git

for beginners

// with some basic commands

#WhoAmI?

@vinhnx

http://vinhnx.github.com

http://github.com/vinhnx

a brief introduction

"In software development, **Git** *(/ ɡ ɪt/) is a*
distributed revision control
and ***source code management*** *(SCM) system*
with an emphasis on speed…
Git was initially designed and developed
by **Linus Torvalds** *for* **Linux kernel** *development…"*

– Git (Wikipedia) –

tl;dr

Installing git in your development machine

MAC:

- http://code.google.com/p/git-osx-installer
- http://mac.github.com/

Windows:

- http://code.google.com/p/msysgit
- http://windows.github.com/ <- highly recommended! -If you're using Windows PC like me ;)

Eclipse plugins:

- http://eclipse.github.com/

Mobile:

- http://mobile.github.com/

Let’s get started!

Setup name & email

```
→ git config --global user.name "Your Name“
→ git config --global user.email "your_email@whatever.com"
```

Setup line ending preferences

* For Unix/Mac users:

```
→ git config --global core.autocrlf input
→ git config --global core.safecrlf true
```

* For Windows users:

```
→ git config --global core.autocrlf true
→ git config --global core.safecrlf true
```

your first commit

```
→ mkdir lerepo
→ cd lerepo
→ git init
Initialized emtpy Git repository /somedir/lerepo/.git/
→ echo “oh hai” > hai.txt
→ git add hai.txt
→ git commit -m “first commit”
[master (root-commit) 8970fa6] first commit
1 file changed, 1 insertion(+)
create mode 100644 hai.txt
```

Me: Think of repo (repository) as a folder.
Them: Repo (repository) is commonly refers to a location for storage, often for safety or preservation.
-> read this for more detail

git add means telling git that "I want you to index this/these file(s)!"

Say, you already had another file in your repository, named, **script.js**.

```
→ ls -l
total 2
-rw-r--r-- 1 admin          None  7 Oct  1 10:23 hai.txt
-rwxr-xr-x 1 Administrators None 47 Oct  1 10:57 script.js
```

... and you want to add both files. (**hai.txt** & **script.js**)

```
→ git add hai.txt script.js
→ git commit -m "index all the files"
[master (root-commit) 8d018aa] index all the files
 2 files changed, 4 insertions(+)
 create mode 100644 hai.txt
 create mode 100644 script.js
```

... but in come cases, you just want to add *only* specific file(s) manually, just tell git so:

```
→ git add hai.txt
→ git commit -m “updated hai.txt”
[master (root-commit) 8970fa6] first commit
1 file changed, 1 insertion(+)
create mode 100644 hai.txt
```

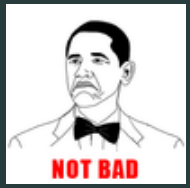
cool,
now what?!

checking **status** --before indexing (staging)

```
→ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       hai.txt
nothing added to commit but untracked files present (use "git add" to
track)
```

checking **status** --after indexing (staging)

```
→ git status -sb
# On branch master
nothing to commit (working directory clean)
```



quick tip

viewing **status** the short/clean way

```
→ git status -sb
## Initial commit on master
?? hai.txt
```

English, Do You Speak It?

- You are now on branch **master**
- Your file (hai.txt) has been modified
- … but you haven't staged it yet
- … so, your index is unclean === dirty ☹

history
lesson

```
→ git log --pretty=oneline
56230b1 updated hai.txt (Vinh Nguyen, 33 seconds ago)
8d018aa index all the files (Vinh Nguyen, 13 minutes ago)
```

quick tip

(again)

