

Version Control



Git Commands – Chapter 3



KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Deepak K.C. ; deepak.kc@kamk.fi ;

Git basics

- *Most operations only need local files & resources to operate*
 - Generally no information is needed from another computer in your network
 - Entire history of the project on a local disk
 - To check the history of the project, Git does not need to go out to the server, simply reads it from a local browser
 - Possible to continue working even offline and commit after getting to a network connection

Git Basics

- *Git has integrity: Everything in git is checksummed before it is stored*
 - This makes it impossible to change the contents of any file without git knowing it
 - SHA-1 hash mechanism is used for checksumming
 - Git stores everything in its database based on its hash value of its contents rather than a file name

Three states files can reside in Git

➤ Committed

➤ Data is safely stored in your local database

➤ Modified

➤ Have changed the file but is not committed into the database yet

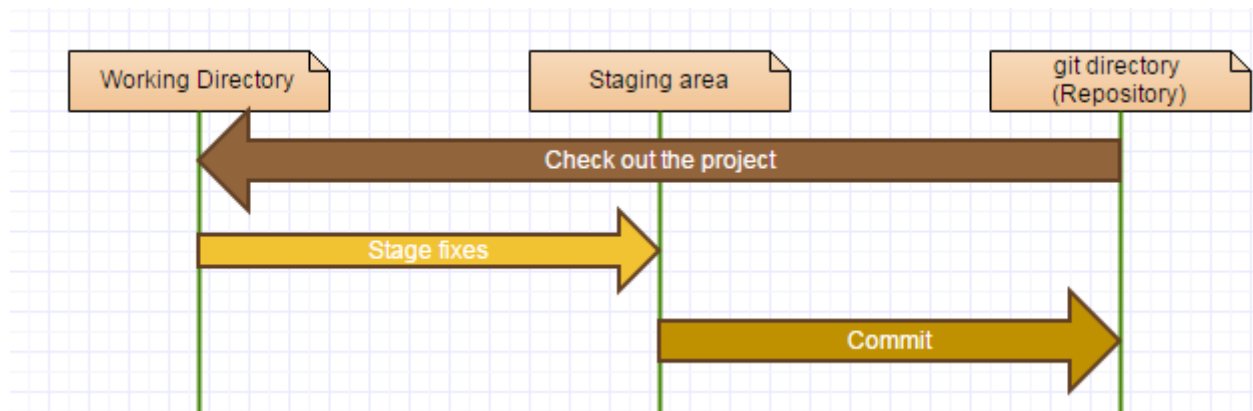
➤ Staged

➤ A modified file is marked in its current version to go into your next commit snapshot

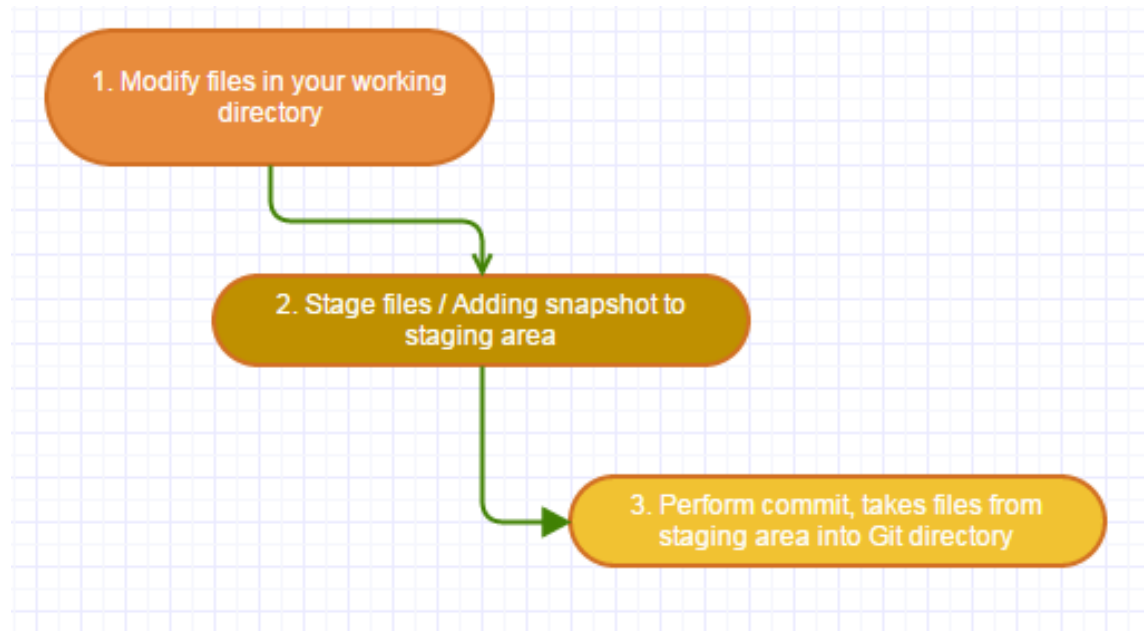
These 3 states of file forms 3 sections of a Git project: the git directory, working directory & the staging area

