

Branching and Merging

Objectives

- Undo uncommitted changes
- Revert Committed Changes
- Update last commit message


```
$ vim file1
```

```
[edit file, change contents]
```

```
$ git status
```


Git status

```
Gouravs-MacBook-Pro:myapp gouravshah$ git status
On branch master
Your branch is up-to-date with 'origin/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)

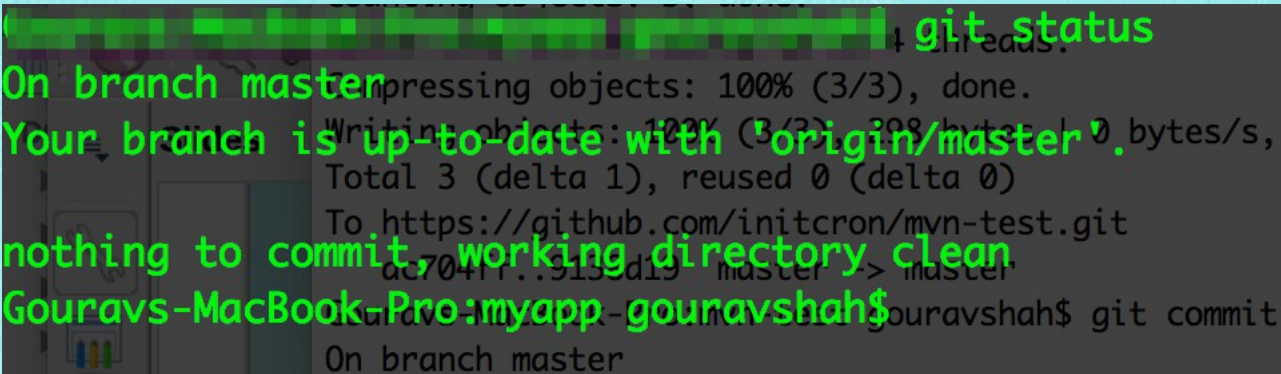
        modified:   file1

nothing to commit, working directory clean
Gouravs-MacBook-Pro:myapp gouravshah$ git push origin
```



```
$ git checkout file1
```

```
$ git status
```



A terminal window screenshot with a dark background. The text is white, and several lines are overlaid with green text. The green text reads: "git status", "On branch master", "Your branch is up-to-date with 'origin/master'", "nothing to commit, working directory clean", and "Gouravs-MacBook-Pro:myapp gouravshah\$". The white text in the terminal shows the output of the git status command, including "On branch master", "nothing to commit, working directory clean", and "Gouravs-MacBook-Pro:myapp gouravshah\$ git commit".

```
On branch master
nothing to commit, working directory clean
Gouravs-MacBook-Pro:myapp gouravshah$ git commit
```


Reverting Committed Changes

```
$ vim file1  
[edit file, change contents]  
$ git commit file1 -m  
"modifying file1"  
$ git log
```


Soft Reset

```
$ git log
```

```
$ git reset --soft HEAD~
```

```
$ git log
```

```
$ git status
```


Reset

```
$ git status
```

```
$ git reset HEAD~
```

```
$ git status
```

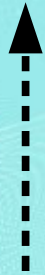
```
$ git log
```


Hard Reset

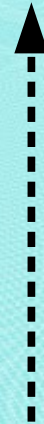
```
$ git status  
$ git reset --hard HEAD~  
$ git status  
$ git log
```


Create a develop branch

```
$ git branch develop
```



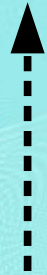
Create
branch



Branch
name

Switch to develop branch

```
$ git checkout develop
```



Create
branch



Branch
name

Create and switch

```
$ git checkout -b develop
```



-b option with checkout will combine creating and switching to branch operations. Above command will create a new branch called develop and switch to it

Check current branch

```
Gouravs-MacBook-Pro:myapp gouravshah$ git branch
develop
* master
Gouravs-MacBook-Pro:myapp gouravshah$ git checkout develop
Switched to branch 'develop'
Gouravs-MacBook-Pro:myapp gouravshah$ git branch
* develop
master
Gouravs-MacBook-Pro:myapp gouravshah$
```

A terminal window screenshot showing the process of switching to the 'develop' branch in Git. The first command 'git branch' shows 'develop' as the current branch and '* master' as the previous one. The second command 'git checkout develop' is highlighted with an orange box, and the output 'Switched to branch 'develop'' is shown. The third command 'git branch' is executed, and the output shows '* develop' as the current branch, which is also highlighted with an orange box and pointed to by an orange arrow, with 'master' listed below it. The prompt 'Gouravs-MacBook-Pro:myapp gouravshah\$' is visible at the end of the last line.

