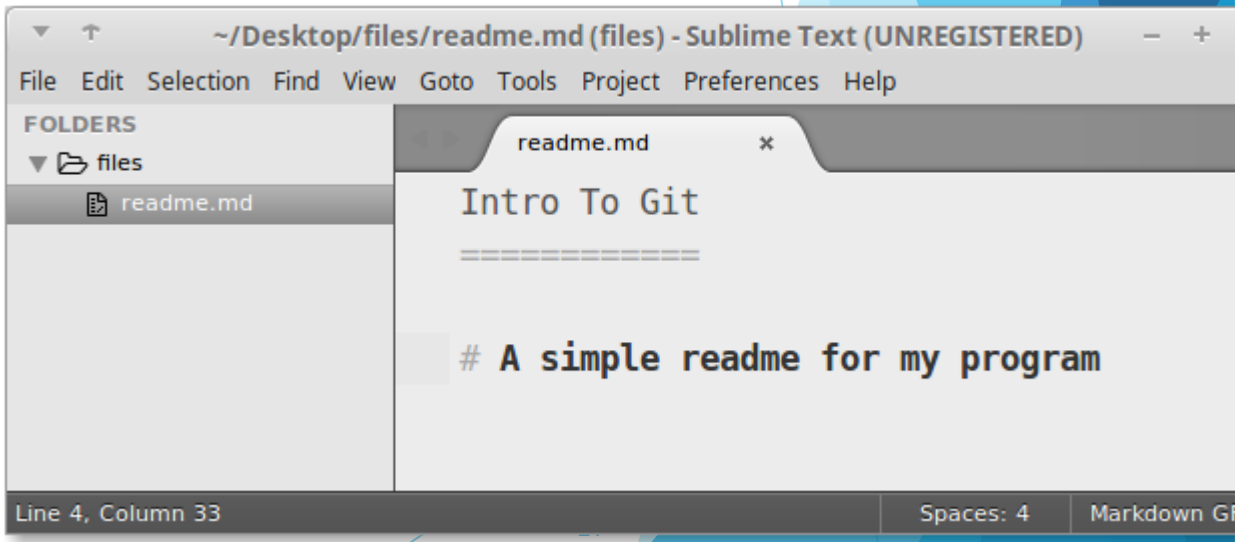
- Creating a working directory ..... ($ mkdir files)
- Initializing the repository ..... ($ cd files $ git init)
- Checking the status! ..... ($ git status)
- Create a readme
- Adding to the staging area ..... ($ git add .)
- Committing staged changes ..... ($ git commit)
- Starting to Program
- Ignoring (compiled) files ..... ($ vim .gitignore)
- Checking the status! ..... ($ git status)
- Making more changes
- Stage & Commit ..... ($ git add . $ git commit)
- Partial stage & globing ..... ($ git add -p .)
- Reverting History

Creating History
Starting

2.1

- Creating a working directory ..... ($ mkdir files)
- Initializing the repository ..... ($ cd files $ git init)
- Checking the status! ..... ($ git status)
- Create a readme





Creating History
Staging & Committing

2.2

- Checking the status! ..... ($ git status)
- Adding to the staging area ..... ($ git add .)
- Committing staged changes .....($ git commit)

Terminal

```
techplex@plexon ~/Desktop/files $ git status
On branch master

Initial commit

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

        readme.md

nothing added to commit but untracked files present (use "git add" to track)
techplex@plexon ~/Desktop/files $
```

Terminal

```
techplex@plexon ~/Desktop/files $ git add .
techplex@plexon ~/Desktop/files $ git status
On branch master

Initial commit

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

        new file:   readme.md

techplex@plexon ~/Desktop/files $ git commit
```

Terminal

```
Adding a readme
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   readme.md
#
~
~
~
:wq
```

Terminal

```
techplex@plexon ~/Desktop/files $ git commit
[master (root-commit) 68cd527] Adding a readme
 1 file changed, 4 insertions(+)
 create mode 100644 readme.md
techplex@plexon ~/Desktop/files $
techplex@plexon ~/Desktop/files $
```

