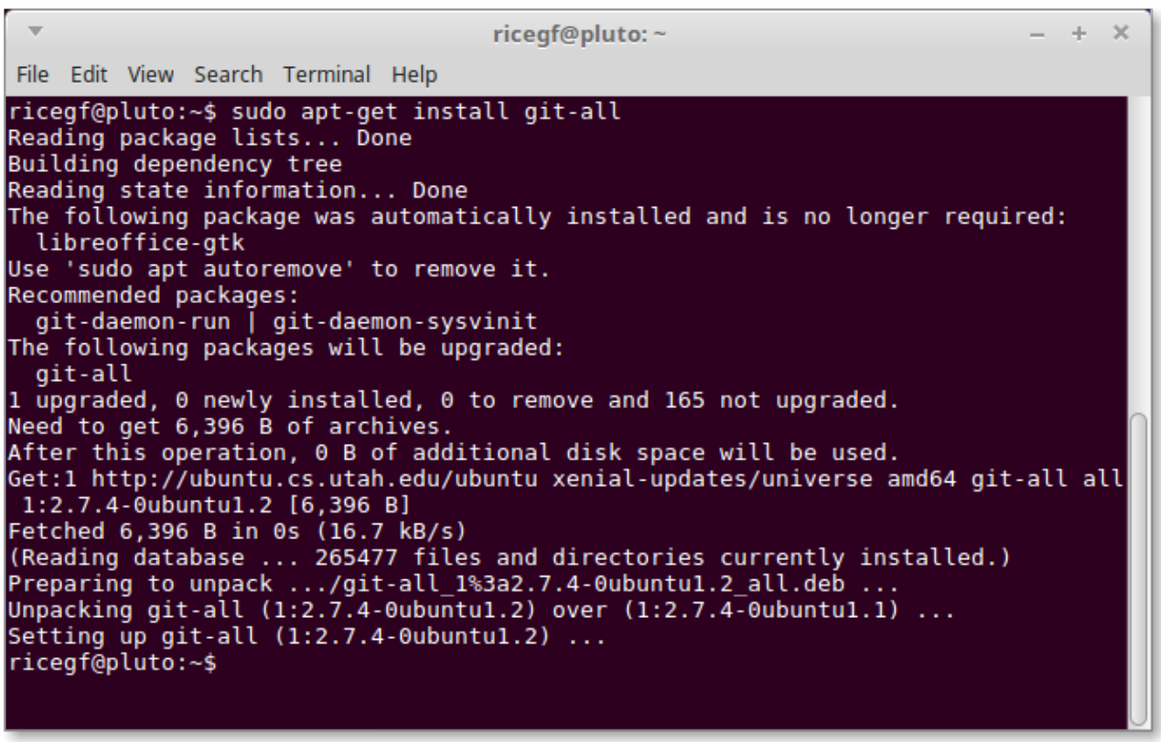
Using git in 5 Pages

(version control on the bash command line)

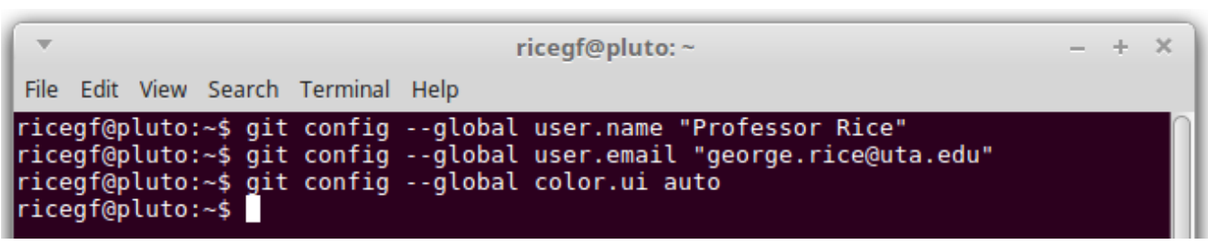
1 Installing git

1. On Linux bash, type **sudo apt-get install git-all** followed by your password.
   On Windows, download the latest version from https://git-scm.com/download/win.
   On Mac, download the latest version from http://git-scm.com/download/mac, or just try to use it and follow the installation prompts.