The Ultimate Log Format

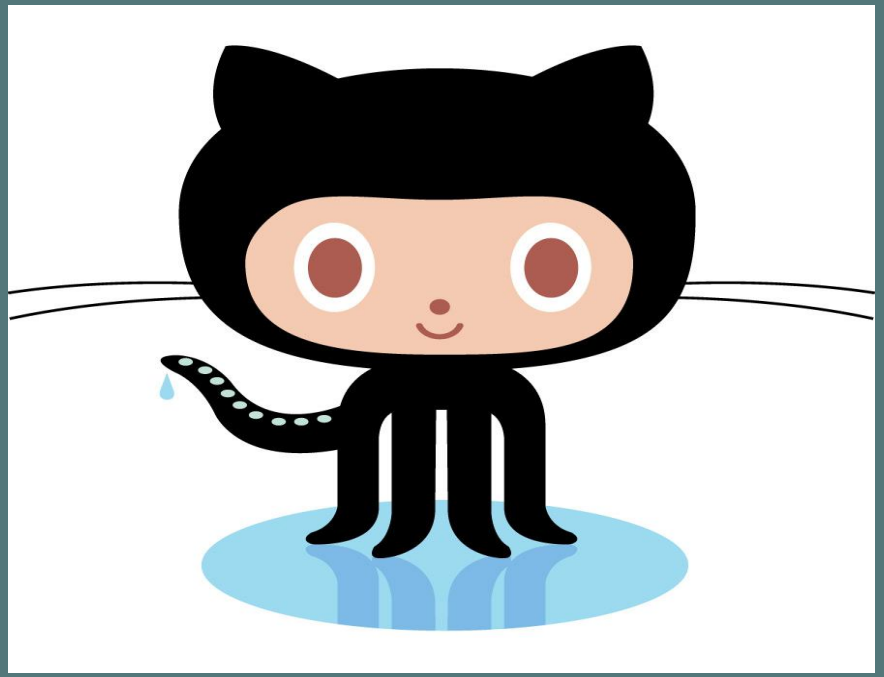
```
→ git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short
```

Take a look in detail:

--pretty="..." defines the format of the output.
%h is the abbreviated hash of the commit
%d are any decorations on that commit (e.g. branch heads or tags)
%ad is the author date
%s is the comment
%an is the author name
--graph informs git to display the commit tree in an ASCII graph layout
--date=short keeps the date format nice and short

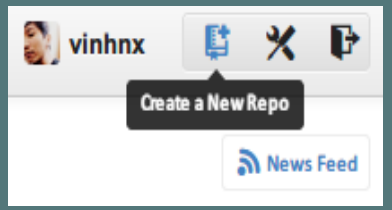
Ref: http://gitimmersion.com/lab_10.html

Your first Github repo



your first github repo

1. Register an account at github.com.
2. Create new repo.
3. Fill in information.
4. Copy your SSH.. Eg: `git@github.com:yourname/your-repo-name.git`
5. In your terminal/command prompt/whatever. Type in, with **hub** is your remote name (optional)




```
→ git remote add hub git@github.com:yourname/your-repo-name.git
```

6. Let me check

```
→ git remote -v
hub git@github.com:yourname/your-repo-name.git (fetch)
hub git@github.com:yourname/your-repo-name.git (push)
```

not
so fast...
(another tip)

It's recommended to run **git status** again to check whether if files are staged properly and most importantly, your current working branch
** [in default, we are all working ~~for~~ on **master** branch].
** a branch is like your feature lab – wherein you can experiment something cool. If "it just works", merge it to master branch for your final code, else you may forget or abandon it ...

```
→ git status
# On branch master
nothing to commit (working directory clean)
```

Kay, things seem rather good up to now! ☺
Press arrow right to next slides...

cat

:3

cat-file

(not really)

In order to check and see more information for what we have done in recent activities. You can run this:

```
→ git cat-file -p HEAD
tree 56230b16f2e04c1a385008f78ad761152fdf0480
parent 8d018aa81664cf7caad20a3ab7bcd6d03354a18b
author Vinh Nguyen <name@mail.com> 1349065326 +0100
committer Vinh Nguyen <name@mail.com> 1349065326 +0100

updated hai.txt
```

- Right now, don't bother the **cat-file -p** command.
- What does **HEAD** mean, it's simply your current branch/ last committed stage. (Don't confuse with **Index** (staging area) and **working tree** (the stage of files in checkout)). -- See this for more details:
- Now you have information about your **most recently commit** (HEAD === master), You can inspect last 2, by HEAD^n with n is an specific number, eg: HEAD^1.
- Remember the **5W**, the **what/when/where/who/why**
- Still remember history?! Can you spot what in common ? ☺

```
→ git log --pretty=oneline
56230b1 updated hai.txt (Vinh Nguyen, 33 seconds ago)
8d018aa index all the files (Vinh Nguyen, 13 minutes ago)
```

Noticed that in your **log**, we have 2 commits up to now, represented by the first **7** characters of **SHA** (Secure Hash Algorithm).