Working directory, staging area and git directory (repository)

- Working directory
 - Single checkout of one version of the project
 - Files are pulled out of the compressed database in the git directory & placed on disk for you to use or modify
- Staging area
 - It is a file generally contained in a git directory
 - Stores information about what should go into your next commit
 - Sometimes also referred to as the “index”
- Git directory (repository)
 - Where metadata & object database for a project is stored
 - Most important part of Git
 - It is what is copied when cloning a repository from another computer



Basic git workflow



Git vs. Github

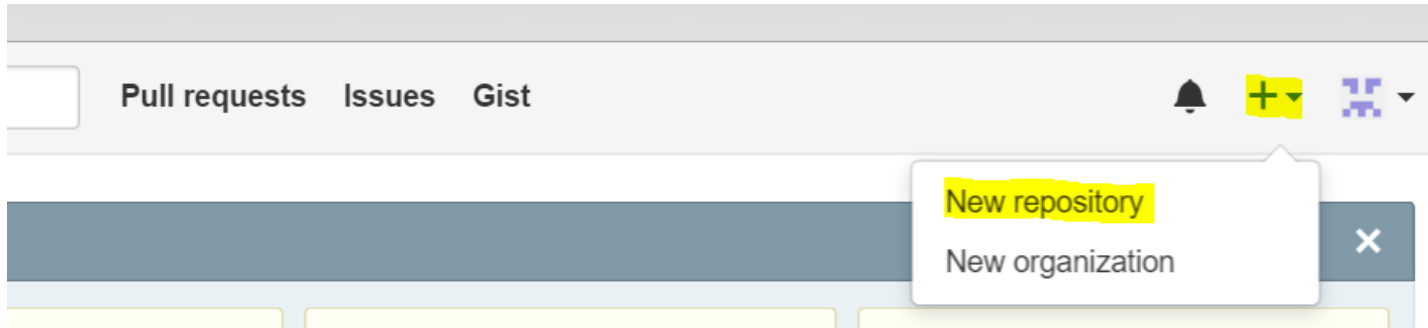
- GitHub allows to
 - Share your repositories
 - Access other users' repositories
 - Store remote copies of your local repositories on GitHub's server
- You don't need GitHub(remote on the web) to use Git (Local on your computer)

Creating a GitHub Repository

- Two methods
 - Starting a repository from scratch
 - Forking another user's repository

Starting a repo from scratch

➤ Log in to your GitHub account & Click on New Repository



Starting a repo from scratch


- Create a name for your repo & describe it briefly
- Select “Public” (Private requires a paid or education account)
- Check the box next to “Initialize this repository with a README”
- You can add a license and .gitignore to your repo
- Click the “Create Repository” button

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

Repository name

 dipaish ▾

/

Great repository names are short and memorable. Need inspiration? How about **didactic-winner**.