Creating History
Programming

2.3

- Starting the Program
- Compile the program

~/Desktop/files/main.c (files) - Sublime Text (UNREGISTERED) (on

File Edit Selection Find View Goto Tools Project Preferences Help Arc

FOLDERS
files
main.c
readme.md

main.c

```c
// Sample Hello World Program
// B.Bourque 10/12/15

#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

master*, Line 3, Column 1 | Tab Size: 4 | C++

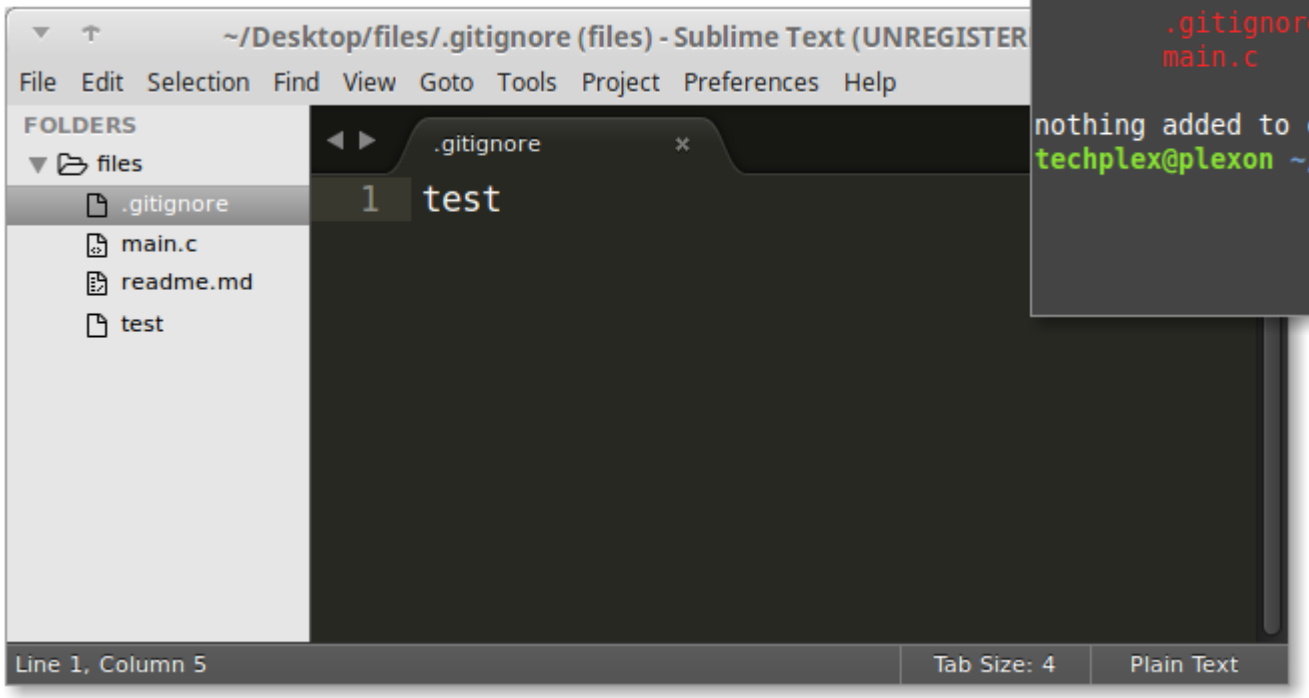
Terminal

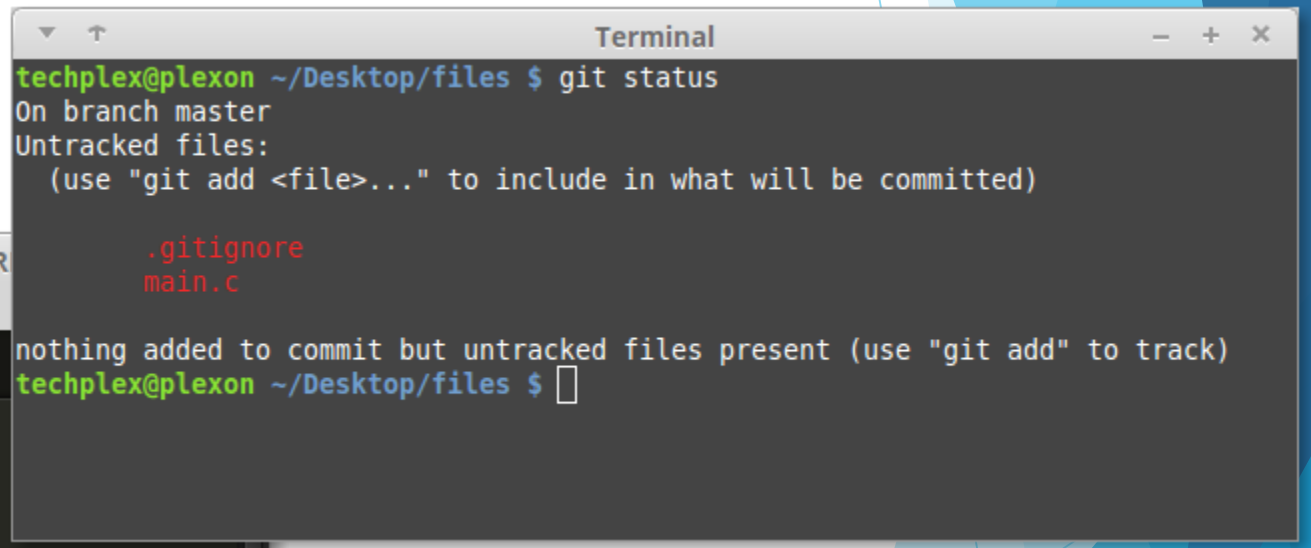
```
techplex@plexon ~/Desktop/files $ gcc main.c -o test
techplex@plexon ~/Desktop/files $ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        main.c
        test