56230b1 8d018aa

And surprisingly, but not quite, in your **git cat-file –p HEAD** command, we now see them again. What the fuzz?

```
tree 56230b16f2e04c1a385008f78ad761152fdf0480
parent 8d018aa81664cf7caad20a3ab7bcd6d03354a18b
author Vinh Nguyen <name@mail.com> 1349065326 +0100
committer Vinh Nguyen <name@mail.com> 1349065326 +0100

updated hai.txt
```

Git is just connection of dots. Think of it as a graphical base, a dot/node represented by a branch in a tree, dot will be our working **HEAD**. Tree with SHA **56230b1** is a child of its parent as SHA **8d018aa**.

Read again, we may notice **git cat-file –p HEAD,** we now have more information like: author of last commit, name of committer, time when commit occurred, and the message of last commit…

push

Kay, back to **push** topic.
Now your Github repo is ready for continous development.
Why don't we push it?

```
→ git push -f hub master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (7/7), 594 bytes, done.
Total 7 (delta 0), reused 0 (delta 0)
To git@github.com/yourname/your-repo-name.git
+ c80efd2…40807b2 master -> master(force update)
```

PUBLIC vinhnx / lerepo

Pull Request | Unwatch | Star 0 | Fork 0

Code | Network | Pull Requests 0 | Issues 0 | Wiki | Graphs | Admin

reposity for my slide.

Clone in Windows | ZIP | HTTP | SSH | Git Read-Only | git@github.com:vinhnx/lerepo.git | Read+Write access

branch: master | Files | Commits | Branches 1 | Tags | Downloads

Latest commit to the **master** branch

updated hai.txt

vinhnx authored 2 hours ago commit 40807b25d1

lerepo /

name	age	message	history
hai.txt	2 hours ago	updated hai.txt [vinhnx]	
script.js	2 hours ago	index all the files [vinhnx]	

summary & more

```
# BASIC COMMAND …
## note: command with slash '/' in between can be optionally chosen, eg: -s/-b, you can choose either -s or -b
→ git init # to initialize a git repo
… hardcore hacking …
→ git status -s/-b/-sb # show file added to staging area, files with changes, untracked files
→ git log hai.txt # show recent commits on hai.txt
→ git add ./-A/[file/files] # adding file(s) to index
→ git commit -m "commit message" # message of a commit
### working remotely
→ git remote add/delete [remote.name] [git.url] # adding/deleting remote
→ git push [remote.name] [branch.name] # update the [remote] with your commit from [branch.name]
→ git pull # fetch changes from remote and merge into current branch
→ git fetch [remote.name] # update the remote-tracking branch for [remote.name] (default is origin)
```

```
# EVEN MORE COMMAND …
## note: HEAD === most recent commit on your working branch. As I said before, default is master.
→ git add [dir] # add all files in [dir] directory
        → git add .
### add all files under current directory, untracked file included
→ git rm [file1] [files2] … [fileN] # remove n files from the project
→ git reset HEAD [file] # remove specified file from next commit
### branching
→ git checkout -b [branch.name] # create a new branch and switch to it
→ git branch -d [branch.name] # delete a branch
→ git rev-parse HEAD # show me SHA of last commit
→ git cat-file -t HEAD # what type of last commit in current working branch
→ git cat-file -p HEAD # all your last commit's information belong to us ;)
→ git clone # clone a repo
```

tweaking git

(make it more awesome)

```
# Add colors to your ~/.gitconfig file:
[color]
  ui = auto

[color "branch"]
  current = yellow reverse
  local = yellow
  remote = green

[color "diff"]
  meta = yellow bold
  frag = magenta bold
  old = red bold
  new = green bold

[color "status"]
  added = yellow
  changed = green
  untracked = cyan
```

```
# Highlight whitespace in diffs [color]
  ui = true

[color "diff"]
  whitespace = red reverse

[core]
  whitespace=fix,-indent-with-non-tab,trailing-space,cr-at-eol

# Show files ignored by git:
ign = ls-files -o -i --exclude-standard
```

```
# Add aliases to your ~/.gitconfig file:
[alias]
  st = status
  ci = commit
  br = branch
  co = checkout
  df = diff
  dc = diff --cached
  lg = log -p
  ls = ls-files
  lol = log --graph --decorate --pretty=oneline --abbrev-commit

  lola = log --graph --decorate --pretty=oneline --abbrev-commit --all
```