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾ ⓘ

Create repository

Creating a local copy

- To be able to make changes to the repo, you need to create a local copy of this repo on your computer
 - Open Git Bash
 - Create a directory on your computer where you will store your copy of the repo – **mkdir test**
 - Navigate to this new directory – **cd test**
 - Initialize a local Git repository in this directory – **git init**
 - Point your local repository at the remote repository you just created on the GitHub server
 - **git remote add origin <https://github.com/username/test.git>**

Creating a local copy

MINGW64:/c/Users/deepakkc/test-repository



```
deepakkc@DESKTOP-EF1P61T MINGW64 ~  
$ pwd  
/c/Users/deepakkc  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~  
$ mkdir test-repository  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~  
$ cd test-repository/  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~/test-repository  
$ git init  
Initialized empty Git repository in C:/Users/deepakkc/test-repository/.git/  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~/test-repository (master)  
$ git remote add origin https://github.com/dipaish/test.git  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~/test-repository (master)  
$
```

Forking another user's repo

- Forking : the process of making a copy of another user's repo
- Forking is an important aspect of open source software development
- By navigating to the desired repository on the GitHub website, to fork click the fork button
- Once you click on the Fork button, you have a copy of the desired repository on your GitHub account



Clone the Repo

- Cloning: once a repo is forked, to have it locally on your computer, you need to clone the repo
 - **git clone** <https://github.com/username/testrepo.git>
 - **This will clone the repo into your current directory**

Clone the repo

```
deepakkc@DESKTOP-EF1P61T MINGW64 ~  
$ pwd  
/c/Users/deepakkc  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~  
$ mkdir clone_repo  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~  
$ cd clone_repo  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~/clone_repo  
$ git clone https://github.com/dipaish/game.git  
Cloning into 'game'...  
remote: Counting objects: 116, done.  
remote: Total 116 (delta 0), reused 0 (delta 0), pack-reused 116  
Receiving objects: 100% (116/116), 94.34 KiB | 0 bytes/s, done.  
Resolving deltas: 100% (60/60), done.  
Checking connectivity... done.  
  
deepakkc@DESKTOP-EF1P61T MINGW64 ~/clone_repo  
$ |
```

Basic Git Commands



Git – getting help

➤ Git help <verb>

➤ Example: Git help config

Git basics

➤ Getting a Git repository

➤ Two main approaches

- First: Taking an existing project or directory & importing into Git
- Second: Cloning an existing Git repository from another server

➤ Initializing a repository in an existing directory

- To track an existing project in Git, get into the project's directory & type
 - `$ git init`
 - This creates a new subdirectory `.git` and contains all of your necessary repository file (a git repository skeleton)
 - But nothing is tracked yet
 - To start version controlling existing files, you should begin tracking those file and do an initial commit
 - Add command is used to specify files you want to track, followed by a git commit
 - `Git add *.c`
 - `Git add License`
 - `Git commit -m 'initial project version'`
 - Now we have a Git repository with tracked files & an initial commit.