nothing added to commit but untracked files present (use "git add" to track)
techplex@plexon ~/Desktop/files $
```

Creating History
Ignoring

2.4

- Ignoring (compiled) files ... ($ vim .gitignore)
- Checking the status! ..... ($ git status)





Creating History
Committing

2.5

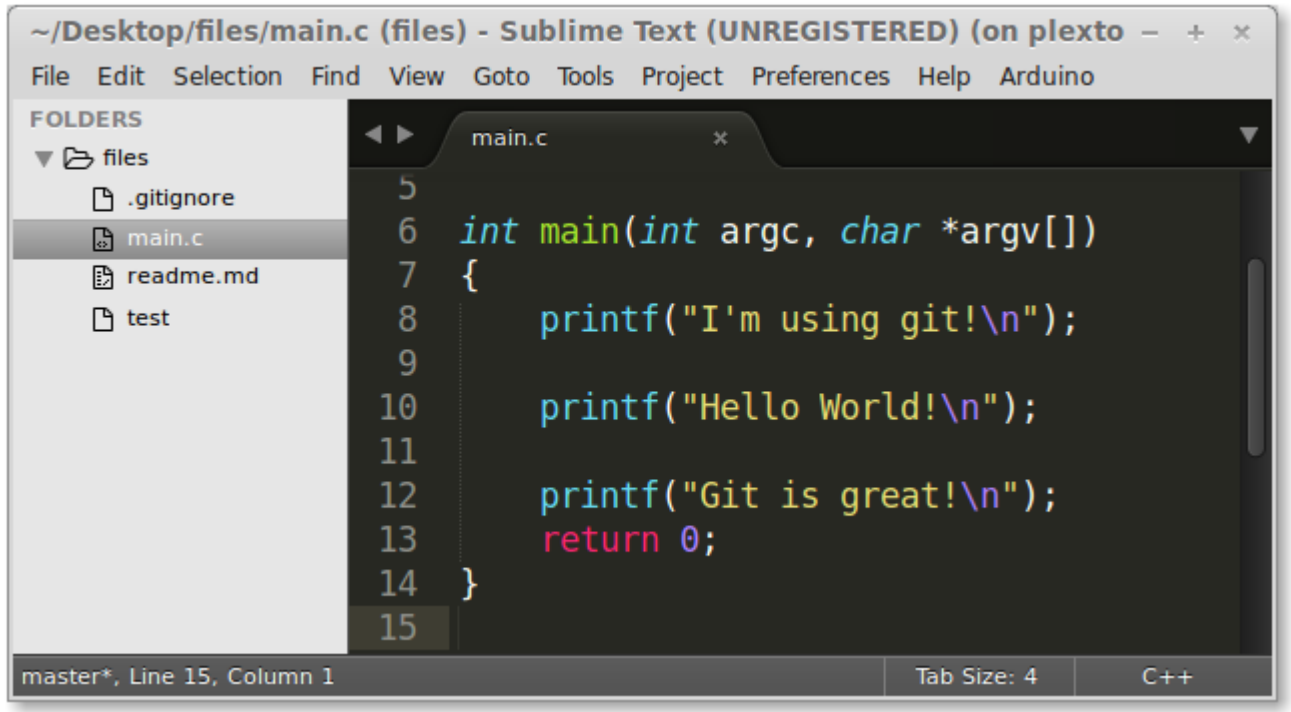
- Stage & Commit ..... ($ git add . $ git commit)

```
Terminal
techplex@plexon ~/Desktop/files $ git add .
techplex@plexon ~/Desktop/files $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore
        new file:   main.c

techplex@plexon ~/Desktop/files $ git commit -m "adding main program"
[master bb858ef] adding main program
 2 files changed, 11 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 main.c
techplex@plexon ~/Desktop/files $
```

Creating History
More Committing

2.6

- Adding more code



Creating History
More Committing

2.7

- Partial stage & globing ..... ($ git add -p .)