ref

Gitref: http://gitref.org
Github help: https://help.github.com/
Git-scm: http://git-scm.org/
Pro git book: http://git-scm.com/book

```
→ git help [command]
```

Advanced Git by Github's Matthew McCullough
http://marakana.com/s/advanced_git_graphs__hashes__compression_matthew_mccullough_github,1280/index.html

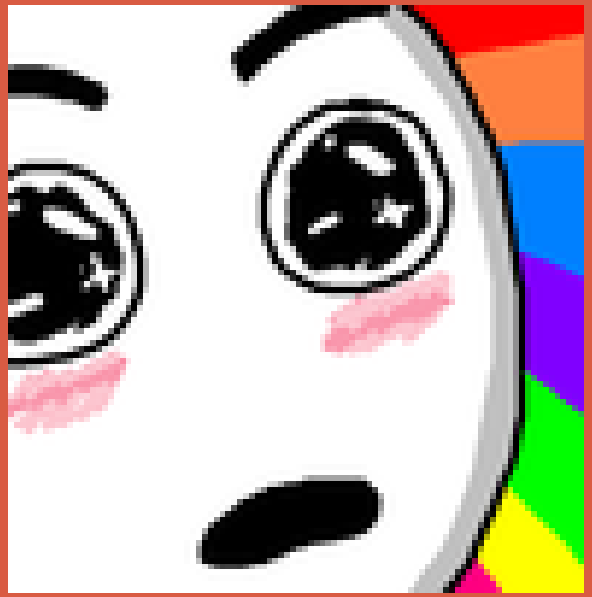
Introduction to Git with Scott Chacon of GitHub
http://marakana.com/s/video_introduction_to_git_with_scott_chacon_of_github,399/index.html

Git Magic - Tommy MacWilliam '13 from Harvard
http://cs50.tv/2011/fall/seminars/Git_magic/Git_magic.mp4

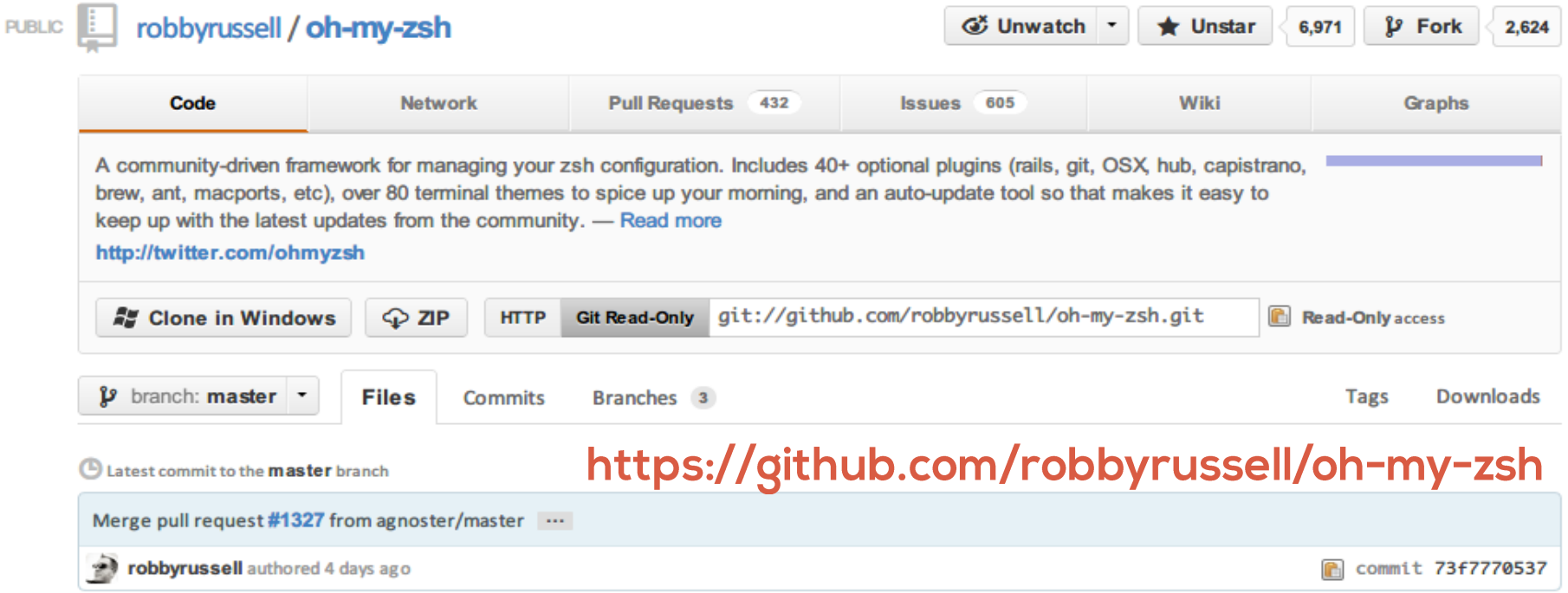
must watch!