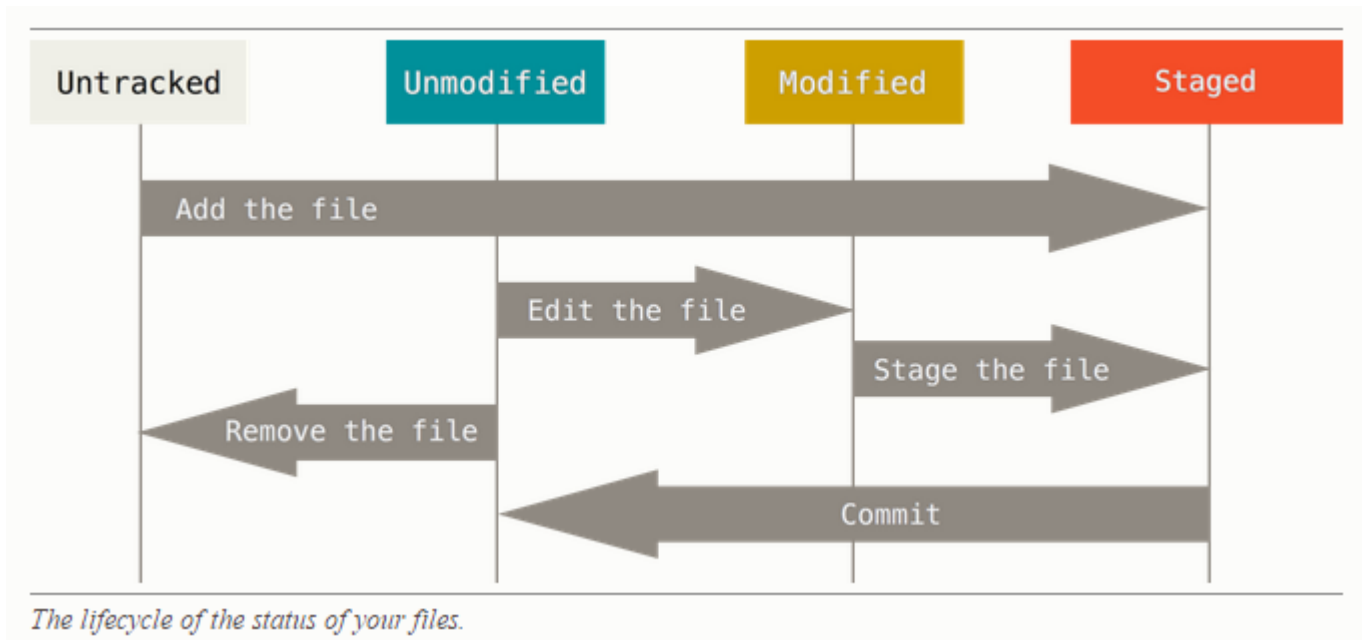
Cloning an existing repository

- Git clone to get a copy of an existing Git repository
- On running git clone Every version of every file for the history of the project is pulled down by default
- Examples
 - git clone <https://github.com/libgit2/libgit2>
 - to clone the repository into a directory named something other than “libgit2”, you can specify that as the next command-line option:
 - git clone https://github.com/libgit2/libgit2 mylibgit

Recording changes to the repository

- After making changes, snapshots of those changes should be committed into a repository each time the project reaches a state you want to record
- Each file in working directory can have two states
 - Tracked: files from the last snapshot, can be modified, unmodified or staged
 - Untracked: other files in the working directory which were not in your last snapshot & not in your staging area
- On cloning repository for the first time
 - All files are tracked and unmodified
 - On modifying them, git sees them as modified, you then stage these modified files and then commit all your staged changes. (repetitive cycle)