```
bash
techplex@plexon ~/Desktop/files $ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   main.c
#
no changes added to commit (use "git add" and/or "git commit -a")
techplex@plexon ~/Desktop/files $
```

```
techplex@plexon: ~/Desktop/files (on plextop)
 int main(int argc, char *argv[])
 {
+       printf("I'm using git!\n");
+
        printf("Hello World!\n");
+
+       printf("Git is great!\n");
        return 0;
-}
\ No newline at end of file
+}
Stage this hunk [y,n,q,a,d,/,s,e,?]?
```

```
bash
Stage this hunk [y,n,q,a,d,/,s,e,?]? ?
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk nor any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk nor any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
@@ -5,6 +5,10 @@
```

Creating History
More Committing

2.8

- Partial stage & globing ..... ($ git add -p .)

```
techplex@plexon: ~/Desktop/files (on plextop)
 int main(int argc, char *argv[])
 {
+        printf("I'm using git!\n");
+
         printf("Hello World!\n");
+
+        printf("Git is great!\n");
         return 0;
-}
\ No newline at end of file
+}
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
```

```
bash
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 3 hunks.
@@ -5,4 +5,6 @@

 int main(int argc, char *argv[])
 {
+        printf("I'm using git!\n");
+
         printf("Hello World!\n");
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y
@@ -8,2 +10,4 @@
         printf("Hello World!\n");
+
+        printf("Git is great!\n");
         return 0;
Stage this hunk [y,n,q,a,d,/,K,j,J,g,e,?]? n
```

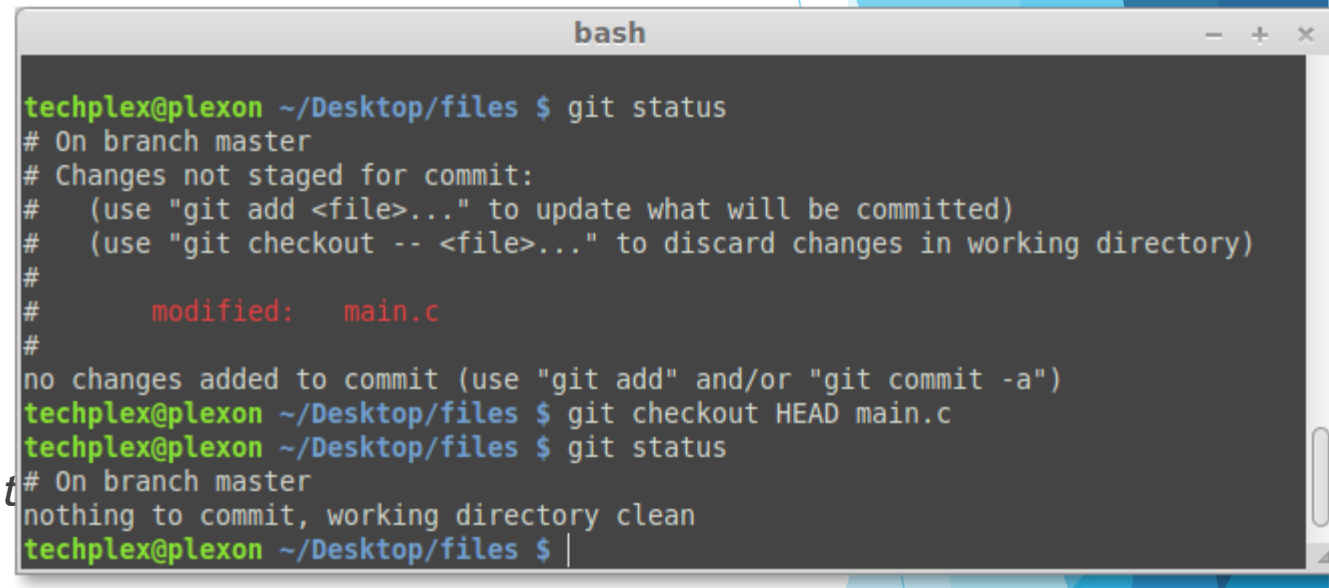
- Commit ..... ($ git commit)

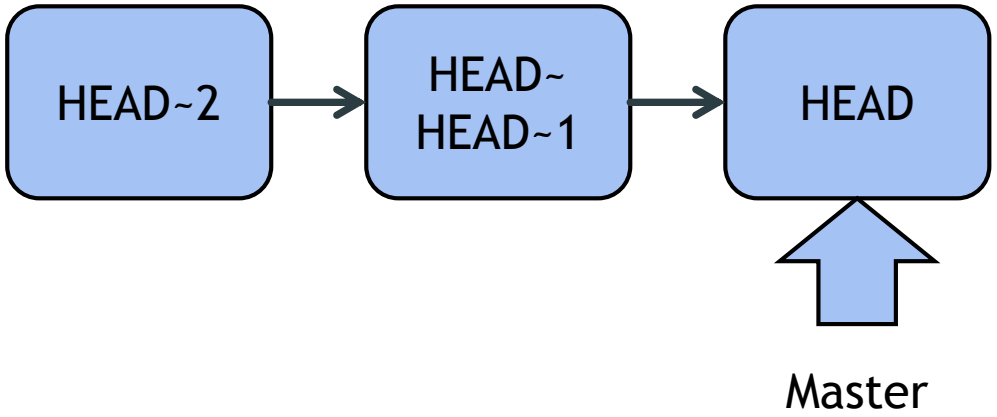
Reverting Local Changes

2.9

- Undoing local changes
  - $ git checkout HEAD *file(s)*
  - $ git checkout HEAD^ *file(s)*

- *HEAD = The current tip of the master*
- *HEAD~ = First parent of the tip of master*
- *HEAD~2 = First parent of the First Parent of the t*



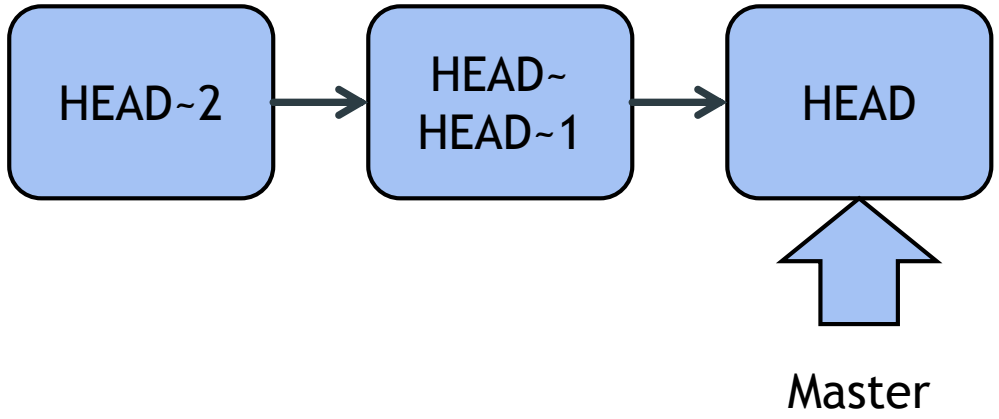


Viewing History

2.10

- git log

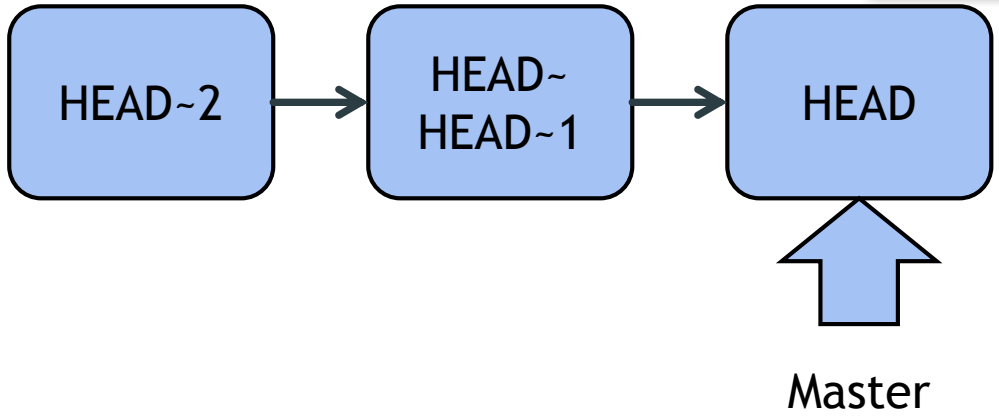
```
bash
commit 024d1769e08cee47959a19abb6c4283555172ee7
Author: Blake Bourque <Techplex.Engineer@gmail.com>
Date:   Fri Oct 16 10:41:33 2015 -0400

    Partial adding main

commit e0959efe652395e04ab9d2e7840ea021a0b22866
Author: Blake Bourque <Techplex.Engineer@gmail.com>
Date:   Fri Oct 16 10:22:06 2015 -0400

