GIT Class

kishore.kodical@ticketmaster.com

ticketmaster®

1. git commit

2. git push

3. leave building

GIT - Agenda

- Getting Started & Creating a new Project
- Making Changes, Staging & Committing
- Logs and history in Git
- Hashes & Tagging
- Undoing Changes
- Working with Branches
- Handling conflicts
- Cloning Repos
- Working with Remote Repos

GIT - Getting Started

Create a “Hello, World” program

- Starting in a working directory, create an empty directory named **hello**
- Next create a file named **hello.sh**

```
$ mkdir hello

$ cd hello

$ cat hello.sh
echo "Hello, World"
```

GIT - Getting Started

Create the local Repository & check Status

You have a directory with a single file.

To create a git repository from that directory, run the **git init** command.

```
$ git init
Initialized empty Git repository in /Users/kishore/git/hello/.git/


$ ls .git/
HEAD        branches    config      description hooks       info        objects
    refs
```

GIT - Getting Started

Now check the status of the repository:

```
$ 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:   hello.sh
```

GIT - Getting Started

Add the program to the Repository

Now add **hello.sh** to the repo

```
$ git add hello.sh

$ git commit -m "First Commit"
[master (root-commit) 2e263c7] First Commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.sh

$ git status
On branch master
nothing to commit, working directory clean
```

This means the repo has all changes made in the current directory.

Making Changes

Change the **hello.sh** file

Modify the file to accept an argument from command line

```
$ cat hello.sh
echo "Hello," $1
```

Now check the status

```
$ 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:   hello.sh

no changes added to commit (use "git add" and/or "git commit -a")
```

Making Changes

Status of changing the **hello.sh** file

- Git knows of the change to the file but it has not been recorded
- Status message gives hints on what to do next
- Git add to add to the repo
- Git checkout to discard

Note – The step of git add is also called **staging**

Staging Changes

Lets add a change and check the Status

```
$ git add hello.sh

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

    modified:   hello.sh
```

- Change has been staged but not recorded in the repo
- Next commit will include the staged changes
- Git reset to unstage the change

Committing Changes

Can commit using '-m' or using native editor such as vi or emacs

```
$ git commit
[master 843aaac] Adding CL Argument
 1 file changed, 1 insertion(+), 1 deletion(-)
 mode change 100644 => 100755 hello.sh
```

- Commit shows details on what was changed

Check status again:

```
$ git status
On branch master
nothing to commit, working directory clean
```

Staging v/s Committing

Why? – Separation allows to determine what goes into each commit.

```
$git add a.sh
$git add b.sh
$git commit –m "Changes for a and b”

$git add c.sh
$git commit –m "Unrelated change to c”
```

- Make changes till you are ready to commit
- Unrelated file edits can be part of separate commits
- Fine tune commits (logging)

Changes not files

Git tracks changes to the file and not the file itself.

Make a change to **hello.sh** as below:

```
name=$1
echo "Hello," $name
```

Now stage this change:

```
$git add hello.sh
```

Now make another change to have a default value if argument is not passed

```
name=$1
name="${name:=there}"
echo "Hello," $name
```

Changes not files

Now check the status

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

1st change is staged and ready for commit but 2nd is not

Changes not files

Commit the change and recheck status

```
$ git commit -m "Added a variable"
[master cda9b6d] Added a variable
 1 file changed, 4 insertions(+), 1 deletion(-)

$ 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:   hello.sh

no changes added to commit (use "git add" and/or "git commit -a")
```

Now **hello.sh** has unrecorded changes but none in staging

Changes not files

```
$ git add .
```

Now the 2nd change is staged and ready to be committed as well.

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

    modified:   hello.sh
```

Note the use of "." It adds all files in current directory for staging *(Use with Caution)*

Lastly commit the changes

```
$ git commit
```

Logging

Knowing Project History is as important as committing changes

This will give a list of changes done to the repo

```
$ git log
commit cda9b6d96b2104cb33fa993e3d45e8d5c39f2a3b
Author: kishore.kodical <kishore.kodical@ticketmaster.com>
Date:   Mon Apr 18 15:33:46 2016 -0700

    Added a variable

commit 843aaace1ac58062cbb15ecd05f48b3af747e825
Author: kishore.kodical <kishore.kodical@ticketmaster.com>
Date:   Mon Apr 18 10:23:12 2016 -0700

    Adding CL Argument

commit 2e263c78363738cf05615f00dc1de242f1306768
Author: kishore.kodical <kishore.kodical@ticketmaster.com>
Date:   Sun Apr 17 20:07:34 2016 -0700

    First Commit
```

Logging

Log viewing is highly customizable and there are several options available.