Life cycle of the status of your files



Checking the status of your files

➤ Clone this game git clone

➤ git clone <https://github.com/jirick1987/tictactoe>

➤ To check the status, get into the tictactoe directory

➤ Git status

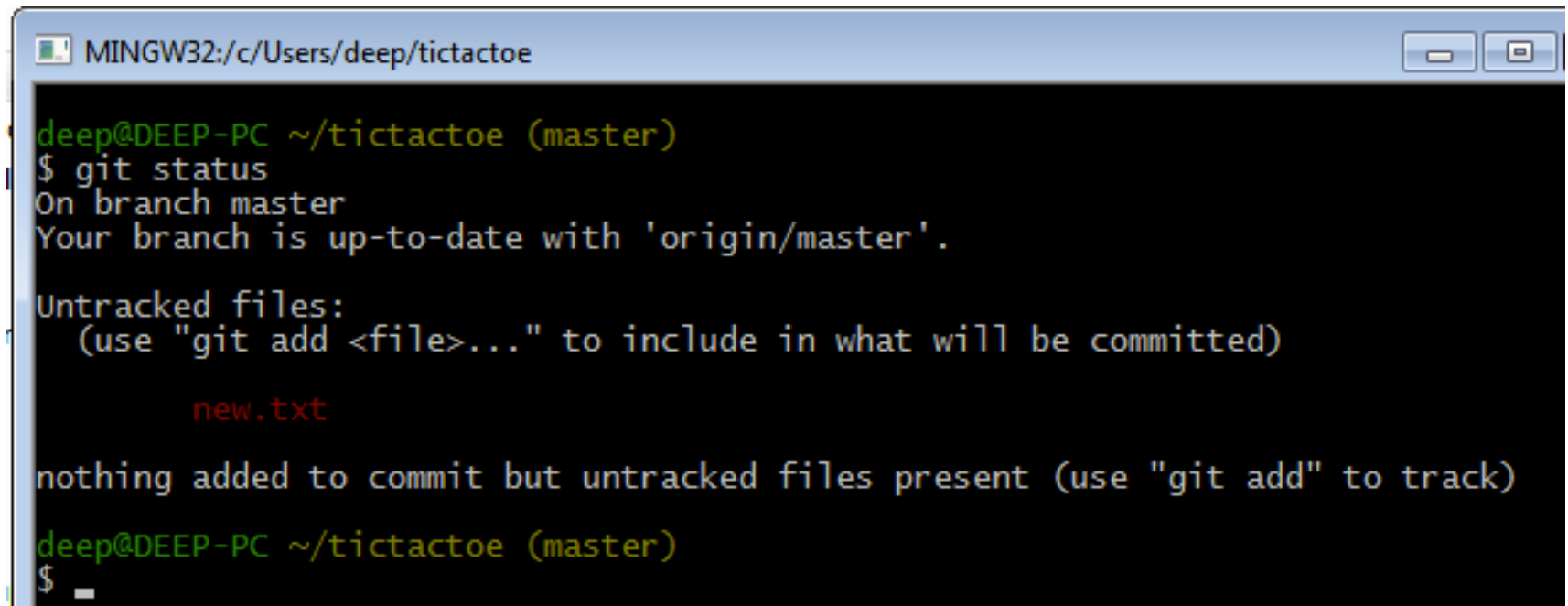
```
deep@DEEP-PC ~/tictactoe (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

deep@DEEP-PC ~/tictactoe (master)
$
```

Adding new file to the project directory

- Create a file in your project directory & check the status

A terminal window titled 'MINGW32:/c/Users/deep/tictactoe' showing the output of the 'git status' command. The output indicates the user is on the 'master' branch, which is up-to-date with 'origin/master'. It lists 'new.txt' as an untracked file and suggests using 'git add' to track it. The prompt '\$' is shown at the bottom.

```
MINGW32:/c/Users/deep/tictactoe

deep@DEEP-PC ~/tictactoe (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

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

        new.txt

nothing added to commit but untracked files present (use "git add" to track)

deep@DEEP-PC ~/tictactoe (master)
$
```

This newly created file is untracked. It is needed to tell Git to include in your Snapshots by using `git add` command. To begin tracking you need to run the following

```
deep@DEEP-PC ~/tictactoe (master)
$ git add new.txt
```