    Adding code and gitignore

commit d2abf4d68d44769ae56fc53bb94f218346e25949
Author: Blake Bourque <Techplex.Engineer@gmail.com>
Date:   Fri Oct 16 10:21:07 2015 -0400
:
```



Time Traveling – Viewing Old Code Detaching the HEAD 2.11

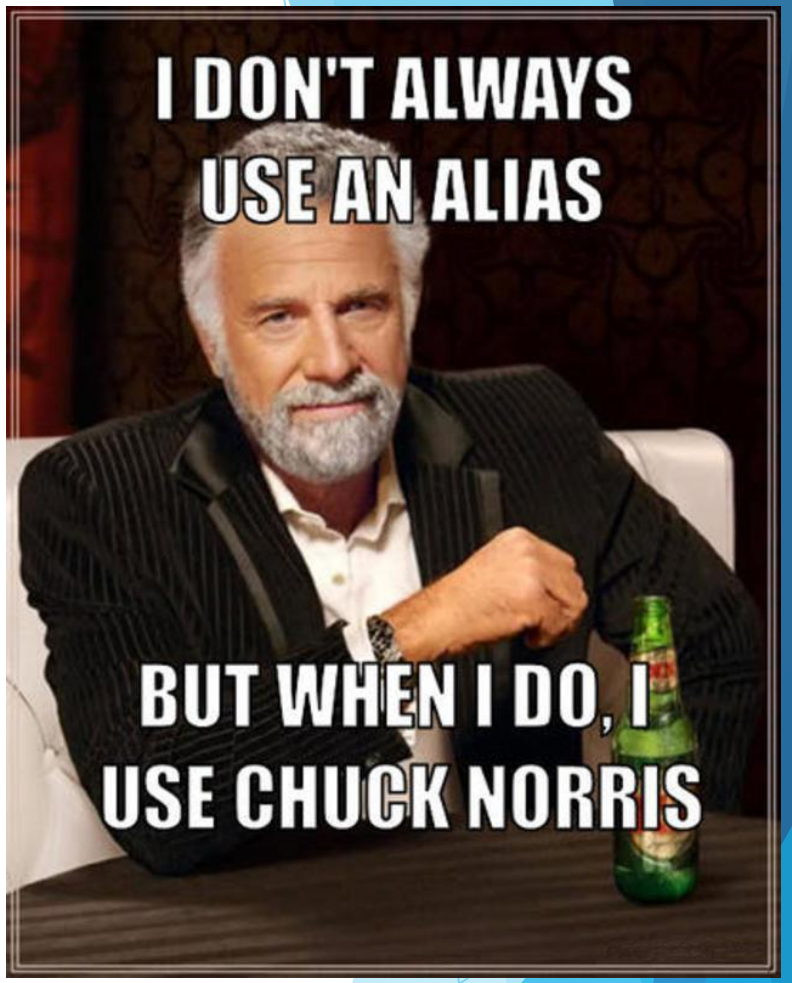
$ git checkout *hash file(s)*

```
bash
techplex@plexon ~/Desktop/files $ git log
techplex@plexon ~/Desktop/files $ git co e0959efe
Note: checking out 'e0959efe'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at e0959ef... Adding code and gitignore
techplex@plexon ~/Desktop/files $
```



Module 3:
Type less. Do more.

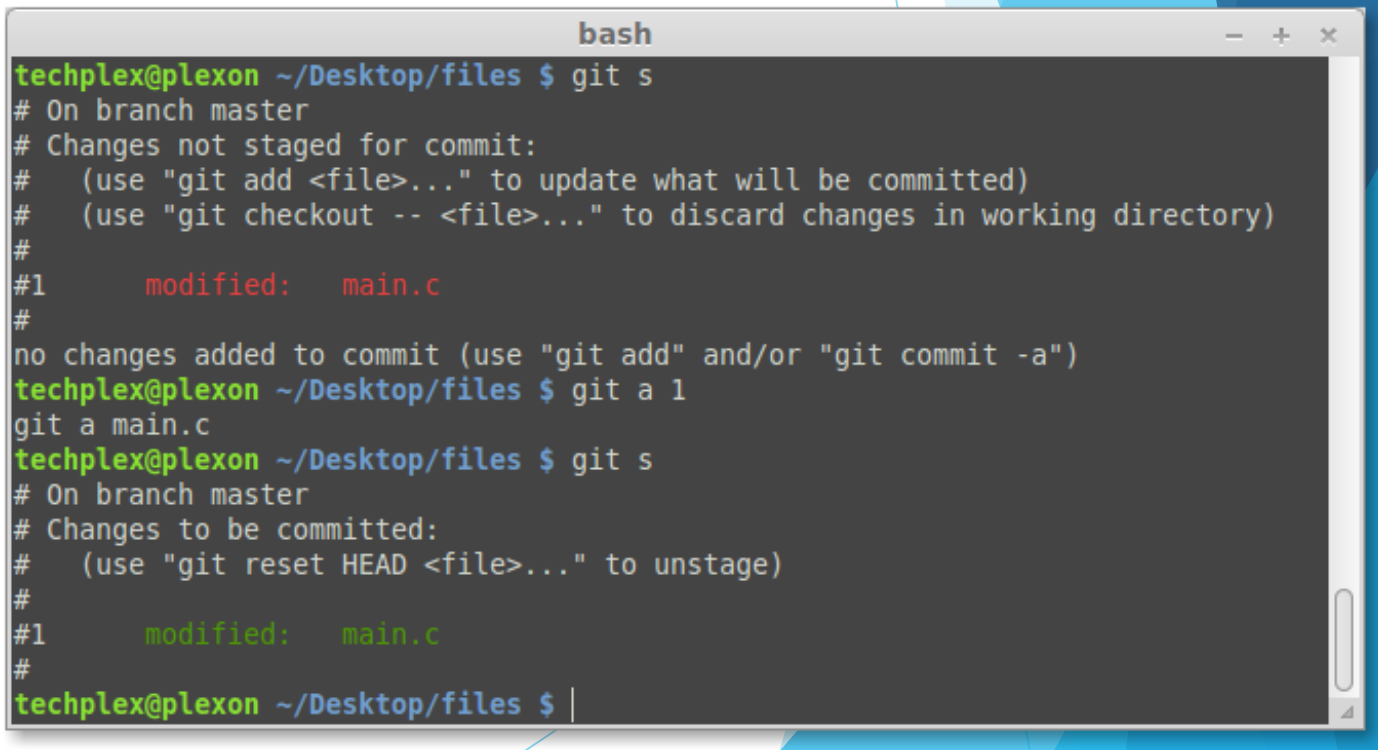
git-number and aliases



Type Less. Do More.
The magic of git-number

3.1

- Download git-number from github http://git.io/vCIWL
- Copy the three programs onto your path
  - git-id
  - git-list
  - git-number
- Alias git to git-number (add to .bashrc)
  - alias git='git number -c git‘
- Logout & Back in



Type Less. Do More.

Making aliases for fun and profit

3.2

.gitconfig