```
$ git log --pretty=oneline
cda9b6d96b2104cb33fa993e3d45e8d5c39f2a3b Added a variable
843aaace1ac58062cbb15ecd05f48b3af747e825 Adding CL Argument
2e263c78363738cf05615f00dc1de242f1306768 First Commit
```

The Ultimate git log view:

```
$ git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
* cda9b6d 2016-04-18 | Added a variable (HEAD -> master) [kishore.kodical]
* 843aaac 2016-04-18 | Adding CL Argument [kishore.kodical]
* 2e263c7 2016-04-17 | First Commit [kishore.kodical]
```

Fear not, we will use aliases so we don't have to type so much every time

Logging

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

--pretty="..."	defines the format of the output.
%h	is the abbreviated hash of the commit
%d	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	display the commit tree in an ASCII graph layout
-date=short	-keeps the date format nice and short

Aliases

Useful to have aliases for repeatedly used commands

Add the aliases to `.gitconfig` file in your `$HOME` dir

```
[alias]
  co = checkout
  ci = commit
  st = status
  br = branch
  hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
  type = cat-file -t
  dump = cat-file -p
```

We will talk about the checkout command later

Aliases

More tips and tricks to configure shell aliases:

```
alias gs='git status '
alias ga='git add '
alias gb='git branch '
alias gc='git commit'
alias gd='git diff'
alias go='git checkout '
alias gk='gitk --all&'
alias gx='gitx –all’
alias got='git '
alias get='git '
```

Hashes and Branches

Checkout command will copy any snapshot from history to the working directory

```
$ git hist
* e22dcd9 2016-04-19 | Added default name (HEAD -> master) [kishore.kodical]
* cda9b6d 2016-04-18 | Added a variable (HEAD -> master) [kishore.kodical]
* 843aaac 2016-04-18 | Adding CL Argument [kishore.kodical]
* 2e263c7 2016-04-17 | First Commit [kishore.kodical]
```

2e263c7 is the hash from the 1st commit

Checkout the first commit and examine its contents

Note - Hash values in git will differ from what is pasted here

Hashes and Branches

```
$ git checkout 2e263c7
Note: checking out '2e263c7'.

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 2e263c7... First Commit
```

- Output explains situation pretty well.

Hashes and Branches

Examine the contents of **hello.sh**

```
$ cat hello.sh
```

Notice contents of the script is the original.

Lets return to the master branch

```
$ git checkout master
Previous HEAD position was 2e263c7... First Commit
Switched to branch 'master'
```

Examine the contents of **hello.sh** again

```
$ cat hello.sh
```

- Master is the name of the default branch
- Checking out a branch by name (-b), creates a new branch

Tagging Versions

Commits can be tagged for reference instead of hashes (hard to read)

```
$ git tag v1
```

Current version of the program is referred to as v1

To tag a previous version, we need to check it out first.
Can use ^ instead of the hash – indicates the parent of v1

```
$ git checkout v1^
Note: checking out 'v1^'.

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 cda9b6d... Added a variable
```

Tagging Versions

Check hello.sh now.

$ cat hello.sh - This is the version before we added default name.

Lets tag this version as v1-beta and move between tagged version

```
$ git tag v1-beta

$ git checkout v1
Previous HEAD position was cda9b6d... Added a variable
HEAD is now at e22dcd9... Added default name

$ git checkout v1-beta
Previous HEAD position was e22dcd9... Added default name
HEAD is now at cda9b6d... Added a variable
```

Notice it tells you all which version hash you are in currently.

Tagging Versions

You can see what tags are available in your repo.