Exercise

- Create two branches
 - br1
 - br2

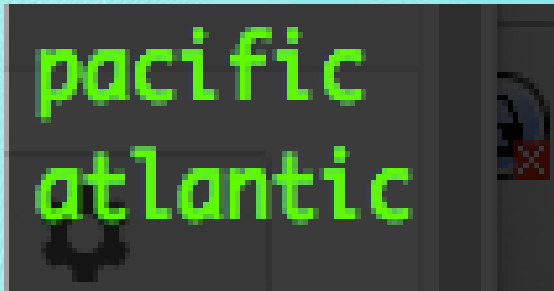
Dont switch to any of these branches, keep curent branch as develop

Working on develop

Lets create a new file, add and push it to develop branch

Exercise

- Create file2 with following contents



```
pacific  
atlantic
```

- Add and push to develop branch


```
$ vim file2
```

```
[edit file, add content]
```

```
$ git add file2
```

```
$ git commit file2
```


Switch to develop branch

```
$ git push origin develop
```



Branch
name



While working with multiple branches, its pertinent to specify the branch name, else git will assume “master” branch

Difference between branches

```
$ git branch
```

```
$ ls
```

```
$ git checkout master
```

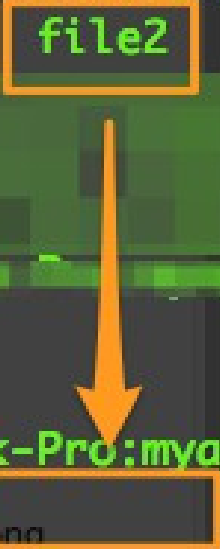
```
$ git branch
```

```
$ ls
```


develop vs master

```
Gouravs-MacBook-Pro:myapp gouravshah$ git branch
* develop
  master
Gouravs-MacBook-Pro:myapp gouravshah$ ls
README file1 file2
Gouravs-MacBook-Pro:myapp gouravshah$ git branch
* develop
  master
Gouravs-MacBook-Pro:myapp gouravshah$ ls
README file1
Gouravs-MacBook-Pro:myapp gouravshah$
```

file2 missing in master

An orange arrow points from the 'file2' box in the first terminal output to the empty box in the second terminal output, illustrating that 'file2' is present in the 'develop' branch but missing in the 'master' branch.

Merging

- Merging will integrate changes from another branch into current branch

From master branch

```
$ git merge develop
```

```
$ git push origin master
```


Merge Conflicts

- Conflicts could happen when same line inside file is modified in two different branches
- Lets simulate merge conflict

From master branch

```
$ git branch checkout master
```

```
$ echo "master" >> file2
```

```
$ git commit file2 -m "changed from  
master"
```

```
$ git push origin master
```


From develop branch

```
$ git checkout develop
```

```
$ echo "develop" >> file2
```

```
$ git commit file2 -m "changed from  
develop"
```

```
$ git push origin develop
```


merge

```
$ git checkout master
```

```
$ git merge develop
```

```
Auto-merging file2
```

```
CONFLICT (content): Merge conflict in file2
```

```
Automatic merge failed; fix conflicts and then commit the result.
```


Ways to fix conflict

Option 1

- Edit file and manually fix

```
pacific
atlantic
<<<<<<< HEAD
master

=====
develop
>>>>>>> develop
```


Option 2

- Select changes from either our branch or theirs

```
$ git checkout --theirs file2  
$ git checkout --ours file2
```


Fix Conflict

```
$ git checkout --theirs file2
```

```
$ git status
```

```
$ git add file2
```

```
$ git commit
```

```
$ git push origin master
```


From master branch

```
$ git merge develop
```

```
$ git push origin master
```


Exercise

- Create a new feature branch called “script1”
- Add a bash scripts script1.sh
- Commit changes
- Merge branch to master

file: script1.sh

```
#!/bin/bash  
echo "script1"  
mkdir -p /opt/myapp
```


Deleting branches

```
$ git branch --delete br1  
$ git branch --delete br2
```


Update production

```
$ vim file1.sh  
[ change green => pink ]
```

```
$ vim script2.sh
```

Add the following

```
#!/bin/bash  
echo "script2"
```

```
$ git add script2
```


Git status

C

```
Gouravs-MacBook-Pro:myapp gouravshah$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
new file:   script2
```

```
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:   file1
```