2. Configure git with the following commands, using your own name and email address. These commands allow git to credit you with your version management activity in your repositories, and also to use color-coded text in its output.

- **git config --global user.name "Professor Rice"**
- **git config --global user.email "george.rice@uta.edu"**
- **git config --global color.ui auto**



2 Getting help with git

1. **A list of git commands hyperlinked to a wealth of additional documentation and tutorials is available on-line at https://git-scm.com/docs/git.**

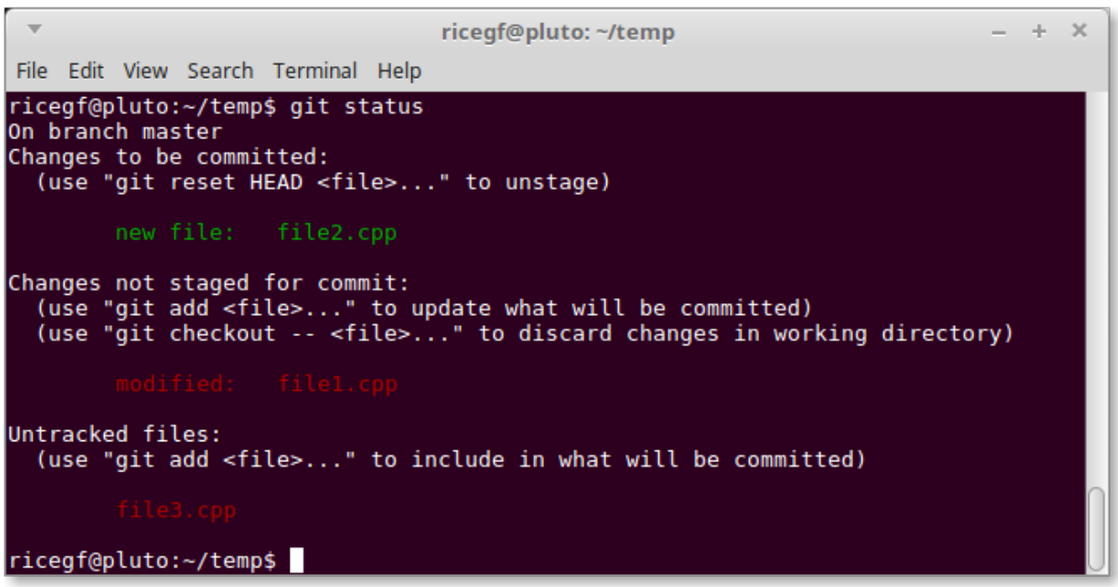
2. **git help** lists all common commands.
   git help [command] provides additional help on a command.

```
ricegf@pluto: ~
File  Edit  View  Search  Terminal  Help
ricegf@pluto:~$ git help
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Forward-port local commits to the updated upstream head
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
```

```
ricegf@pluto: ~
File  Edit  View  Search  Terminal  Help
GIT-CHECKOUT(1)                    Git Manual                    GIT-CHECKOUT(1)

NAME
       git-checkout - Switch branches or restore working tree files

SYNOPSIS
       git checkout [-q] [-f] [-m] [<branch>]
       git checkout [-q] [-f] [-m] --detach [<branch>]
       git checkout [-q] [-f] [-m] [--detach] <commit>
       git checkout [-q] [-f] [-m] [[-b|-B|--orphan] <new_branch>] [<start_point
>]
       git checkout [-f|--ours|--theirs|-m|--conflict=<style>] [<tree-ish>] [--]
 <paths>...
       git checkout [-p|--patch] [<tree-ish>] [--] [<paths>...]

DESCRIPTION
       Updates files in the working tree to match the version in the index or
       the specified tree. If no paths are given, git checkout will also
       update HEAD to set the specified branch as the current branch.

 Manual page git-checkout(1) line 1 (press h for help or q to quit)
```