typo? ...don't worry ;)

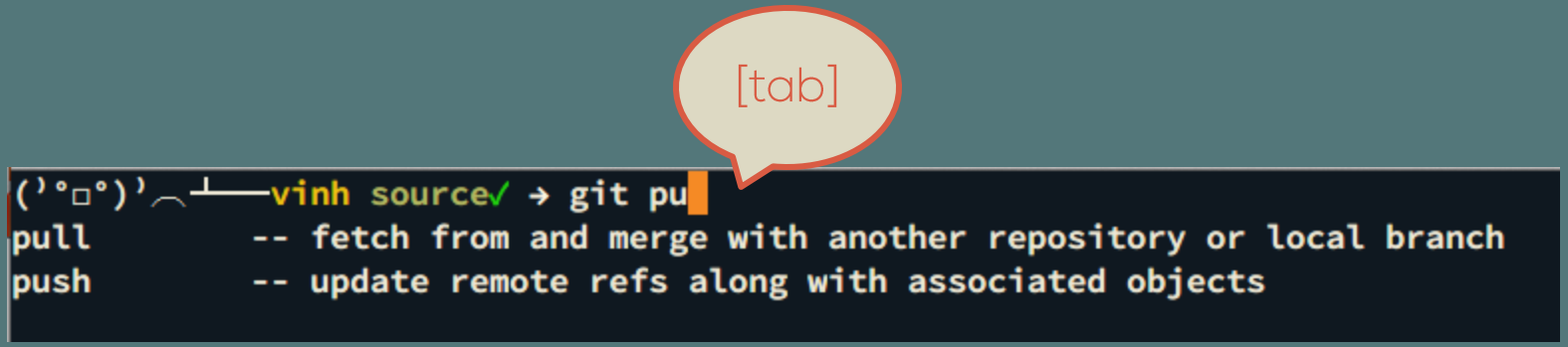
```
(ʾ°□°)ʾ︵┻━━test master… → git add -a
error: unknown switch `a'
usage: git add [options] [--] <filepattern>...

    -n, --dry-run         dry run
    -v, --verbose         be verbose

    -i, --interactive     interactive picking
    -p, --patch           select hunks interactively
    -e, --edit            edit current diff and apply
    -f, --force           allow adding otherwise ignored files
    -u, --update          update tracked files
    -N, --intent-to-add   record only the fact that the path will be added later
    -A, --all             add changes from all tracked and untracked files
    --refresh             don't add, only refresh the index
    --ignore-errors       just skip files which cannot be added because of errors
    --ignore-missing      check if - even missing - files are ignored in dry run
```



last but totally not least



"oh-my-zsh: your life in a shell" -- fox



awesome, autocomplete is awesome!!!

Well, that's it!
I hope these could help you
get acquainted with fundamental about
git.
;)

Thanks!

@vinhnx