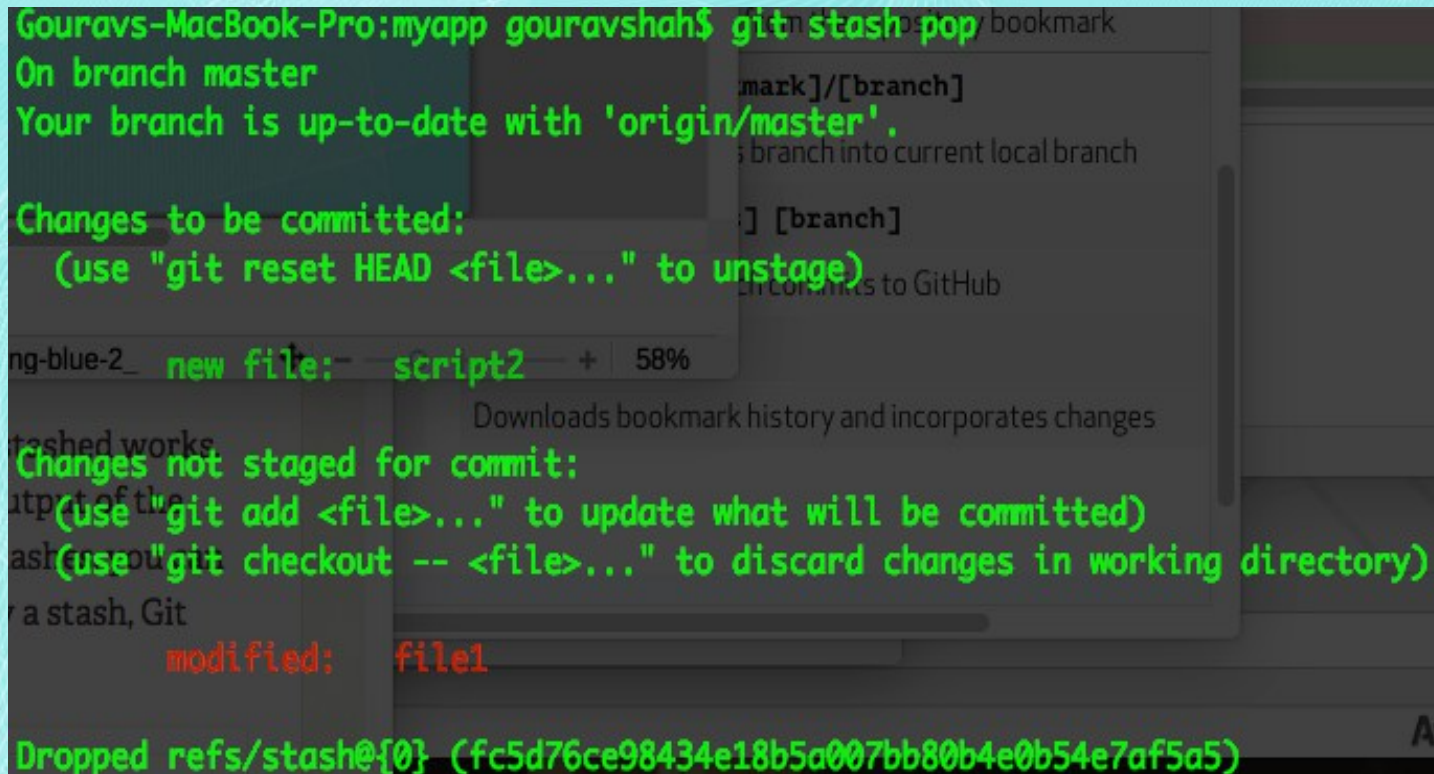

stash

```
$ git stash  
$ git status
```

```
Gouravs-MacBook-Pro:myapp gouravshah$ git status  
On branch master  
Your branch is up-to-date with 'origin/master'.  
  
nothing to commit, working directory clean
```


Bringing back stashed changes

```
$ git stash pop
```



A terminal window screenshot showing the output of the `git stash pop` command. The text is as follows:

```
Gouravs-MacBook-Pro:myapp gouravshah$ git stash pop
On branch master
Your branch is up-to-date with 'origin/master'.

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

    new file:   script2

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:   file1

Dropped refs/stash@{0} (fc5d76ce98434e18b5a007bb80b4e0b54e7af5a5)
```

The screenshot also shows a diff view for 'script2' with a 58% similarity to a previous version, and a file named 'file1' that has been modified. The terminal title bar indicates the user is in the 'myapp' directory on a MacBook Pro.

Tagging

```
$ git tag -l  
$ git tag -a v1.0 -m "my first release"  
$ git push --tags
```


Synchronize

```
$ git pull origin master  
$ git push origin master
```