3 Creating and adding files to a local repository

- Git commands consist of the word “git” and a second word such as “init”, “add”, or “commit”.
- Local information is stored in a hidden directory, and is **not backed up**. To protect your local repository, either use git with an online repository such as GitHub or back up your development directory using e.g., the Linux rsync or tar commands. See the last section for info.
- **git init** initializes a local repository for your files in the current directory. It is initially empty.
- **git status** reports info about tracked and untracked files in the current directory.



  - “On branch master” just means that you’re working on your main “branch”, named “master”. Don’t worry about it until you start using branches, for which see the last section.
  - “Changes to be committed” lists the files that git will store in its repository the next time you issue a **git commit** command.
  - “Changes not staged to commit” are files that git is tracking, and have changed, but are not ready to store in the repository – a **git add** command is needed if you want to store them.
  - “Untracked files” are files that git is ignoring – a **git add** command is needed to track them.
- **git add [file(s)]** tells git to begin tracking the listed files (if it wasn’t already), and be ready to store them in the repository at the next **git commit** command.
  - **git add -u** adds all files already added to the repository that have changed. It also notes removals (otherwise, you’d use **git rm [files()]** to remove files.
  - **git reset [file(s)]** is the ‘undo’ of add – it tells git to NOT store added file(s) in the repository at the next commit after all.
- **git commit -m “[message]”** stores all added files to the repository as a “commit”.
  - Think of a commit as a single version of ALL of your files in the repository. Each commit has a name (its “hash”), e.g., fecb2790d1c8a3a19fed445a017870aabcd577d2, of which only the first few characters are usually needed, e.g., fecb279.
  - **git commit -am “[message]** is **git add -u** and **git commit -m “[message]”** in one step.
- **git tag [tagname] [commit]** enables you to assign a more human-readable tagname to a commit (though the hash remains available). So **git tag 1.0.0 fecb279** allows you to type 1.0.0[1] (presumably a “version number”) wherever you would normally type fecb279 to identify a commit. *For our purposes here*, you can treat HEAD as a predefined tag for your *most recent* commit (you’ll understand more once you learn about *branching*).

1 Some projects put a “v” before a version number, e.g., v1.0.0. Follow your project’s standard. If you’re creating a project, don’t over-think it – just pick one!

4 Exploring the local repository

- **git log** shows all changes to the local repository.
  - **git log --oneline** shows just the name of the commit and your associated commit message. Add **--decorate** if you'd like to see tag names as well as the hashes.

```
ricegf@pluto: ~/temp
File  Edit  View  Search  Terminal  Help
ricegf@pluto:~/temp$ git log
commit 5335f1342c7c22bd1e5482dfaa42a43eccbb6aa0
Author: Professor Rice <george.rice@uta.edu>
Date:   Mon Aug 14 14:49:52 2017 -0500

    Add hello world

commit 44d37ad88add34dc534b2f75ee96a3c2dbb1f4c9
Author: Professor Rice <george.rice@uta.edu>
Date:   Mon Aug 14 14:36:13 2017 -0500

    First commit
ricegf@pluto:~/temp$ git log --oneline
5335f13 Add hello world
44d37ad First commit
ricegf@pluto:~/temp$
```

- **git diff** lists all changes between files in the repository (since the last commit) and the current files in the local directory. Unchanged lines of text are in white (shown for context), red lines of text preceded by a minus ("-") have been deleted, and green lines of text preceded by a plus ("+") have been added. Note that a *changed* line is represented by a deletion <u>and</u> an addition.

```
ricegf@pluto: ~/temp
File  Edit  View  Search  Terminal  Help
ricegf@pluto:~/temp$ git diff
diff --git a/file1.cpp b/file1.cpp
index 9b79668..f1e2f94 100644
--- a/file1.cpp
+++ b/file1.cpp
@@ -1,4 +1,5 @@
 #include <iostream>
+using namespace std;
 int main() {
-  std::cout << "Hello, World!" << std::endl;
+  cout << "Hello, World!" << endl;
 }
ricegf@pluto:~/temp$
```