```
$ git tag
v1
v1-beta
```

You can also check for tags in the logs

```
$ git hist
* cda9b6d 2016-04-18 | Added a variable (HEAD, tag: v1-beta) [kishore.kodical]
* 843aaac 2016-04-18 | Adding CL Argument [kishore.kodical]
* 2e263c7 2016-04-17 | First Commit [kishore.kodical]
```

You can see the tags available as well as which tag is currently HEAD

Undoing changes (Before Staging)

You make local changes which you want to revert.

```
$ git checkout master
Previous HEAD position was cda9b6d... Added a variable
Switched to branch 'master'

$ cat hello.sh

$ git status
…
Changes not staged for commit:
…
```

$ cat hello.sh

```
#Bad comment
name=$1
name="${name:=there}"
echo "Hello," $name
```

hello.sh has been modified but not staged yet for commit.

Undoing changes (Before Staging)

To Revert, checkout the file and check status.

```
$ git checkout hello.sh

$ git status
On branch master
nothing to commit, working directory clean
```

The bad comment is no longer part of the file.

$ cat hello.sh

```
name=$1
name="${name:=there}"
echo "Hello," $name
```

Undoing changes (Before Committing)

You have staged changes which you want to revert.

```
$ cat hello.sh

$ git add hello.sh

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

    modified:   hello.sh
```

- hello.sh has been staged but not yet committed
- Fortunately, git tells you how to revert

$ cat hello.sh

```
#Will stage but not commit
name=$1
name="${name:=there}"
echo "Hello," $name
```

Undoing changes (Before Committing)

Now revert the changes as recommended by git.

```
$ git reset HEAD hello.sh
Unstaged changes after reset:
M   hello.sh

$ git checkout hello.sh

$ git status
On branch master
nothing to commit, working directory clean
```

$ cat hello.sh

```
name=$1
name="${name:=there}"
echo "Hello," $name
```

- Resets to whatever version of file was in HEAD & Clears the staging area
- Still need to checkout to undo the changes. Now working directory is clean again

Undoing changes (After Committing)

You want to undo what has already been committed.

```
$ cat hello.sh

$ git add hello.sh

$ git commit -m "DO NOT want this commit"
[master 43b7f00] DO NOT want this commit
 1 file changed, 1 insertion(+)
```

hello.sh has been committed.

```
$ git revert HEAD –no-edit
[master a46c0a2] Revert "DO NOT want this commit"
 1 file changed, 1 deletion(-)
```

Use revert to go to the previous commit.

$ cat hello.sh

```
#Bad commit
name=$1
name="${name:=there}"
echo "Hello," $name
```

Undoing changes (After Committing)

You can skip to any desired commit using hash value

```
$ git hist
* a46c0a2 2016-04-20 | Revert "DO NOT want this commit" (HEAD ->
master) [kishore.kodical]
* 43b7f00 2016-04-20 | DO NOT want this commit [kishore.kodical]
…
```

Now check the contents of hello.sh

```
$ cat hello.sh
```

$ cat hello.sh

```
name=$1
name="${name:=there}"
echo "Hello," $name
```

We can remove commits using ‘git reset’ but that is not recommended for shared repositories.

Amending Commits

You want to **fix** a commit.

```
$ cat hello.sh

$ git add hello.sh

$ git commit -m "Add Author Commit"
[master ce453eb] Adding Author
```

$ cat hello.sh

```
#Author: Kishore
name=$1
name="${name:=there}"
echo "Hello," $name
```

```
$ git hist
* ce453eb 2016-04-20 | Adding Author (HEAD -> master) [kishore.kodical]
```

hello.sh has been committed. Oops, we forgot to add email.

Amending Commits

Lets add the email address but this time we **fix** the commit instead of making a new one.

```
$ cat hello.sh

$ git add hello.sh

$ git commit --amend -m "Add Author/Email"
[master 2953014] Add Author/Email
```

--amend lets you have just one commit instead of two.

Review history - Original 'Adding Author' comment replaced

```
$ git hist
* 2953014 2016-04-20 | Add Author/Email (HEAD -> master)
[kishore.kodical]
```

$ cat hello.sh

```
#Author: Kishore/kishore@tm.com
name=$1
name="${name:=there}"
echo "Hello," $name
```

Moving Files

Git lets us easily move files with few commands

```
$ mkdir main

$ git mv hello.sh main/

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

    renamed:    hello.sh -> main/hello.sh
```

Moving Files

Git removed the original file and created main/hello.sh

We could have done:

```
$ mkdir main
$ mv hello.sh main
$ git add main/hello.sh
$ git rm hello.sh
```

You can use operating system commands but it may be more work.

Don't forget to commit:

```
$ git commit –m "Moved hello.sh to main"
```

Working directly with objects

You can dig down starting with a commit using the hashes

```
$ git hist —-max-count=1
* 2953014 2016-04-20 | Add Author/Email (HEAD -> master) [kishore.kodical]
```

Using the hash from the example above

```
$ git cat-file -t 2953014
commit

$ git cat-file -p 2953014
tree 7f80e63dd793bd9efc7b1d8c92ebc08b6fe63698
parent a46c0a265f3a85f99f96786f89fc482682831e4d
author kishore.kodical <kishore.kodical@ticketmaster.com> 1461199818 -0700
committer kishore.kodical <kishore.kodical@ticketmaster.com> 1461199913 -0700
```

You can get the type of hash **(commit, tree or blob)**

Working directly with objects

This is the dump of the commit object from master branch.

```
$ git cat-file -p 7f80e63dd
100755 blob 52f44c51ee0a4fd69a2e03ef339499bf0eb63c93    hello.sh

$ git cat-file -p 52f44c51
#Author: Kishore/kishore@tm.com
name=$1
name="${name:=there}"
echo "Hello," $name
```

Try using the **-t** option on the other hashes

Creating a Branch

Create a new branch to isolate changes from master

```
$ git checkout –b test
Switched to a new branch 'test'
```

Tip: can do git branch <name> followed by git checkout <name>

```
$ git status
On branch test
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    renamed:    hello.sh -> main/hello.sh
```

Add a file **error.sh** to the repo and modify **hello.sh** to use that.

Creating a Branch

$ cat main/error.sh

```
#!/bin/bash
error () {
        echo "Error:$?"
        echo "Please enter one name only"
        exit
}
```

$ cat main/hello.sh

```
#Author: Kishore/kishore@tm.com
source error.sh

if [ $# -gt 1 ]; then
 error
fi
name=$1
name="${name:=there}"
echo "Hello," $name
```

Creating a Branch

Now stage and commit the changes to both files.

```
$ git add main/error.sh

$ git commit -m "Added error file”
[test 1306fac] Added error file
 1 file changed, 7 insertions(+)
 create mode 100644 main/error.sh

$ git add main/hello.sh

$ git commit -m "Updating hello.sh"
[test d62cbc0] Updating hello.sh
 1 file changed, 5 insertions(+)
```

We now have a branch **‘test’** with 2 new commits in it

Navigating a Branch

Now that we have more than one branch, lets move around branches

```
$ git branch
  master
* test
```

Get the list of branches and git tells you which branch you are on

```
$ git checkout master
Switched to branch 'master'
$ cat main/hello.sh
```

You can confirm by checking the hello.sh file, will be the original
Now go back to the test branch

```
$ git checkout test
Switched to branch 'test'
$ cat main/hello.sh
```

The contents of hello.sh now confirm, we are back in the test branch

Viewing Branches

When branches go off with their changes, need to merge different and potentially conflicting changes

Add a new file in the master branch

```
$ cat main/README
This is the Hello World Program

$ git checkout master

$ git add main/README

$ git commit -m "Added README”
```

Viewing Branches

View the diverging branches

```
$ git hist –all
* 8f9b656 2016-04-21 | Added README (HEAD -> master) [kishore.kodical]
* 068e15d 2016-04-21 | Added main directory [kishore.kodical]
| * d62cbc0 2016-04-21 | Updating hello.sh (test) [kishore.kodical]
| * 1306fac 2016-04-21 | Added error file [kishore.kodical]
| * 19129bd 2016-04-21 | Added main directory [kishore.kodical]
|/
* 2953014 2016-04-20 | Add Author/Email [kishore.kodical]
* a46c0a2 2016-04-20 | Revert "DO NOT want this commit" [kishore.kodical]
* 43b7f00 2016-04-20 | DO NOT want this commit [kishore.kodical]
* e22dcd9 2016-04-19 | Added default name (tag: v1) [kishore.kodical]
```

- The **–graph** option in the **hist** alias is in action that makes viewing branches easier.
- We can see both branches and master branch is current HEAD
- **--all** shows all branches. Default is to show current branch

Merging Branches

When branches go off with their changes, need to merge different and potentially conflicting changes

Go to the test branch and merge master onto test.