```
[alias]
        a = add -A              # also stage deletions
        am = commit --amend     # sometimes the history books got it wrong
        b = branch
        c = commit
        cl = clone
        co = checkout
        d = diff
        nb = checkout -b
        s = "!git-number"       # <3 git-number
        unstage = reset         # reset is scarry

[core]
        editor = vim
[color]
        ui = true
[push]
        default = simple
```

Type Less. Do More.

Using aliases for fun and profit

3.3

```
Terminal
techplex@plexon ~/Desktop/files $ git
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

1       helper.c
2       helper.h

nothing added to commit but untracked files present (use "git add" to track)
techplex@plexon ~/Desktop/files $ git add .
techplex@plexon ~/Desktop/files $ git s
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

1       new file:   helper.c
2       new file:   helper.h

techplex@plexon ~/Desktop/files $ git c
```

Module 4: Branching & Merging

Experimenting & Recovering

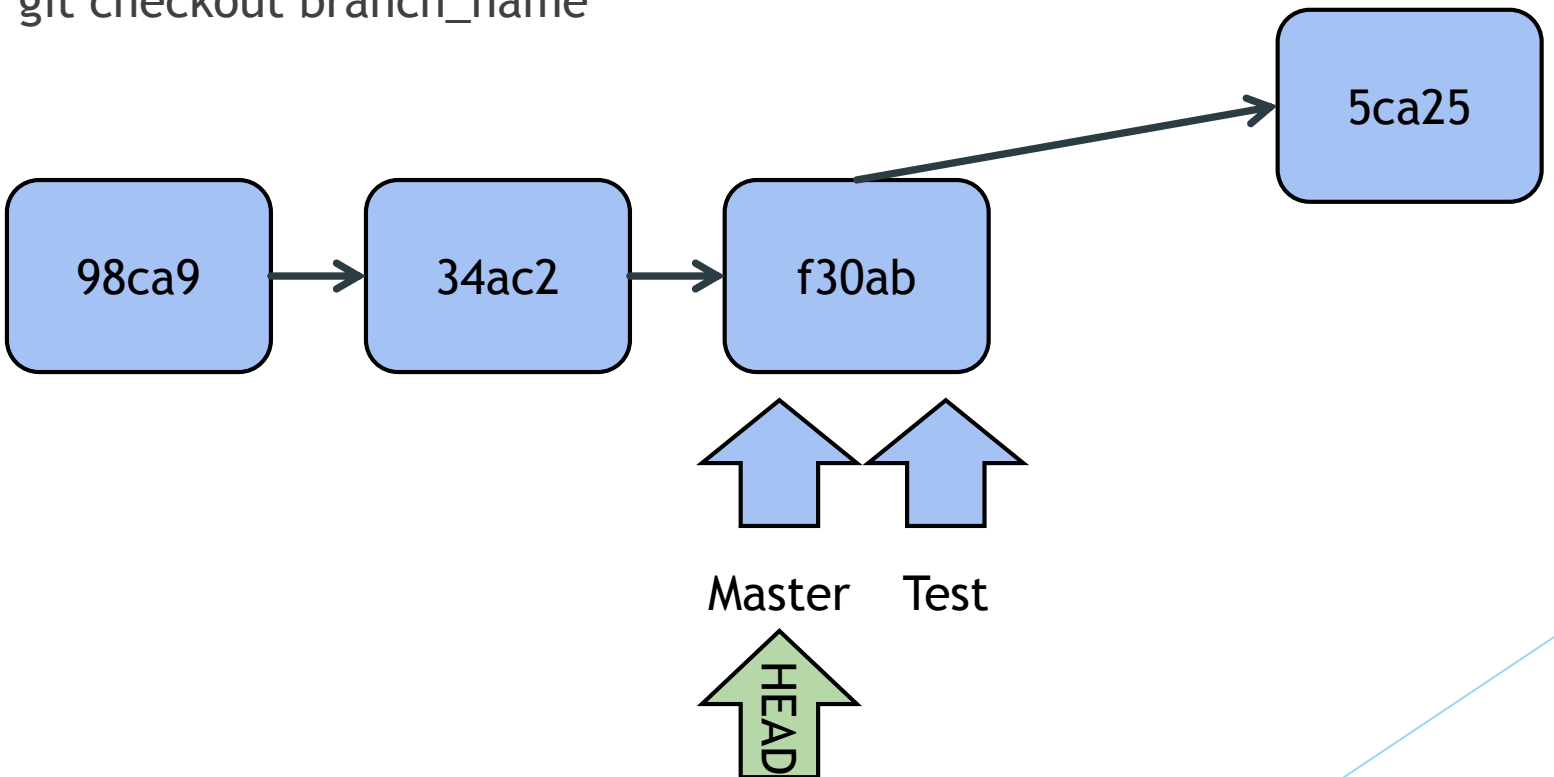
DON'T DO THIS!

Creating branches

4.1

- $ git nb branch_name
- $ git checkout -b branch_name
- $ git branch branch_name
- $ git checkout branch_name

Todo:
1. Create test branch
2. Make a code Change
3. Commit change



Navigating branches

4.2

- $ git checkout master
- $ git status
- What happens to the working directory when we change branches?

Merging a branch
“Clean” merging

4.2

- $ git co test
- ▶ Add a line to the file
- $ git add .
- $ git commit
- $ git co master
- $ git merge test

Merging a branch
Dealing with conflicts

4.3

- Create a conflict
  - $ git co test
  - $ # change line 8 to “test”
  - $ git add .; git commit
  - $ git co master
  - $ # change line 8 to “master”