  - You may also specify a starting and ending commit separated by two dots, as well as a specific filename, e.g. **git diff 5335f13..44d37ad file1.cpp**. This lists changes to the file file1.cpp between the first and second commit above, rather than the most recent commit ("tagged" as HEAD, remember?) and the current file in the directory.

```
ricegf@pluto: ~/temp
File  Edit  View  Search  Terminal  Help
ricegf@pluto:~/temp$ git diff 5335f13..44d37ad file1.cpp
diff --git a/file1.cpp b/file1.cpp
index 9b79668..e69de29 100644
--- a/file1.cpp
+++ b/file1.cpp
@@ -1,4 +0,0 @@
-#include <iostream>
-int main() {
-  std::cout << "Hello, World!" << std::endl;
-}
ricegf@pluto:~/temp$
```

5 Recovering earlier file versions and commits

- **git checkout [commit]** will switch the current directory to contain the files at the time that the commit was made. *This is for temporary viewing only, not for editing.* This is also an exception to our earlier definition of HEAD (called the "detached HEAD state") – here, HEAD is temporarily an alias for [commit]. Return to the most recent commit using **git checkout master**.
  - **git checkout [commit] [file]** discards all changes made to the file since the most recent commit, and replaces it with the file from the specified commit (remember, HEAD is the alias for the most recent commit). The changes discarded cannot be recovered.
  - **git reset --hard [commit]** discards all changes made to ALL files since the specified commit. All of the changes in all of the commits after [commit] cannot be recovered by any means. **Be careful!**
  - **git revert --no-edit [commit]..HEAD** creates a new set of commits (with a default message each) that "unrolls" the changes from each commit you've made, respectively, back to the specified commit. It is similar to **git reset --hard**, except that changes made since the specified commit are NOT discarded and can still be accessed via **git checkout**.

    You can do this manually with a single commit instead by using **git checkout [commit] .** (notice the period at the end), then **git commit -m "reverting to [commit]"**.

6 Submitting your homework with a git repository

1. In your homework directory, create a directory for your solution (e.g., **mkdir CSE1325-01**) and change to it (**cd CSE1325-01**).
2. Initialize a git repository (**git init**) for this homework assignment.
3. Create your full_credit directory (**mkdir full_credit**) and change to it (**cd full_credit**).
4. Develop your solution in this directory using **git add** and **git commit** each time you have any version of your code worth not losing. That's probably no less often than every 15 minutes! Also place required screenshots in this directory (**gnome-screenshot -a**), which do **NOT** need to be added to git (though you may if you like).
5. Repeat for all bonus levels, e.g., **mkdir ../bonus && cd ../bonus** and develop your solution.
6. Change to your solution directory CSE1325-01 (**cd ..**) and run **zip -r CSE1325-01.zip .** (notice the period at the end). The file CSE1325-01.zip is created, containing everything you need. Then submit CSE1325-01.zip to Blackboard as your solution. The grader will be able to unzip this file to their disk and *begin using git there exactly where you left off* to more effectively grade your solution.

7 Expanding your git Skills

The git suite has become the most common version control tool among software developers. **You need to know git well.** You can learn more about git's capabilities, including branching, merging, stashing, tagging, pulling, and pushing, via numerous on-line resources. Here are a few to get you started.

- For readers, the (free!) official book is at https://git-scm.com/book. If you're feeling old school, you can buy a paper copy from major book sellers.
- If you like videos, try https://www.youtube.com/githubguides.
- If you prefer interactive tutorials, try the short one at https://try.github.io/levels/1/challenges/1.