```
$ git checkout test
Switched to branch 'test'

$ git merge master
Merge made by the 'recursive' strategy.
 main/README | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 main/README
```

Merging brings the edits in diverging branches together.

Merging Branches

Now take a look at history.

```
$ git hist –all
*   a61d483 2016-04-22 | Merge branch 'master' into test (HEAD -> test) [kishore.kodical]
|\
| * 8f9b656 2016-04-21 | Added README (master) [kishore.kodical]
| * 068e15d 2016-04-21 | Added main directory [kishore.kodical]
* | d62cbc0 2016-04-21 | Updating hello.sh [kishore.kodical]
* | 1306fac 2016-04-21 | Added error file [kishore.kodical]
* | 19129bd 2016-04-21 | Added main directory [kishore.kodical]
|/
* 2953014 2016-04-20 | Add Author/Email [kishore.kodical]
* a46c0a2 2016-04-20 | Revert "DO NOT want this commit" [kishore.kodical]
…
* 843aaac 2016-04-18 | Adding CL Argument [kishore.kodical]
* 2e263c7 2016-04-17 | First Commit [kishore.kodical]
```

- Merging master onto test, pickup changes to master
- Keep changes in test in the mainline.
- Creates ugly commit graphs.

Conflicts in Git

There is a good chance you will run into conflicts during merge.

Make a change in master branch that will conflict with the test branch

```
$ git checkout master

$ cat hello.sh
#Author: Kishore
MYNAME=$1
MYNAME="${name:=there}"
echo "Hello," $MYNAME

$ git add main/hello.sh

$ git commit –m “Change variable name”
[master 47b5db5] Variable Name Change
 1 file changed, 2 insertions(+), 2 deletions(-)
```

Conflicts in Git

Now go to the test branch and attempt to merge

```
$ git checkout test
Switched to branch 'test'

$ git merge master
Auto-merging main/hello.sh
CONFLICT (content): Merge conflict in main/hello.sh
Automatic merge failed; fix conflicts and then commit the result.
```

- Conflict message will point you to file needing attention.
- Edit the file to match what is in master to fix conflicts.

Resolving Conflicts

- The first section is the version on the HEAD of the current branch (test).
- The second section is the version on the master branch.
- The rest is common to both branches.
- Need to manually resolve the conflict by fixing hello.sh in the current branch.

```
$ cat hello.sh
#Author: Kishore/kishore@tm.com
<<<<<<< HEAD
source error.sh

if [ $# -gt 1 ]; then
 error
fi
name=$1
name="${name:=there}”
echo "Hello," $name
=======
MY_NAME=$1
MY_NAME="${name:=there}"
echo "Hello," $MY_NAME
>>>>>>> master
```

Resolving Conflicts

Now commit the the changes and you will be able to merge successfully.

```
$ git add hello.sh
```

```
$ git commit -m "Merged master fixed conflict"
[test 1cbfa00] Merged master fixed conflict
```

Once edited try merging again.

```
$ git merge master
Already up-to-date.
```

Cloning Repositories

You want to make a copy of your repository.

Go to the main directory and clone your repository:

```
$ cd ..

$ ls
hello

$ git clone hello cloned_hello
Cloning into 'cloned_hello'...
done.

$ ls
hello        cloned_hello
```

Check the cloned repo and it should be a copy of the original
Check the log and the only difference should be the names of branches.

Notice the change:

```
* c6c4d0b 2016-04-22 | Change variable name (HEAD -> master, origin/master, origin/
HEAD) [kishore.kodical]
```

Cloned Repositories

In the cloned repo, run:

```
$ git remote
origin
```

The cloned repo knows about a remote repo called origin

```
$ git remote show origin
* remote origin
  Fetch URL: /Users/kishore/git/hello
  Push  URL: /Users/kishore/git/hello
  HEAD branch: master
  Remote branches:
    master tracked
    test   tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

Cloned Repositories

From the behavior of cloning, we see:

- Remote repo '**origin**' is the original '**hello**' repo
- Remote repository generally is in a centralized server but could be local too
- Depends on where the current repository was cloned from

Remote Braches

Look at the branches available in the cloned repo:

```
$ git branch
* master
```

The test branch does not exist here since it's a cloned branch.