  - $ git add .; git commit
  - $ git merge test

Merging a branch

Dealing with conflicts

4.4

bash

```
techplex@plexon ~/Desktop/files $ git merge test
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.
techplex@plexon ~/Desktop/files $
```

bash

```
techplex@plexon ~/Desktop/files $ git
# On branch master
# You have unmerged paths.
#   (fix conflicts and run "git commit")
#
# Unmerged paths:
#   (use "git add <file>..." to mark resolution)
#
#1       both modified:      main.c
#
no changes added to commit (use "git add" and/or "git commit -a")
techplex@plexon ~/Desktop/files $
```

~/Desktop/files/main.c (files) - Sublime Text (UNREGISTERED) (on plexto

File Edit Selection Find View Goto Tools Project Preferences Help Arduino

FOLDERS

- files
  - .gitignore
  - main.c
  - readme.md
  - test

main.c

```
5
6  int main(int argc, char *argv[])
7  {
8  <<<<<<< HEAD
9      printf("I'm trying git!\n");
10 =======
11     printf("I'm using\n");
12 >>>>>>> test
13
14     printf("Hello World!\n");
```

bash

```
1 Merge branch 'test'
2
3 Conflicts:
4     main.c
5 #
6 # It looks like you may be committing a merge.
7 # If this is not correct, please remove the file
8 #   .git/MERGE_HEAD
9 # and try again.
10
11
:x
```

Module 5: Collaborating

pushing and pulling

Collaborating
Using Github

5.1

- Team up with a partner
- create a github repo to share
- each of you create a file, commit and push
- edit each others file, push resolve any conflicts

Module 6:
Tricks & Treats

Submitting Logs for Senior Project

6

- Use tags each time you output your logs
  - git tag tag1
- Here is the command:
  - git pr tag1…tag2
- Use enscript here is how to set header
- http://git.io/vCV4U

Resources

- http://gitimmersion.com/