After adding the file, the status changes to

```
deep@DEEP-PC ~/tictactoe (master)
$ 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)

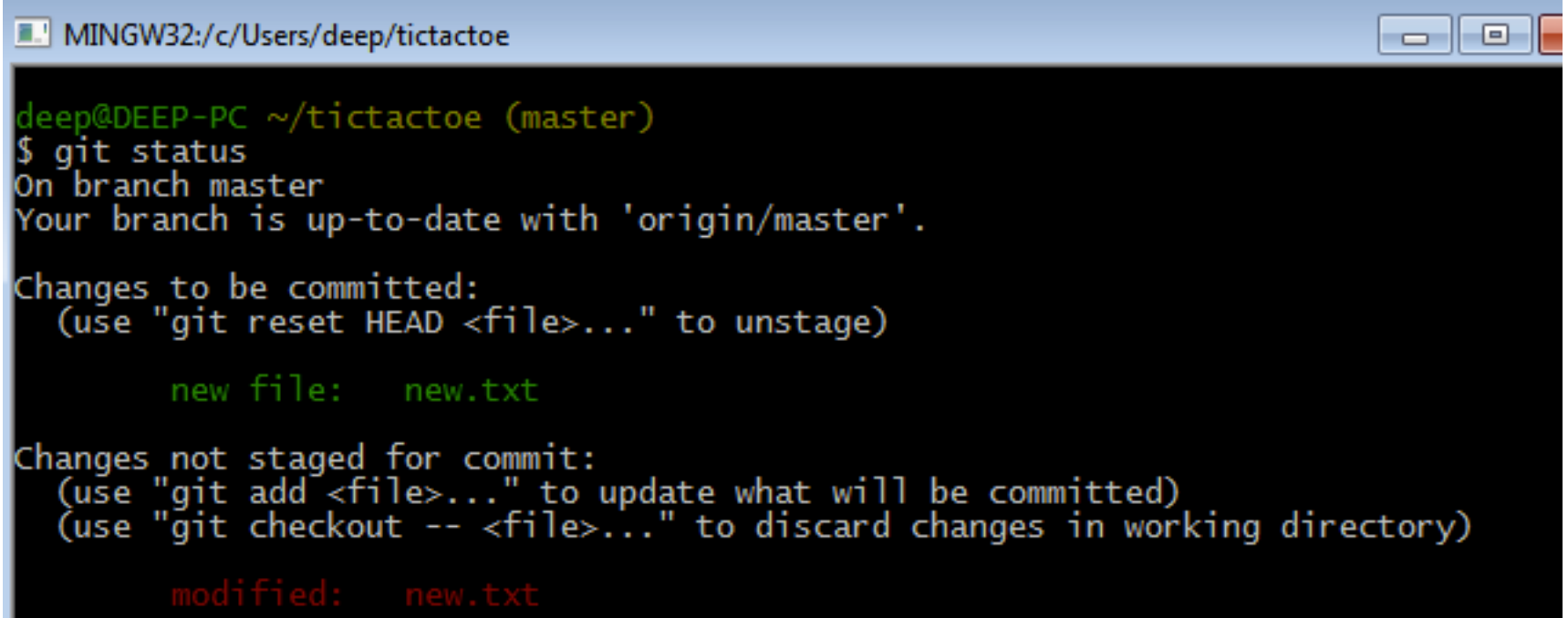
    new file:   new.txt
```

Tracking new files

- Git add command is used to begin tracking a new file
- Git add filename

Staging modified files

- Lets change a file (new.txt) that is already being tracked
 - Lets add some new line of text to it or just make some modification to the text file
 - Now check the status

A terminal window titled 'MINGW32:/c/Users/deep/tictactoe' showing the output of the 'git status' command. The output indicates that the branch 'master' is up-to-date with 'origin/master' and lists 'new.txt' as a new file staged for commit. It also shows instructions for unstaging, adding, or discarding changes.

```
MINGW32:/c/Users/deep/tictactoe

deep@DEEP-PC ~/tictactoe (master)
$ 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:   new.txt

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:   new.txt
```

Staging modified files

- The status mentions that a file new.txt is modified in the working directory but not yet staged.
- To stage, we again run the git add command which is a multipurpose command that is used to
 - Begin tracking new files
 - To stage files
 - Marking merge-conflicted files as resolved
- Now run the following command
 - Git add new.txt
 - Check the status>> git status

```
deep@DEEP-PC ~/tictactoe (master)
$ git add new.txt

deep@DEEP-PC ~/tictactoe (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

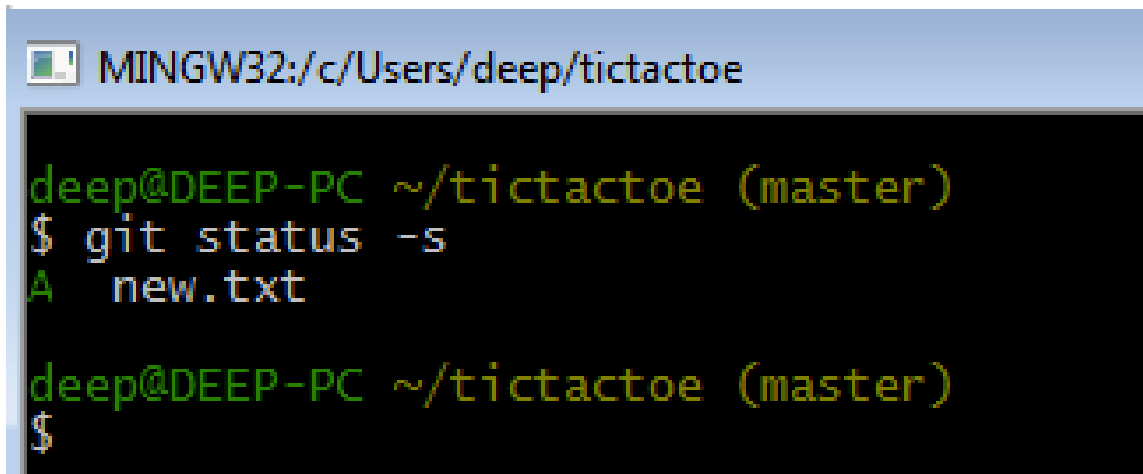
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   new.txt
```