Branch command shows local branches only

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/test
```

- Branches in remote repo are not treated as local branches.
- Must create local branches separately

Fetching Changes

Lets make a change to the original repo so we can pull it to cloned repo

Go to the '**hello**' repo and modify the README file

```
$ git add main/README
$ git commit -m "Changed README in original repo"
```

$ cat main/README
This is the Hello World Program
(changed in original)

Now go back to the cloned repo and pull the changes from remote repo

```
$ cd ../cloned_hello/
$ git fetch
...
From /Users/kishore/git/hello
   c6c4d0b..94049ce  master     -> origin/master
```

Fetch command gets new commits from remote repo but does not merge it into the local branch

Fetching Changes …

Check the history and see the difference

$ git hist --all
* 94049ce 2016-04-27 | Changed README in original repo (origin/master, origin/HEAD) [kishore.kodical]

- All commits from original repo available but not integrated
- Notice commit includes *origin/master* and *origin/HEAD*
- Local master branch points to '*Change variable name*' and not newly fetched commit.

Check and the README file would not have changed
$ git add main/README
$ git commit -m "Changed README in original repo"

$ cat main/README
This is the Hello World Program

- Changes fetched but not merged

Merging Changes

Lets merge the already pulled changes into the current branch

Lets check status before we do that

```
$ git status
…Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)…
```

```
$ git merge origin/master
Updating c6c4d0b..94049ce
Fast-forward
 main/README | 1 +
 1 file changed, 1 insertion(+)
```

Check and the README file has changed

$ cat main/README
This is the Hello World Program
(changed in original)

- Git fetch does not merge the changes but we can manually merge the changes

Pulling Changes

Can do a **pull** instead of fetch/merge

Notice git status from previous slide

```
$ git status
…Your branch is behind 'origin/master' by 1 commit, and can be fast-
forwarded.
  (use "git pull" to update your local branch)…
```

Git tells you to do a git pull.

This one step replaces the need to do fetch and manual merge

```
$ git pull origin/master
```

Make another edit to the main repo and test this.

Remote Repos

Lets put our project in GitHub - http://git.tm.tmcs/

First need to create your repo in Git via the GUI
http://git.tm.tmcs/projects/new
Name it **hello**.

Git provides an API to create remote repos but it cannot be done via 'git' command.
Once the repo is created, push your local repo into it.

```
$ cd hello/
$ git init
Reinitialized existing Git repository in /Users/kishore/git/hello/.git/
```

Remote Repos

```
$ git remote add origin git@git.tm.tmcs:Kishore.Kodical/hello.git

$ git commit
On branch master
nothing to commit, working directory clean

$ git push -u origin master
Counting objects: 33, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (19/19), done.
Writing objects: 100% (33/33), 2.86 KiB | 0 bytes/s, done.
Total 33 (delta 2), reused 0 (delta 0)
To git@git.tm.tmcs:Kishore.Kodical/git-class.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Remote Repos

To verify the remote URL

```
$ git remote –v
origin  git@git.tm.tmcs:Kishore.Kodical/git-class.git (fetch)
origin  git@git.tm.tmcs:Kishore.Kodical/git-class.git (push)
```

To locally remove the remote URL

```
$ git remote rm origin
```

Remote Repos Mistakes

Git remote will accept any URL

What if you entered an incorrect URL

```
$ git remote add origin http://git.tm.tmcs/u/Kishore.Kodical/

$ git push -u origin master
fatal: http://git.tm.tmcs/u/Kishore.Kodical/info/refs not valid
```

- The URL is incomplete and hence its giving a reference error

What if you did not create the repo first

```
$ git remote add origin git@git.tm.tmcs:Kishore.Kodical/something.git

$ git push -u origin master
GitLab: The project you were looking for could not be found.
```

- The repo does not exist in the project

References

Git Setup - https://goo.gl/Fu5Bz0

My favorite lookup - http://rogerdudler.github.io/git-guide/

THANK YOU

ticketmaster®