*Now, new file is staged
& will go into your next commit*

Short status

- Git has a short status flag
- Possible to see changes in more compact way
- To get more simplified output type the following command
 - Git status -s or
 - Git status --short



```
MINGW32:/c/Users/deep/tictactoe

deep@DEEP-PC ~/tictactoe (master)
$ git status -s
A  new.txt

deep@DEEP-PC ~/tictactoe (master)
$
```

Ignoring files

- There will be class of files such as log files or files produced by the build system which you don't want Git to automatically add or even show you as being untracked
- To ignore files, you can create a file listing patterns to match them named `.gitignore`
- Cat command:
 - Used to display, copy, combining and creat text files
 - Syntax `cat filename`

Example of ignoring files

```
$ cat .gitignore
*.[oa]
*~
```

*.[oa] → first line tells to ignore any files ending in .o or .a

*~ → second line tells to ignore any files that ends with ~

```
# no .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the root TODO file, not subdir/TODO
/TODO

# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .txt files in the doc/ directory
doc/**/*.txt
```

GitHub maintains a fairly comprehensive list of good .gitignore file examples for dozens of projects and languages at <https://github.com/github/gitignore> if you want a starting point for your project.

Viewing staged & unstaged changes

- Git diff command is used to know what exactly you have changed.
- Git diff shows the exact lines added & removed
- Git status only mentions which files were changed

```
deep@DEEP-PC ~/tictactoe (master)
$ 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:   new.txt

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:   new.txt

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

    .gitignore.swp
```

```
deep@DEEP-PC ~/tictactoe (master)
$ git diff
diff --git a/new.txt b/new.txt
index 413b507..bf803d6 100644
--- a/new.txt
+++ b/new.txt
@@ -1,2 +1,3 @@
 new file
 -new line is added that is modified
 \ No newline at end of file
 +new line is added that is modified
 +one more line is added
 \ No newline at end of file
```

Git diff

- To see what have been changed but not yet staged
 - `Git diff`
- To see what haven been changed and will go into your next commit
 - `Git diff --staged`
- To see what have been staged so far (`--staged` and `--cached` are synonyms)

Committing your changes

- Once everything is staged, next step is to commit changes.
- Remember to run `git add` on files that have been created or modified as anything that is still unstaged won't go into this commit.
- **`Git commit -m "message"`**
 - **Message is description of what you did**
 - **This updates your local repo not the remote repo on GitHub.**

Pushing

- Since local commits are now saved, you would like to update on the remote (GitHub)
 - **git push**
 - **Provide your github username and password**

Branches

- When working on a project with a version that is being used by many people, you may not want to edit that version so you can create a branch
 - `git checkout -b branchname`
- To see what branch you are on type
 - `git branch`
- To switch back to the master branch
 - `git checkout master`

Pull requests

- Pull request is done when you fork someone's repo or have multiple branches and working separately
- Sometimes you want to merge your changes into other branch/repo, you need to send a pull request
- Pull request is a feature of Github

References

- <http://git-scm.com/doc>
- <http://guides.beanstalkapp.com/version-control/intro-to-version-control.html>
- <https://www.atlassian.com/git/tutorials/using-branches/>